Aalborg Universitet

An Empirical Analysis of Cyber Deception Systems

Srinivasa, Shreyas

Publication date:
2023

Document Version
Publisher's PDF, also known as Version of record

[Link to publication from Aalborg University](#)

AN EMPIRICAL ANALYSIS OF CYBER DECEPTION SYSTEMS

BY
SHREYAS SRINIVASA

DISSERTATION SUBMITTED 2023

AALBORG UNIVERSITY
DENMARK

An Empirical Analysis of Cyber Deception systems

Ph.D. Dissertation
Shreyas Srinivasa

Dissertation submitted March 03, 2023

Abstract

The exponential growth in digitalization and the demand for context-aware processing have led to a rise in Internet-connected services, thereby increasing the risk of cyber attacks. Recent observations on the prevailing strategy of cyberattacks suggest the significance and high impact. The strategies employed by threat actors have matured to be more discrete, audacious, and impactful, targeting any form of digital entities ranging from human wearable devices to low earth orbit satellites. With such advancements, detecting such malicious entities is becoming increasingly challenging.

Defensive security is a discipline that deals with detecting, protecting, and preventing malicious processes with security systems (e.g., viruses, worms, spyware, trojan, and adware). Enterprise infrastructure uses defensive security elements such as Intrusion Detection Systems and firewalls to identify or limit suspicious traffic into the network. However, these systems do not provide an overview of the adversary techniques and methods that can be used to model attacker behavior or to determine the threat landscape. Moreover, these layer-3 systems are either highly dependent on signature-based detection methodology, which requires a constant update of its database, or on anomaly-based detection, thereby limiting the detection of novel actors.

Defensive security solutions or tools are used to detect, track and prevent security incidents. The data gathered from these tools and solutions are further analyzed to determine any targeted attacks or threats against the organization. This curated dataset can be helpful not only to the organization that collects and processes it and several other teams from other organizations to prepare their defenses. Cyber Threat Intelligence is the process of identifying potential cyber threats by analyzing data from multiple sources and understanding the tactics, techniques, and procedures used by the adversary. Deception-based systems like honeypots are excellent data sources for threat intelligence data. Honeypots can gather new attack vectors and offer fewer false positives. However, deception-based systems must be carefully designed, implemented, and deployed to increase their purpose.

Malware like Mirai has exploited vulnerable and misconfigured systems in its malicious campaigns. Adversaries like Mirai used reconnaissance techniques to look for vulnerable systems that could be compromised and leveraged for performing attacks or

iii

penetrating the network. Identifying and securing such systems is critical to limiting exploitation. Although it is known that adversaries use Internet-scanning techniques to find vulnerable services, they can be used as a defensive approach to proactively identify and secure them. Internet security measurements perform Internet-wide scans to identify misconfigured systems exposed to the Internet. The measurements further provide insights into the vulnerable landscape and the adoption of secure configurations.

In this thesis, I evaluate deception-based defensive security solutions by performing an empirical analysis that explores the current state, challenges, and possible solutions for their applicability. The thesis mainly focuses on Honeypots, deception-based, defensive security systems that simulate a target system and are designed as a trap mechanism to lure attackers. The attack data captured is analyzed to study the attacker's methods and behavior or to assess the threat landscape. In addition to attack data from honeypots, Internet security measurements are performed to enrich our data and establish a causal relationship between compromised systems being leveraged for launching attacks on the Internet.

The main scientific contributions of the thesis include performing an empirical analysis of the state-of-the-art deception systems to identify their limitations and extend the findings by proposing techniques that undermine their feasibility. Following the results from the analysis, we propose methods to address these limitations and conduct studies that evaluate our proposed techniques by deriving key results and curated datasets.

Ultimately, we set out to explore how deception-based systems can be used to increase our defenses against malicious entities, who are often underestimated. During my research, I understood better what security researchers endure to detect malicious actors and the importance of collecting and processing data to identify potential threats. It is a cat-and-mouse game when it comes to cybersecurity. In this work, I deliberately switch sides to improve defensive strategies.

Resumé

Den eksponentielle vækst i digitaliseringen og efterspørgslen efter kontekstafhængig informationsbehandling har ført til en stigning i antallet af internet-tilsluttede enheder og tjenester, hvilket øger risikoen for cyberangreb. De seneste trends indenfor cyberangreb viser at angreb kan få store konsekvenser for både virksomheder og samfund, og at aktørerne bag truslerne anvender stadigt mere modne og avancerede strategier: De er diskrete, dristige og virkningsfulde, rettet mod enhver form for digitale enheder lige fra små bærbare enheder på mennesker, til satellitter i lavt kredsløb om jorden. Med sådanne fremskridt bliver det stadig mere udfordrende at opdage sådanne ondsindede enheder. Forskellige typer af angreb er i konstant udvikling for at udforske og udnytte sårbare digitale tjenester med det formål at bryde deres fortrolighed, integritet og tilgængelighed.

Defensiv sikkerhed er en disciplin, der beskæftiger sig med detektion, beskyttelse og forebyggelse af ondsindede processer (f.eks. vira, orme, spyware, trojan, adware). Virksomheders infrastruktur bruger typisk defensive sikkerhedselementer såsom intrusion detection systems og firewalls til at identificere eller begrænse mistænkelig trafik ind i netværket. Disse systemer giver dog ikke et overblik over modstandernes teknikker og metoder, der kan bruges til at modellere angriberens adfærd eller til at bestemme trusselslandskabet. Desuden er disse lag-3-systemer enten stærkt afhængige af signaturbaseret detektionsmetodologi, som kræver en konstant opdatering af datbaser, eller af anomali-baseret detektion, hvilket begrænser detektionen af nye aktører og angrebsformer.

Defensive sikkerhedsløsninger eller værktøjer bruges til at opdage, spore og forhindre sikkerhedshændelser. Dataene indsamlet fra disse værktøjer og løsninger analyseres yderligere for at bestemme eventuelle målrettede angreb eller trusler mod organisationen. Dette kuraterede datasæt er ikke kun nyttigt for den organisation, der indsamler og behandler det, men også for andre organisationer i forhold til at forberede deres forsvar. Cyberthreat Intelligence er processen med at identificere potentielle cybertrusler ved at analysere data fra flere kilder og forstå de taktikker, teknikker og procedurer, der bruges af modstanderen. Deception-baserede systemer som honeypots er fremragende datakilder til threat intelligence. Honeypots kan samle nye angrebsvektorer og resul-

tere i færre falsk positive detektioner. Imidlertid skal deception-baserede systemer være omhyggeligt designet og implementeret for at udfylde deres formål.

Malware som Mirai har udnyttet sårbare og forkert konfigurerede systemer i sine ondsindede kampagner, og der er blandt andet brugt rekognosceringsteknikker til at lede efter sårbare systemer, der kunne kompromitteres og udnyttes til at udføre angreb eller trænge ind i netværket. At identificere og sikre sådanne systemer er afgørende for at begrænse angribernes muligheder for at udnytte dem. Selvom det er kendt, at modstandere bruger internet-scanningsteknikker til at finde sårbare tjenester, kan de også bruges som en defensiv tilgang til proaktivt at identificere og sikre dem. Internet-sikkerhedsmålinger udfører globale scanninger for at identificere forkert konfigurerede systemer, der er eksponeret mod internettet. Målingerne giver yderligere indsigt i såvel angrebsflader som udbredelse af sikre konfigurationer.

I denne afhandling evaluerer jeg deception-baserede defensive sikkerhedsløsninger ved at udføre en empirisk analyse, der kortlægger den nuværende sitauation samt ud-fordringer og mulige løsninger for deres anvendelighed. Afhandlingen fokuserer hoved-sageligt på Honeypots, hvilket er deception-baserede defensive sikkerhedssystemer, der simulerer rigtige systemer, der kan være mål for angreb. Honeypots er designet som fælder der lokker angribere til. De registrerede angrebsdata analyseres for at studere angriberens metoder og adfærd, og/eller for at vurdere trusselslandskabet. Ud over angrebsdata fra honeypots udføres internetsikkerhedsmålinger for at berige vores data og etablere en årsagssammenhæng mellem kompromitterede systemer, der udnyttes til at iværksætte angreb på internettet.

Afhandlingens vigtigste videnskabelige bidrag omfatter en empirisk analyse af avancerede deception-baserede systemer, for derigennem at identificere deres begrænsninger. Dernæst undersøges nye teknikker, der kan bruges til at svække honeypots og deres anvendelse. Med udgangspunkt i disse undersøgelser og analyser foreslår vi metoder til at adressere disse begrænsninger og udføre undersøgelser, der evaluerer vores foreslåede teknikker. Udover resultaterne præsenteres også nye datasæt, der er brugbare for det videnska-belige community.

Ultimativt ville vi gerne undersøge, hvordan deception-baserede systemer kan bruges til at øge vores forsvar mod ondsindede aktører, som ofte undervurderes. I løbet af min forskning forstod jeg bedre, hvordan ondsidede aktører i cyberspace opererer, og hvordan de kan opdages, herunder vigtigheden af at indsamle og behandle data for at identificere potentielle trusler. Det er et cat-and-mouse-game, når det kommer til cybersikkerhed. I dette arbejde skifter jeg bevidst side for bedre at forstå de ondsidede aktører, og derigennem forbedre defensive strategier.

Acknowledgments

Dedication

First and foremost, I want to emphasize that this section does not justify the support I received throughout my thesis from my family, friends, supervisors, and colleagues. I want to express my sincere gratitude to all of you. Please consider this page as a humble token of my gratitude for all the support you have provided. I also take this opportunity to apologize for my ignorance during my thesis period towards certain things and situations, stating that it was not deliberate. The last three years have been challenging; we have all endured it with great strength and patience.

The decision to pursue this thesis would not be possible without the guidance of my parents (Mrs. Shantha and Mr. Srinivasa Sastry), sister (Mrs. Sahana Srinivas Page), and support from my lovely wife, Pooja Dixit. My parents, sister, wife, and I come from humble backgrounds and could have never imagined what we are today without the education and support received from our community. I thank my community for all encouragement over the years. My wife, who has always been my best buddy, has constantly motivated and encouraged me during this thesis. Together, we have faced several challenges during the last three years; without her support, it would be impossible to face them. She has helped me transform and channel my thoughts and, most importantly, has been patient. With the course of this thesis, we became proud parents and welcomed our daughter Ameya Sastry. Ameya is a bundle of joy that fills me with energy and positivity. She kept me motivated with her gentle pats on my back when I whispered to her about issues. Like her mother, she patiently listened to all my irrelevant blabber and merrily accompanied me to conferences. Being a parent changed my perspective, and I thoroughly enjoy every moment.

I want to thank my dearest friends, who are my lifelines. They make us feel at home, away from home. I received immense support, and I am incredibly grateful to have them in my life. The list of friends is long, and mentioning only a few here would be irrational. I was fortunate enough to make good friends everywhere I lived for a considerable period of my life. I thank them for their continued support and sincerely appreciate your inclusiveness in these challenging times.

My journey at Aalborg University would never be possible without the advice from my co-supervisor Assoc. Prof. Dr. Emmanouil Vasilomanolakis (Manolis). It was from him first that I heard about a suitable opportunity. I want to express my gratitude to my supervisor Prof. Dr. Jens Myrup Pedersen, who has been a constant mentor. I admire his positivity, calmness, and, most importantly, his resistance to cold weather! Both my supervisors have been monumental in my thesis and in providing guidance. I thank them both for the trust and patience they invested in me, although they knew that my research entailed several complications in terms of security and ethicality.

I cherished my research stay at the University of Cambridge. The learnings from my stay have shaped new perspectives toward research. The long enchanting early morning walks around Cambridge fueled many thoughts and instill my passion. The compassion of Dr. Alice Hutchings and Dr. Richard Clayton spellbinds me. Every moment spent with Dr. Clayton made me realize my inner passion for research. I am forever grateful for this opportunity.

I am lucky to have wonderful colleagues who never tire of listening to my opinions. I especially want to thank Dimitrios Georgoulias and Rasmi-Vlad Mahmoud for their patience, support, and feedback. It is always true that many ideas originate with small discussions and over coffee.

I want to thank our students at Aalborg University, who trusted me with their project supervision. I have always enjoyed working on projects and learning mutually from them.

Lastly, I am forever in debt to the fantastic artists that have created inspiring and engaging music that has helped me stay focused during the thesis. My special thanks go to the bands Green Day, Iron Maiden, Metallica, and Indian Ocean. Thank you.

I dedicate this thesis to my parents, sister, wife, daughter, and friends.

I make a special dedication to all the dear family and friends we lost to the pandemic, war, and natural disasters. We miss you.

Thank you.

Contents

**F   Deceptive directories and "vulnerable" logs: a honeypot study of the
    LDAP and log4j attack landscape**   **151**

**G Open for hire: attack trends and misconfiguration pitfalls of IoT de-
    vices**   **169**

III   Epilogue   243

Conclusion   245

Directions for Future work   251

Thesis Details

Thesis Title: An Empirical Analysis of Deception-based systems in the network security of SMEs

Ph.D. Student: Shreyas Srinivasa

Supervisors: Prof. Jens Myrup Pedersen,
Aalborg University

Assoc. Prof. Emmanouil Vasilomanolakis,
Technical University of Denmark (DTU)

The main body of this thesis consist of the following papers.

[A] Srinivasa, S., Pedersen, J. M. & Vasilomanolakis, E., "Gotta catch 'em all: a Multistage Framework for honeypot fingerprinting", *Digital Threats: Research and Practice Association for Computing Machinery*, ACM DTRAP vol.1, no.1, month.1 2023 doi:https://doi.org/10.1145/3584976.

[B] Srinivasa, S., Pedersen, J. M. & Vasilomanolakis, E., "Towards systematic honeytoken fingerprinting", *International Conference on Security of Information and Networks (ACM SIN) . Ors, B. & Elci, A. (eds.). Association for Computing Machinery*, p. 1-5 5 p. 28

[C] Msaad, M., Srinivasa, S., M. Andersen, M., H. Audran, D., Uche Orji G, C. & Vasilomanolakis, E., "Honeysweeper: Towards stealthy Honeytoken fingerprinting techniques", *In: Reiser, H.P., Kyas, M. (eds) Secure IT Systems.NordSec 2022. Springer*, Lecture Notes in Computer Science, vol 13700. Springer, Cham. https://doi.org/10.1007/978-3-031-22295-5_6

[D] Srinivasa, S., Pedersen, J. M. & Vasilomanolakis, E., "RIoTPot: a modular hybrid-interaction IoT/OT honeypot", *ESORICS 2021: 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4–8, 2021, Proceedings, Part II*, Springer, Vol. 2. p. 745-751 7 p. (Lecture Notes in Computer Science, Vol. 12973).

[E] Srinivasa, S., Pedersen, J. M. & Vasilomanolakis, E., "Interaction matters: a comprehensive analysis and a dataset of hybrid IoT/OT honeypots", *2022, Annual Computer Security Applications Conference (ACSAC) 2022. Association for Computing Machinery*, vol. X, no. X, p. 742–755, 14 p.

[F] Srinivasa, S., Pedersen, J. M. & Vasilomanolakis, E., "Deceptive directories and "vulnerable" logs: a honeypot study of the LDAP and log4j attack landscape", *Mar 2022, Proceedings - 7th IEEE European Symposium on Security and Privacy Workshops, Euro S and PW 2022. IEEE*, p. 442-447 6 p.(IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)).

[G] Srinivasa, S., Pedersen, J. M. & Vasilomanolakis, E., "Open for hire: attack trends and misconfiguration pitfalls of IoT devices", *Nov 2021, IMC '21: Proceedings of the 21st ACM Internet Measurement Conference. Association for Computing Machinery*, p. 195-215 21 p.

[H] Srinivasa, S., Georgoulias, D., Pedersen, J. M. & Vasilomanolakis, E., "A Bad IDEa: Weaponizing uncontrolled online-IDEs in availability attacks", *Mar 2022, IEEE European Symposium on Security and Privacy, Workshop on Attackers and Cyber-Crime Operations. IEEE*, p. 82-92 11 p. 9799405. (IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)).

In addition to the main papers, the following publications have also been made.

[1] Lygerou, I., Srinivasa, S., Vasilomanolakis, E., Stergiopoulos, G & Gritzalis, D., "A decentralized honeypot for IoT Protocols based on Android devices", *6 Aug 2022, In: International Journal of Information Security*, 21, 6, p. 1211-1222 12 p., 1615-5270.

This thesis has been submitted for assessment in partial fulfillment of the PhD degree. The thesis is based on the submitted or published scientific papers which are listed above. Parts of the papers are used directly or indirectly in the extended summary of the thesis. As part of the assessment, co-author statements have been made available to the assessment committee and are also available at the Faculty. The thesis is not in its present form acceptable for open publication but only in limited and closed circulation as copyright may not be ensured.

Preface

This thesis came together at the Department of Electronic Systems at Aalborg University, under the kind supervision of Prof. Dr. Jens Myrup Pedersen (Aalborg University) and co-supervision of Assoc. Prof. Dr. Emmanouil Vasilomanolakis (Technical University of Denmark). The contributions of the thesis are represented in the form of a collection of papers. The included papers are peer-reviewed publications The layout of the published papers has been revised to align with the thesis document with the content unchanged.

The thesis is structured as follows. Chapter 1 provides an introduction, motivation, and outline of the thesis by highlighting the contributions of each paper. Part I covers a background for the preliminary concepts. Part II presents the collection of papers that form the core contribution of the thesis. The thesis is concluded in Part III with the conclusion and directions for future research.

<div align="right">

Shreyas Srinivasa
Aalborg University, March 3, 2023

</div>

Chapter 1

Introduction

Digitalization and automation have paved the way for modern evolution. The digital infrastructure has facilitated significant improvements in everyday life and all market sectors, thereby increasing efficiency and productivity. With this increasing digital adoption, there is a challenge to trust and security in this environment.

In light of the high demand for contextual data and accessibility, security has assumed minor importance.With the development of digital systems and networks, the complexity and discreetness of attacks further reformed. An advancement in defensive security strategies is imperative to address and limit the impact of cyberattacks. This is not at least the case for small and medium-sized enterprises (SMEs), where a recent Danish analysis [1] demonstrates that 40% of the companies have a digital security level that is too low compared to their risk profile. It is hence necessary to create awareness and emphasize the need for cybersecurity measures in SMEs.

The evolution of cyberattacks has led to the systemization of threats. Cyber threat actors vary based on their skills, capabilities, resources, and inspiration. They are classified into cybercriminals, nation-state actors, hacktivists, terrorist groups, thrill seekers, and insider threats [3]. The attack surface and threat landscape scope are extensive, with diverse threat actors. Protection against these various threats is not only complex but also highly challenging. Moreover, the threat actors evolve their strategies to evade protection and detection mechanisms.

Defensive security systems aim to detect, predict, and prevent cyber threats. Traditionally, detection systems are either signature or anomaly-based [2]. They rely on signatures of known threats or constantly check for traffic or system behavior anomalies through learning algorithms. However, due to the changing landscape, it is necessary to look for novel threats. Furthermore, it is essential to understand the behavior of the adversaries to characterize the threat actors. In firewalls and intrusion detection systems, the focus is on detecting and preventing known suspicious actors based on

a predefined set of rules, signatures, blocklists, or anomalies. These systems are not designed to identify novel threats and are limited to specific protocol services. A technique to identify unknown threats is to deceive the adversaries by deliberately exposing a system modeled as a trap to observe the attack strategies. Such strategies that lure attackers by setting up traps are categorized as deception-based systems.

Cyber Deception-based systems are modeled to simulate systems or services to deceive adversaries, and capture attacks [5]. The idea is to expose a seemingly vulnerable system to an active digital environment to detect suspicious activity. These systems aim to provide high fidelity in the logged activities with reduced false positives. Moreover, any interaction or activity with a system with no production value can be considered severe. An example of a cyber deception-based system is honeypots. Honeypots are decoys that simulate a target entity and are configured to capture any interaction with them [4] [6]. Traditionally honeypots were modeled with a focus on specific protocols or services and aimed at capturing malicious traffic within a network. However, the research on honeypots has dramatically advanced to capture complex threats by altering the deception layer. However, cyber deception systems have their limitations. Both adversaries and defenders are fighting to lead in the deception race. The use of deception-based techniques (e.g., phishing) by adversaries is known. Furthermore, offenders leverage the limitations of deception in honeypot-like systems to avoid any potential interaction, evade detection, and even exploit them. This poses a severe threat that challenges deception-based systems' goals and applicability. To gain a competitive advantage over adversaries, it is required to propose new techniques and address the limitations.

1 Thesis Statement

The thesis statement and the research questions are formulated as follows:

Cyber Deception systems aim at capturing adversarial attempts and strategies to facilitate further analysis and modeling of defenses against them. Due to the deceptive strategies employed, these systems are subjected to challenges against evolving adversarial strategies that intend to detect these systems. The value of a deception-based system lies in its ability to remain undetected. This thesis aims to accomplish the following research questions:

- **RQ1:** *What are the limitations of cyber-deception systems that affect their feasibility?*

- **RQ2:** *What techniques can be used to undermine the use of cyber deception?*

- **RQ3:** *How can the limitations of cyber deception be addressed to improve the applicability?*

2 Outline and Contributions

The thesis is structured into three parts. Part I introduces the preliminary concepts and provide background on cyber deception (Chapter 2), attacks against cyber deception (Chapter 3), and Internet security measurements (Chapter 4). Part II presents a three-fold collection of papers and their respective contributions. The papers are grouped by their contributions concerning the thesis statement and are listed below alongside highlighted contributions of each paper.

Honeypot Fingerprinting

This section of the Papers aims to address **RQ1 and RQ2** to find the limitations and techniques that undermine the feasibility of honeypots. The papers present research on attacks against cyber deception focused on honeypot fingerprinting.

- **Paper A** * (ACM DTRAP 2023) Honeypot fingerprinting techniques determine if the end system is a honeypot. On successful fingerprinting, the purpose of honeypots is undermined. We present novel methods for active honeypot fingerprinting (so-called probe-based). These are combined with a number of SotA and third-party (so-called meta-scan) fingerprinting techniques in the form of a multistage fingerprinting framework. We scan 2.9 billion IP addresses of the IPv4 space, discover 187 million IP addresses with relevant open ports and identify a total of 21,855 honeypots.

- **Paper B** (ACM SIN 2020) Honeytoken is an umbrella term for a subset of honeypots without protocol or system emulation. Instead, a honeytoken usually emulates some resource (e.g., a file or a username/password) that is part of a real system and triggers an alert whenever it is accessed or used. In this paper, we attempt a preliminary study on the possibility of fingerprinting honeytokens. We systematically classify the different honeytoken technologies and proceed by determining ways for their identification. Furthermore, we provide proof of concept experiments that demonstrate the feasibility of the proposed fingerprinting mechanisms. To our knowledge, this is the first paper to examine honeytoken fingerprinting.

- **Paper C** (NordSec 2022) Honeytokens are an important tool in proactively identifying data breaches and intrusion detection, as they raise an alert when a deceptive entity is accessed. In such deception-based defensive tools, it is vital that the adversary does not detect the presence of deception. However, recent research shows that adversaries may fingerprint honeypots and honeytokens. This work explores

---

*includes dataset

potential fingerprinting attacks against the most common open-source honeyto-
kens. Our findings suggest that an advanced attacker can identify the majority of
honeytokens without triggering an alert. We extend the fingerprinting techniques
based on our previous findings. Furthermore, we propose methods that help im-
prove the deception layer, the information received from the alerts, and the design
of honeytokens.

New paradigms in Cyber Deception

This section of the Papers aims to address **RQ3** and presents research into new paradigms
in cyber deception that address some limitations identified in RQ1 and RQ2. The re-
search presents RIoTPot, a hybrid interaction and modular honeypot, and performs
evaluation studies to understand the attack landscape and the influence of operational
parameters in honeypots.

- **Paper D** * (ESORICS 2021) Honeypots are often used as a proactive attack detec-
  tion mechanism and as a source of threat intelligence data. However, many honey-
  pots are poorly maintained and cumbersome to extend. Moreover, low-interaction
  honeypots are prone to fingerprinting attacks due to their limited emulation ca-
  pabilities. Nonetheless, low-interaction honeypots are essential for environments
  with limited resources. This paper introduces RIoTPot, a modular and hybrid-
  interaction honeypot for Internet-of-Things (IoT) and Operational Technology
  (OT) protocols mainly used in Industrial Control System environments.

- **Paper E*** (ACSAC 2022) In this paper, we extend and evaluate RIoTPot, a
  hybrid-interaction honeypot, by exposing it to attacks on the Internet and per-
  forming a longitudinal study with multiple evaluation parameters for three months.
  Furthermore, we publish the aforementioned study in the form of a dataset avail-
  able to researchers upon request. We leverage RIoTPot's hybrid-interaction model
  to deploy it in three interaction variants with six protocols deployed on both cloud
  and self-hosted infrastructure to study and compare the attacks gathered. We fin-
  gerprint the attacker's IP addresses to identify the type of devices participating
  in the attacks. Our results indicate that the honeypot interaction levels are im-
  portant in attracting specific attacks and scanning probes.

- **Paper F** (AD&D Euro S&PW 2022) This paper presents a study of attacks on
  the LDAP by deploying honeypots that simulate multiple profiles that support
  the LDAP service and correlating the attack datasets obtained from honeypots
  deployed by the *Honeynet Project* community.

---

*includes dataset

Internet Security Measurements

This section of the Papers aims to perform measurements to understand the attack landscape and evaluate the research from **RQ3**. The studies follow a measurement-based approach for analyzing and studying the impact of misconfigured systems deployed on the Internet. The studies aim at deriving key results and producing enriched datasets to facilitate further research in the community.

- **Paper G** * (ACM IMC 2021) In this paper, we perform an Internet-level IPv4 scan to unveil 1.8 million misconfigured IoT devices that may be exploited to perform large-scale attacks. These results are filtered to exclude a total of $8,192$ devices that we identify as honeypots during our scan. We deploy six state-of-art IoT honeypots to study current attack trends for one month. We gather a total of $200,209$ attacks and investigate how adversaries leverage misconfigured IoT devices. In particular, we study different attack types, including denial of service, multistage attacks, and attacks from infected online hosts. Furthermore, we analyze data from a /8 network telescope covering 81 billion requests towards IoT protocols.

- **Paper H** (WACCO Euro S&PW 2022) The increasing need for remote work and collaborative workspaces have led to the IDE-as-a-service paradigm that offers online code editing and compilation with multiple language support. In this paper, we show that a multitude of online IDEs do not run control checks on the user code and can be therefore leveraged by a botnet. We examine the concept of uncontrolled execution environments and present a proof of concept to show how uncontrolled online-IDEs can be weaponized to perform large-scale attacks by a botnet.

Part III concludes the thesis statement and discusses future work.

References

[1] erhvervsstyrelsen.dk. (2021) Digital sikkerhed i danske smv'er. https://erhvervsstyrelsen.dk/sites/default/files/2021-09/Digital%20sikkerhed%20i%20danske%20SMVer_Erhvervsstyrelsen_Sep2021_EUwebtilg%C3%A6ngelig02.pdf.

[2] C. Flynt, "intrusion detection," *login Usenix Mag.*, vol. 26, no. 3, 2001.

[3] V. Mavroeidis, R. Hohimer, T. Casey, and A. Jesang, "Threat actor type inference and characterization within cyber threat intelligence," in *2021 13th International Conference on Cyber Conflict (CyCon)*. IEEE, 2021, pp. 327–352.

[4] N. Provos *et al.*, "A virtual honeypot framework." in *USENIX Security Symposium*, vol. 173, no. 2004, 2004, pp. 1–14.

[5] C. Wang and Z. Lu, "Cyber deception: Overview and the road ahead," *IEEE Security & Privacy*, vol. 16, no. 2, pp. 80–85, 2018.

[6] L. Zhang and V. L. Thing, "Three decades of deception techniques in active cyber defense-retrospect and outlook," *Computers & Security*, vol. 106, p. 102288, 2021. [Online]. Available: https://arxiv.org/abs/2104.03594

Part I

Preliminary Concepts

Chapter 2

Cyber Deception Systems

Deception is one of the many traits living organisms exhibit to protect, pretend, and prey. In his book *The Art of War, (5 B.C)*, Sun Tzu states, "All warfare is based on deception" [5]. Conventional use of deception includes military, marketing, and magic. Some forms of deception include masking, repackaging, dazzling, mimicking, inventing, and decoying [2]. In cybersecurity, deception can be leveraged in both offensive and defensive techniques. In offensive security, cyber deception techniques are mostly limited to impersonation, while they can be leveraged to a wider scope in defensive systems.

Cyber deception in defensive security focuses on impersonating a seemingly vulnerable system to lure attackers. Honeypots are defensive cyber deception-based systems and can be defined as "A security resource whose value lies in being probed, attacked or compromised" [3]. The main components of a honeypot include deception and logging. The deception layer is responsible for simulating a service or a system. The logging component ensures that all the traffic between the adversary and the honeypot is captured in a structured format. Honeypots are classified into low, medium, and high interaction based on the deception layer and the interaction level they offer to the attackers. The low-interaction honeypots (li-HP) have a limited simulation of service and low operation costs. For example, a li-HP is limited to responding to specific requests or commands and may always be static. The medium-interaction honeypots (mi-HP) provide an extended simulation than the li-HP and may include a collection of services or a target device profile (e.g., a Windows 7 system). High-interaction honeypots (hi-HP) are actual systems that run full services as targets. These honeypots have higher operating costs and maintenance. To limit the operational costs, hi-HPs can be configured as virtual machines or containers to achieve the full system perspective. Hi-HPs may gather higher traffic on careful configuration than li-HP and mi-HPs.

The Honeynet Project offers several open-source honeypots specific to protocols, devices, or operational environments [1]. Most of the honeypots are either li-HP or

mi-HP. The honeypots from the Honeynet Project serve as an excellent resource for security training and awareness programs.

Honeypots capture malware and form a good source of threat intelligence data. As honeypots are non-production systems, there is no real reason to interact with them. Hence, fewer false positives exist, and all traffic can be considered suspicious. The attack data captured by honeypots can be used for analyzing the threat landscape and modeling. Furthermore, the captured data can be modeled to understand the attacker's behavior and techniques. Honeypots can be further classified based on their use in research and industry. Although honeypots can capture a multitude of attacks, most of the traffic is from automated attacks like bots. Moreover, honeypots can capture specific attacks like insider attacks if configured to operate within a network. With the simplicity that honeypots provide, they are used in research and industry studies to determine the attack landscape.

Honeytoken is an umbrella term for a subset of honeypots without protocol or system emulation. Instead, a honeytoken usually emulates some resource (e.g. a file or a username/password) that is part of a real system and triggers an alert whenever it is accessed or used [4]. For example, a honeytoken can be a *.docx* file containing an obfuscated script triggered when the file is opened. An advantage of honeytokens over traditional honeypots is that they operate with lower system resources and are simpler to manage. In addition, they are easy to generate and deploy. Honeytokens can indirectly detect the presence of diverse attack vectors (e.g., malware) and identify direct attacks like unauthorized access attempts. Due to their simple design and flexibility, honeytokens are popular and are used by system administrators [6].

Although honeypots have many advantages, they are prone to be vulnerable to certain attacks. We discuss the potential attack vectors that may undermine the value of honeypots and possibly exploit them in the next chapter. Other forms of cyber deception include tarpits, moving target defense, and honeynets [6]. In this thesis, we evaluate honeypots and present our research in Papers D, E and F.

References

[1] T. H. Project, "The honeynet project."

[2] N. Rowe, "A taxonomy of deception in cyberspace," in *International Conference on Information Warfare and Security*, 2006, pp. 173–181.

[3] L. Spitzner, "The honeynet project: Trapping the hackers," *IEEE Security & Privacy*, vol. 1, no. 2, pp. 15–23, 2003.

[4] ——, "Honeytokens: The other honeypot. 2003," *Internet: http://www. securityfocus. com/infocus/1713*, 2006.

[5] S. Tzu, "The art of war," in *Strategic Studies.* Routledge, 2008, pp. 63–91.

[6] L. Zhang and V. L. Thing, "Three decades of deception techniques in active cyber defense-retrospect and outlook," *Computers & Security*, vol. 106, p. 102288, 2021. [Online]. Available: https://arxiv.org/abs/2104.03594

Chapter 3

Attacks against Deception

Fingerprinting is the process of profiling or determining the end system through probing and analysis. Traditionally fingerprinting approaches are used to distinguish legitimate devices from adversarial systems and perform vulnerability assessments [2, 3]. Conversely, fingerprinting techniques can be used to determine the characteristics of the target machine that could be useful in identifying the potentially vulnerable points [1]. Tools like NMap and p0f are widely used in reconnaissance to determine the version of the services running on the end system [4, 7].

Fingerprinting techniques are classified into active and passive. Active fingerprinting involves using probes that interact with the target system to invoke specific responses that help infer the target system. Using this technique requires active interaction with the end system, which can be noisy. Passive fingerprinting involves examining data related to the end system and does not involve interaction. The data about the end system could be from network traffic or metadata from external databases.

Deception fingerprinting determines if the end system is a honeypot. As honeypots are not real systems and are simulation-based, they are limited in responses. Through selective active probing, adversaries can run commands that honeypots are known to have a static or limited response. Honeypot fingerprinting has recently been an active research area, and several approaches based on both active and passive techniques have been proposed [5, 6].

Honeypot fingerprinting can be leveraged by adversaries to avoid or limit their interaction with honeypots, thereby limiting their purpose. Moreover, adversaries can develop a list of known honeypots that bot campaigns can use as a blocklist. Attackers can further exploit misconfigured honeypot systems to escape into the production networks or leverage them to launch attacks on the Internet. In this thesis, we explore and propose honeypot fingerprinting techniques to address the limitations of honeypots and improve them. We present our work on honeypot fingerprinting in Papers A, B and C.

13

References

[1] L. G. Greenwald and T. J. Thomas, "Toward undetected operating system fingerprinting," in *Proceedings of the First USENIX Workshop on Offensive Technologies*, ser. WOOT '07. USA: USENIX Association, 2007.

[2] H. Jafari, O. Omotere, D. Adesina, H.-H. Wu, and L. Qian, "Iot devices fingerprinting using deep learning," in *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, 2018, pp. 1–9.

[3] A. Keliris and M. Maniatakos, "Remote field device fingerprinting using device-specific modbus information," in *2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2016, pp. 1–4.

[4] G. Lyon, "Nmap network mapper," 2021. [Online]. Available: https://nmap.org/

[5] S. Morishita, T. Hoizumi, W. Ueno, R. Tanabe, C. Gañán, M. J. van Eeten, K. Yoshioka, and T. Matsumoto, "Detect me if you. . . oh wait. an internet-wide view of self-revealing honeypots," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, IEEE.   Washington DC, USA: IEEE, 2019, pp. 134–143.

[6] A. Vetterl and R. Clayton, "Bitter harvest: Systematically fingerprinting low-and medium-interaction honeypots at internet scale," in *12th {USENIX} Workshop on Offensive Technologies ({WOOT} 18)*, 2018.

[7] M. Zalewski, "the new p0f: 2.0. 8," *http://lcamtuf. coredump. cx/p0f. shtml*, 2006.

Chapter 4

Internet Security Measurements

Internet, at its core, is an interconnected global packet-switched network that facilitates seamless communication through data. With the rise in digitalization and automation, there is an increase in Internet-connected services, focusing on accessibility and context awareness. This upsurge has entailed several security incidents due to misconfigurations and unpatched environments. Adversaries use probes that periodically scan the Internet to identify and infiltrate vulnerable services.

The advancement in networking infrastructure has led to faster accessibility and communication. Internet security measurements aim at implementing studies to monitor, observe and analyze the Internet infrastructure to identify possible security trends and implications [3]. Internet-wide scanning services like Shodan and Censys perform daily scans of the Internet to provide information on services exposed [1, 4]. Furthermore, with open-source scanning tools like ZMap, it is possible to complete scanning the Internet in less than an hour [2]. While adversaries effectively use Internet scanning to look for vulnerable systems, the technique can also be leveraged for reporting the findings to the administrators. Furthermore, the datasets obtained from Internet scanning can be analyzed to determine the attack landscape and risk analysis.

In this thesis, we conduct measurement-based studies to determine the attack landscape and to enrich our findings. Furthermore, we share the results from our measurement studies to facilitate further research and collaboration. In this thesis, we present our research and findings from the measurement studies in Papers G and H.

References

[1] Censys, "Censys search," 2021. [Online]. Available: https://censys.io/

[2] Z. Durumeric, E. Wustrow, and J. A. Halderman, "Zmap: Fast internet-wide scanning and its security applications," in *Proceedings of the 22nd USENIX Conference on Security*, ser. SEC'13.   USA: USENIX Association, 2013, p. 605–620.

[3] M. S. Pour, C. Nader, K. Friday, and E. Bou-Harb, "A comprehensive survey of recent internet measurement techniques for cyber security," *Computers & Security*, p. 103123, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404823000330

[4] SHODAN, "Shodan," 2021. [Online]. Available: https://www.shodan.io/

Part II

Papers

Deception Fingerprinting

Paper A

Gotta catch 'em all: a Multistage Framework for honeypot fingerprinting

Shreyas Srinivasa, Jens Myrup Pedersen, Emmanouil Vasilomanolakis

The layout has been revised.

Abstract

Honeypots are decoy systems that lure attackers by presenting them with a seemingly vulnerable system. They provide an early detection mechanism as well as a method for learning how adversaries work and think. However, over the last years a number of researchers have shown methods for fingerprinting honeypots. This significantly decreases the value of a honeypot; if an attacker is able to recognize the existence of such a system, they can evade it. In this article, we revisit the honeypot identification field, by providing a holistic framework that includes state of the art and novel fingerprinting components. We decrease the probability of false positives by proposing a rigid multi-step approach for labeling a system as a honeypot. We perform extensive scans covering 2.9 billion addresses of the IPv4 space and identify a total of 21,855 honeypot instances. Moreover, we present a number of interesting side-findings such as the identification of around 355,000 non-honeypot systems that represent potentially misconfigured or unpatched vulnerable servers (e.g. SSH servers with default password configurations and vulnerable versions). We ethically disclose our findings to network administrators about the default configuration and the honeypot developers about the gaps in implementation that lead to possible honeypot fingerprinting. Lastly, we discuss countermeasures against honeypot fingerprinting techniques.

1 Introduction

Honeypots are decoy systems whose only value lies in being probed, attacked, and compromised. They attempt to lure attackers in, to provide an early warning system, and act as a method for understanding the adversaries' mindset and determining new attack trends [1]. Honeypots are not a stand alone security mechanism but rather important supplements to existing infrastructure (e.g. firewalls and Intrusion Detection Systems (IDS)). Nevertheless, they offer a unique attack understanding and perspective, while exhibiting a very low number of false positives. The latter is due to the fact that any communication towards a honeypot is considered hostile; i.e. benign users have no reason to contact a honeypot.

Honeypots are commonly classified based on the interaction level they offer to the adversary. This results in low, medium and high-interaction honeypots [2]. While the first two categories offer different levels of *emulation* of protocols, the latter (i.e. high-interaction) describes real world systems. High-interaction honeypots are too expensive to maintain and significantly less used than low/medium interaction; hence, we consider them out of the scope of this article. Over the years, low and medium interaction honeypots have been designed and developed to emulate the majority of commonly used protocols. These include SSH (e.g. Kippo [3] and Cowrie [4]), Telnet (e.g. Cowrie [4]), HTTP (e.g. Glastopf [5]), FTP, SMB (e.g. Dionaea [6] and HosTaGe [7]) and also

Industrial Control Systems (ICS) protocols like Modbus and S7 (e.g. Conpot [8] and HosTaGe [9]). Other research-based honeypots include AmpPot [10] that simulates UDP-based protocols like NTP, SSDP which can be abused for DRDoS attacks and RIoTPot [11] a modular and hybrid interaction honeypot that aims at operating a honeypot at ternary interaction levels.

One of the key success criteria for a honeypot is that it is indistinguishable from a real system. This can be translated to the following axiom: *if a honeypot can be easily identified as such, then its value is significantly decreased.* The reason for this is that an adversary can potentially either evade honeypots (e.g. perform reconnaissance and add a blocklist of IP addresses into their malware, to avoid honeypots and reduce the risk of detection [12]) or attempt to take them down (e.g. via a Distributed Denial of Service (DDoS) attack). Note that modern malware (e.g. Hide'n Seek [13]) already include hard-coded IP addresses (e.g. belonging to known security agencies) that are blocklisted from all communications. Honeypot fingerprinting is the process of revealing that a seemingly vulnerable system is, in fact, a honeypot.

In this article, we perform a comprehensive analysis of honeypot fingerprinting techniques. For this, we present a holistic framework that includes a number of novel fingerprinting methods along with all major state of the art techniques. Among others, we propose a new protocol handshake fingerprinting component, a static Transport Layer Security (TLS) certificate method and a Fully Qualified Domain Name (FQDN) check. Furthermore, we present the results of extensive honeypot identification scans over the Internet for 9 prominent honeypot implementations. Our results come as an independent confirmation of previous studies ([14, 15]) but also as a step forward to a more holistic study of honeypots. In particular, due to the multistage checks that our framework performs, we argue that the presented results have a very low probability for false positives. Moreover, we present several insights for IP addresses that are not marked as honeypots, but are likely to be real vulnerable systems. Lastly, we discuss ethical considerations and possible countermeasures against fingerprinting. The core contributions of this article can be summarized as follows:

- We present novel methods for active honeypot fingerprinting (so-called probe-based). These are combined with a number of SotA and third-party (so-called meta-scan) fingerprinting techniques in the form of a multistage fingerprinting framework. We scan 2.9 billion IP addresses of the IPv4 space, discover 187 million IP addresses with relevant open ports and identify **a total of 21,855 honeypots**.

- We showcase that out of the 21,855 identified honeypots, **third-party techniques can only reveal** 33.9% **of the total honeypot population**. On the contrary, we show that most of the honeypots can be identified via our probe-based methodology with less false positives.

- As a side finding, we identify more than 355,000 potentially vulnerable entities

(i.e. SSH and FTP servers) that are not honeypots and appear to use trivial passwords and/or are susceptible to high-severity vulnerabilities.

The rest of the article is structured as follows. We propose our framework for honeypot fingerprinting in Section 2. Section 3 presents our evaluation. Section 4 discusses ethical considerations, fingerprinting countermeasures and the limitations. In addition, in Section 5 we discuss countermeasures against fingerprinting. Section 6 presents the related work on honeypot fingerprinting research. We conclude the article in Section 7.

2 Multistage Honeypot Fingerprinting Framework

Researchers classify fingerprinting techniques as active and passive, based on attacker-honeypot interaction [16]. Active-fingerprinting involves creating specific probes and using them to querying the target system to collect as much data as possible. On the contrary, passive-fingerprinting makes use of available data about the target system for further analysis to determine information about the target.

In the following, we attempt to examine methods in both the active and passive spectrum in Section 2.1 . On the one hand, we assume that attackers prefer passive methods since they come with multiple benefits. Mainly, they are stealthier (i.e. no direct communication to the honeypot is needed) and easier to use (e.g. systems like Shodan [17] already exist and offer an Application Programming Interface (API) for such purposes). On the other hand, our hypothesis is that active approaches can identify a much broader set of honeypots. We propose the novel framework (see Section 2.2) that utilizes both active (Probe-based fingerprinting) and passive fingerprinting(Metascan-based fingerprinting) techniques to fingerprint honeypots deployed on the Internet. The aim of the proposed framework is to systematically fingerprint honeypots with multiple sequential checks to reduce false positives. In comparison to state of the art (c.f. Section 6), we employ novel probing methods that include certificate checks, protocol handshake and metascan methods that check for Internet Service Provider (ISP) and cloud hosting information. The framework is further automated for all the checks involved in each fingerprinting technique that helps in automated transition to stages during the scanning process.

2.1 Overview

This section provides an overview of the proposed multistage fingerprinting framework and the detection techniques.
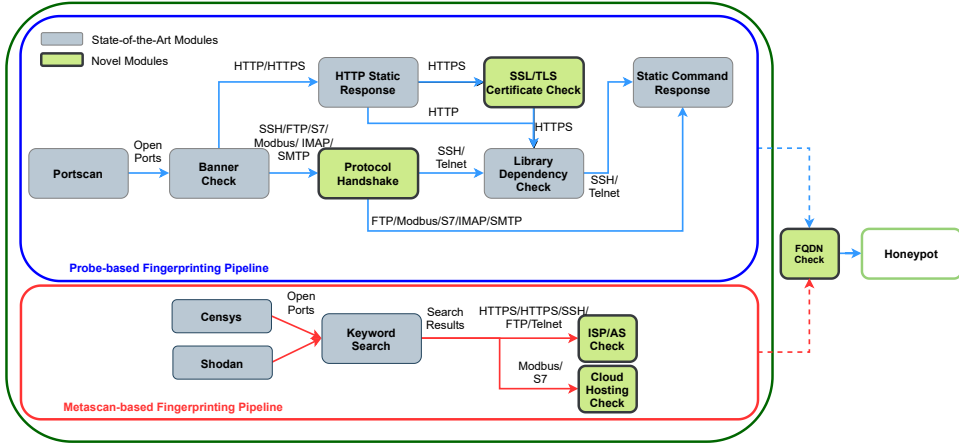
Fig. A.1: Multistage Framework for Honeypot Fingerprinting

Probe-based fingerprinting

Probe-based fingerprinting involves the creation of queries to derive fingerprinting information and involves direct interaction with a system. These methods focus on leveraging the data from responses and classifying the target machine based on fingerprinting identifiers. The information may include system-specific unique identifiers like the Initial Congestion Window (ICW) or the Retransmission Time Out (RTO). Several fingerprinting tools like NMap [18], XProbe2 [19], Metasploit [20], and Hydra [21] utilize probe-based methods to determine the Operating System (OS) and the protocol versions of the target systems. For example, these tools rely on banners advertised and the *Transmission Control Protocol (TCP)* header information to determine the underlying OS. The database of the scanning tool stores the identifiers that are specific to some OS. The identifiers help compare parameter values obtained through probing for determining the OS. The fingerprinting probes derive multilevel system information at network-level, application-level, and the system-level. The integration of the information received from different levels improves detection accuracy.

Metascan-based fingerprinting

Metascan-based fingerprinting is a form of passive-fingerprinting that leverages the known information about the target system without direct interaction. The technique uses the IP address and performs a search on Internet mass-scan engines (e.g. Shodan [17] and Censys [22]) to obtain attributes like hosting provider and the Internet Service Provider (ISP). The data obtained through metascan can be leveraged for fingerprinting purposes. For example, if the target system has the TCP port 502 (i.e.

the Modbus protocol default port) open and the IP address is attached with a network assigned to a university or research facility, this might act as an indication that the target system is a research honeypot. Similarly, if a cloud provider hosts the aforesaid system, it is likely a honeypot because ICS are physical devices that are deployed in an industrial network and are unlikely to be hosted by a cloud provider.

Mass-scan engines like Shodan and Censys crawl the Internet IP-space daily to find vulnerable systems exposed to the Internet. They also store system- and network-specific information about the exposed systems like banners, *HTTP* content, certificate, open ports, and services. Furthermore, they provide metadata like the ISP, Autonomous Systems (AS), and geo-location of the systems. Metascan fingerprinting techniques rely on essential information about target systems for the fingerprinting process. Such information can be obtained through the APIs offered by Shodan and Censys. Hence, the mass-scan engines can act as a substitute for the probe-based checks and provide the information required without interacting with the target systems.

2.2 Framework

We construct a framework that combines both probe- and metascan-based methodologies. The framework is automated for the sequential checks for the probe-based and the metascan-based techniques. The probe-based technique uses methods that involve direct interaction with the target machine to fetch fingerprinting-information, while the metascan-based techniques use methods that involve no direct interaction with the target systems. In particular, the latter uses information derived from the Shodan and Censys mass-scan engines. Some mass-scan engines employ banner-based fingerprinting to fingerprint device types. For example, Censys [22] uses the Recog engine [23] to detect device types using the information received by probing. The methods used in the proposed metascan pipeline are novel specifically towards honeypot fingerprinting. The novel methods of using the information about the ISP and checking if the instance is on a cloud environment assist us in gathering additional information that can be leveraged for the fingerprinting process. In addition, these methods help in reducing of false positives from the results. We term the target system considered for fingerprinting as an *instance* for the rest of our article.

Figure A.1 shows the proposed multistage fingerprinting framework. The framework contains two independent main pipelines of *Probe-based* and the *Metascan-based* techniques. The *Probe-based Pipeline* has multiple stages that are represented as boxes in the figure, with each stage aiming at fingerprinting the instance at various levels (i.e. network-level, application-level, system-level, protocol-level, implementation-level, and configuration-level). The boxes are color-coded to gray and green for further classification. The gray boxes denote that the stages refer to the state of the art, while the green boxes represent the novel methods. The novel methods are comprised of persistent checks that enrich the likelihood of the instance to be a honeypot. In the *Metascan-*

based Pipeline, the stages represent systematic checks referring to passive-fingerprinting techniques and data analysis. Overall, an *instance* is only labeled as a *honeypot* if all (relevant) components of the respective pipeline concur.

Probe-based Fingerprinting Pipeline

The Probe-based fingerprinting pipeline consists of *seven* probing stages. The instances under evaluation transition into the next stage, based on the underlying application service protocol. The probes from each stage fetch information which is then analyzed to derive whether the instance is a honeypot.

Portscan   The pipeline begins by performing a scan on the Internet for open ports specific to the services emulated by the honeypots. Our framework utilizes ZMap [24] for this process (alternatively one could use Masscan [25]). The search results consist of a list of instances having these ports open to the Internet. Recent research reveals faster Internet services across all ports by running a predictive network that learns from extremely small sample sizes [26]. Augmenting such frameworks could improve this stage and as a result the pipeline.

Banner Check   The results of the portscan are further processed in the banner check stage. In this stage, the probes check the banner advertised by the end system with static banners offered by honeypot implementations. Honeypot implementations offer a limited set of banners or even static banners that, in some cases, do not match the actual banners advertised by the services running on the underlying OS. As these banners are hard-coded, they can be matched against a list of known honeypot banners. We use the extended banner grab utility offered by ZMap to fetch banners from instances [27]. SotA honeypot fingerprinting techniques by Vetterl et al. [14] and Morishita et al. [15] employ banner based fingerprinting to detect honeypots. We combine this knowledge (see Table A.10 in the Appendix) to construct a holistic banner list for our framework. The results of this stage provide us with a list of instances and their banners. The instances that match the banners advertised by honeypots progress into the next stage based on the underlying protocol. For the instances that do not match the banners, we perform a vulnerability check that determines the number of vulnerable systems on the Internet with specific protocol versions (see Section 3.3 in the evaluation). Fingerprinting honeypots only with banner checks is prone to false positives, and therefore, we subject the instances to further protocol level and system-level checks.

***HTTP* Static Response**   The filtered instances with *HTTP* and *HTTPS* service identified in previous stages are checked for static *HTTP* content in their response. Honeypots emulating the web services offer limited content by default which can be identified. The instances are queried with an *HTTP GET* request to fetch the content

and then match the static default content offered by the honeypots. Table A.11 in the Appendix shows the *HTTP* response returned by honeypots. Upon match of static content, the instance continues to the next stage. This technique was adapted from [14, 15] for fingerprinting *HTTP*-based honeypots.

SSL/TLS Certificate check   This stage compares certificate-specific attributes to known values from default certificates provided by honeypots. Some honeypots offer hard-coded TLS certificates that can be leveraged to fingerprint honeypot instances. Though there is a change of fingerprint on each certificate, attributes like issuer and provider remain static. We add this stage, particularly for honeypots that use any certificates. During our study, we observe that the Dionaea honeypot contains a certificate issued by a provider name that is consistent in all its deployments [28]. The *SSL/TLS Certificate check* component stage checks the attributes *certificate issuer* and the *common subject name* of the certificate retrieved from web servers to identify Dionaea honeypots on the Internet. The stage can be extended further to include other honeypots that use any certificates. Algorithm 1, in the Appendix, represents the pseudo-code block that checks an instance for Dionaea's default certificate parameters.

Protocol Handshake   The communication of systems over any network is established upon the negotiation of various communication parameters, before building a channel. Honeypots offer limited emulation and communication preferences. This limitation is caused due to the honeypot design or the utilization of certain protocol emulation libraries. We exploit this limitation of deviated behavior, in the protocol negotiation process, to identify honeypots. First, we observe the deviation in the negotiation process and the limited availability of parameters by establishing communication with in-house lab honeypots (see Section 3.2). We develop probes that attempt to establish a connection through limited parameters and observe the response for deviation for all emulated services. Table A.1 summarizes the responses for certain negotiations of protocols. We observe protocol handshake deviations that cause the acceptance of malformed request packets, return limited options for negotiation, or disconnect the session with an arbitrary message that is different from non-honeypot implementations. Algorithm 4 in the Appendix describes the protocol handshake checks. The algorithm accepts a list of instances with their IP address and port. For each instance, a request is sent for session initiation with specific parameters. The response is analyzed for deviations that match the response from honeypots. Upon match, the flag *isDeviated* is set and such instances progress to the next framework stage.

Library Dependency Check   Emulations in low and medium-interaction honeypots are often developed by referring to external libraries. Libraries offer limited emulation capabilities based on their design and frequently return static values in certain queries.

Honeypot	Protocol	Request	Response
Kippo	SSH	SSH-2.0-OpenSSH \n\n\n\n\n\n\n\n\n\n	"bad packet length *" or "protocol mismatch\n"
Cowrie	SSH	1. SSH-2.0-OpenSSH_6.0p1 Debian-4+deb7u2 \n 2. SSH-2.0-OpenSSH_6.0p1 Debian-4+deb7u2 \n	"protocol mismatch\n"
Gaspot	Telnet	I30100	9999FF1B
Conpot	S7	"H", "0300002102f0803207000000000008 \n 00080001120411440100ff09000400110001"\n	0x32
Conpot	Modbus	function_code': None, 'slave_id': 0, \n 'request': '000000000005002b0e0200' \n	Disconnection
Glastopf	HTTP	GET /HTTP/1.0	Server: BaseHTTP/0.3 Python/2.5.1
Dionaea	HTTP	GET /HTTP/1.0	Server: nginx
Amun	HTTP	GET HTTP/1.1	Server: Apache/1.3.29
MTPot	Telnet	WILL (251) Linemode	Won't (252) Linemode

Table A.1: Protocol handshake deviation

Honeypot	Protocol	Library	Updated
Kippo	SSH	TwistedConch	May2015
Cowrie	SSH	TwistedConch	May2018
MTPot	Telnet	telnetsrv	Dec2012
Cowrie	Telnet	TwistedConch	May2018
Dionaea	HTTP	custom	Sep2016
Glastopf	HTTP	BaseHTTPServer	Oct2016
Conpot	HTTP	BaseHTTPServer	Mar2018

Table A.2: Library references in honeypots

Furthermore, some libraries referred by honeypots have not been well maintained. Vetterl et al. have leveraged the use of libraries in honeypots to craft specific probes that return static values [14]. This static information can be used to fingerprint the honeypots. Table A.2 shows the libraries used by many well-known honeypots for the service emulation and their last update. Leveraging the aforementioned static implementation and limited emulation, we develop the probes based on [14] that request for specific information from the end-systems. We compare the response to known static responses from the honeypots. We proceed in case of a match. In honeypots, protocol handshake is also dependent on the library used for emulation purposes and hence these two stages are intertwined. Nevertheless, we use this check to check for additional dependencies that can signal static behavior.

Static Command Response Due to the nature of honeypots, developers are compelled to implement some services with static responses or disconnect the communication for specific command requests. For instance, some honeypots attempt to overcome such issues via a static response (e.g. *"Invalid Command"*), or disconnect with the user. We leverage this gap in implementation for having probes request systems with commands

to expect known static responses from the end systems. Table A.14 shows the static response returned by honeypots for specific commands by our probes.

Metascan-based Fingerprinting Pipeline

Metascan-based techniques aim at honeypot detection using passive-fingerprinting techniques. Our framework uses information available through Shodan and Censys to determine if an instance is a honeypot. The metascan-pipeline consists of *four* stages based on the underlying protocol. Although some SotA, e.g. [15], have used mass-scan engines to search for honeypot signatures, we use persistent checks in our stages to assure that the instance is a honeypot. We use checks to determine if the network belongs to a research facility, has an identified domain attached to it, or if the instance is on a cloud infrastructure. This information helps to further distinguish the honeypots by analyzing operational parameters.

Shodan and Censys Search Contrary to the probe-based scanning that requires us to use a tool to perform the scan, we leverage the available data from Shodan and Censys that perform the scans daily. We search the platforms for systems with open ports concerning the services emulated by honeypots in our tests. The result of the search provides a list of instances that undergo further fingerprinting process. Both Shodan and Censys provide APIs for querying their databases. Algorithm 3 in the Appendix shows the procedure for the search performed on Shodan and Censys. The search results return an IP address and port, for the identified instances.

Keyword Search Shodan and Censys store information about the systems exposed to the Internet that include banners, web content, protocol negotiation parameters, and more. In addition to system-specific information, they also provide metadata about the IP address allocated to the system like geo-location, ISP/AS and the hosting provider. The degree of information and the format available on these databases vary based on the techniques followed by the mass-scan engine. We leverage such information to filter the instances obtained in the previous step. The search is performed with keywords identified from the probe-based stages like static content, banners, and protocol negotiations. Table A.13 shows the used keyword parameters for filtering instances in Shodan and Censys. The resulting data contains a list of instances of systems with specific ports and matching filtered criteria.

ISP and AS Check Honeypots are also classified based on their usage in research and production environments. Research organizations deploy honeypots to gather attack-data for threat intelligence research. Enterprise systems deploy honeypots for proactive attack detection. Following the previous stages, we examine whether the instance is part of a research organization or an institute. It is also possible that an enterprise company

may be hosting a production honeypot with an unassigned domain. For instance, the honeypots deployed in our lab lie under the university AS while they do not have a domain registered to them. To cope with this, this component checks the WHOIS database to search for information about the network to which the system is attached to.

Cloud Hosting Check Cloud infrastructure enables defenders to set up and deploy honeypots on cloud environments to easily gather attack-data. Many honeypot developers offer a container-based configuration of honeypots for easy installation and deployment. As a result, many honeypot instances can be found in cloud instances. We argue that many honeypots are deployed in cloud environments though they are logically invalid for the emulated infrastructure. For example, we find many instances of Conpot, an ICS based honeypot, which emulates industrial cyber-physical systems. However, it is improbable to find ICS devices on cloud networks. This component checks whether instances related to specific ICS protocols (i.e. Modbus and S7) are deployed on a cloud infrastructure.

Fully Qualified Domain Name (FQDN) check

A FQDN is allocated to an Internet-facing system to avoid memorization of the IP addresses. We perform a check to examine whether the identified instances from both pipelines have an assigned Domain Name Service (DNS) domain. Honeypot systems, by design, are fake systems and are unlikely to have domain names allocated as it is risky for the organizations deploying them. For instance, an attacker may claim to have found a vulnerable or compromised system belonging to an enterprise domain, resulting in negative publicity for an organization. Therefore, administrators, in principle, avoid assigning a domain/DNS for the honeypots. We utilize this understanding of the administrators and filter the IP addresses received from the IP pool to find systems without domain names assigned. The reverse DNS lookup is performed using the Domaintools that provides an extensive database for *whois() information [29]*. The IP addresses that do not have a DNS are transitioned to the next state. The FQDN check differs from the AS check, in a way that it checks for any domain associated with the IP address, while the AS check performs a lookup of the IP address allocation by the AS to an entity. The information about the AS and the ISP helps in identifying the type of entity, for example, a research organization or honeypot instance in a production network of an organization.

Framework Output

The output state of the framework provides a list of instances that are inferred as honeypots from our fingerprinting framework. The list contains instances from both the probe-based and the metascan-based honeypots.

3 Evaluation

We evaluate the ability of the proposed multistage honeypot fingerprinting framework in discovering honeypots. The evaluation considers *nine* honeypot implementations and specifically focuses on *nine* protocols as listed in Table A.3. The choice of honeypots is based on a number of factors. First, these honeypots are considered some of the most popular ones and most frequently deployed (see e.g. the ENISA recommendations in [30]). Moreover, these represent the honeypots examined in the majority of the related work (cf. Section 6), which provides us the ability to make some comparisons (e.g. with [14, 15, 31]). Lastly, all of the selected honeypots are open-source implementations.

Our main goal is to examine how many honeypots the framework can identify. We highlight here that the absence of ground truth data for honeypots is a known problem in the field. However, we argue that the multistage nature of the framework highly reduces the probability for false positives (we further discuss this issue in Section 3.5). In addition, we want to determine the relation between the probe-based and metascan-based detection. Our hypothesis is that the probe-based pipeline should produce significantly better results. Still, the question of whether the metascan pipeline can identify honeypots beyond the ones already identified via the probe-based methods is an open question that we will attempt to answer. Lastly, we are interested in further examining encounters with IP addresses that pass some, but not all, of our tests. We believe that these systems might be vulnerable ones, which can easily be exploited by adversaries.

3.1 Lab environment tests

First, we deploy all the honeypot implementations (see Table A.3) in a lab environment and test all probes which are implemented to collect state-specific information like banners, static content, protocol handshake and static command responses. We confirm that honeypots test positive for *all* the different modules (see Figure A.1) of the probe-based phase. Following these tests, we evaluate the multistage framework against the known honeypot instances in the lab environment. All the honeypot instances were successfully detected by our framework.

3.2 Evaluation Setup

After performing the aforesaid experiments, we are now ready to perform an Internet-wide scan. We use the ZMap tool as our scanning tool [24] to scan a total of 2.9 billion IP addresses*. Our tests follow the flow of Figure A.1. That is, we first perform a probe-based scan and afterwards perform an independent metascan by making use of Shodan and Censys [17, 22]. Our experiments were conducted in a period of six

---

*ZMap excludes a number of IP addresses from its scan by default; these include reserved and unallocated IP space.

Honeypots	Ports & Services	Version
Kippo	Ports:22/2222 Services: SSH	0.9
Cowrie	Ports: 22/2222 23/2323 Services: SSH, Telnet	2.1.0
Glastopf	Ports: 80, 8080 Services: HTTP	3.1.2
Dionaea	Ports: 80, 443, 21 Services: HTTP, FTP	0.9.0
Nepenthes	Ports: 21 Services: FTP	0.2.2
Amun	Ports: 23,21,80,36,143 Services: Telnet, FTP, HTTP, SMTP, IMAP	0.2.3
Conpot	Ports: 80, 502, 102 Services: HTTP, Modbus, S7	0.5.2
Gaspot	Ports: 100001 Services: ATG	base [32]
MTPot	Ports: 23 Services: Telnet	base [33]

Table A.3: Honeypots tested in our internal lab environment

months. The experiment is carried out as 3 scanning periods, for the entire-framework. The metascan-based approach was relatively faster to perform the search and analysis, although Shodan and Censys enforce rate limiting on the API requests. Over a period of six months, we conducted three iterations. The results depicted in the following sections provide a summation of all the unique honeypot instances identified from the three scan iterations.

We, once more, highlight that this article does not take into account high interaction honeypots. This is due to the very different characteristics of high interaction honeypots (i.e. real systems instead of emulated ones); in fact, this is the case with all the SotA (e.g. [14, 15, 31]). Hence, both our article as well as all existing related work are prone to false negatives.

3.3 Results

By firstly performing a ZMap scan, we derive Table A.4 that shows the number of identified systems (not necessarily honeypots) on the Internet that exhibit relevant open ports. Subsequently, the framework performs the various checks shown in Figure A.1.

Protocol	Port	No. of Systems on the Internet (in Million)
HTTP	80,8080, 8888	67.31
HTTPS	443	56.06
SSH	22	18.65
FTP	21	10.39
SMTP	25	7.71
Telnet	23	5.27

Table A.4: Number of identified instances and protocols/ports

Honeypot identification

Overall, the framework detected a total of 21,855 honeypots. Figure A.2 shows the honeypot instances detected over three sequential scans over a period of 6 months. Figure A.2 also depicts the change in honeypot instances detected over the 3 scans. The instances of honeypots Gaspot, Conpot and Amun (HTTP) were detected more in the third scan while the others remained constant or reduced. This could be because of honeypots instances undergoing either a churn or because they were simply blocked/offline. IP churn is the rate at which a networked host changes its IP address as a result of a changed configuration by the ISP or the network administrator of the organization. We discuss this further in Section 3.5. The metascan-based technique has identified $7,410$ unique honeypots and the remaining $14,246$ were detected by the probe-based technique. Figure A.3 summarizes the honeypots detected by probe-based and metascan-based approaches for each honeypot. The numbers on the bars indicate the *unique* instances detected by the approaches and scans. An interesting finding is that *all IP addresses identified as honeypots by the metascan-based approach were already detected by the probe-based approach.* This is important as it confirms our hypothesis that probe-based is superior to the metascan. In fact, this suggests that the metascan pipeline can be ignored without any loss of information.

Figure A.4, compares our findings with the SotA measurements from Vetterl et al. (*Bitter Harvest*) [14], Morishita et al. (*Detect Me*) [15] and Zamiri et al. (*Gas What?*) [31]. The figure shows the total honeypot instances detected by SotA in comparison to our approach. We note that the honeypots Nepenthes and Amun were not evaluated by [14]; in addition, [31] only evaluated Gaspot and Conpot honeypots. We want to highlight that the value of this figure does not lie within the improved results on the majority of the honeypots. Direct comparison with previous measurements is not adequate due to the different time frame. Instead, we argue that these results suggest a number of interesting findings. First, they independently confirm previous studies' conclusions with regard to the global (poor) state of honeypot deployments [14]. Sec-

Honeypots detected by scans
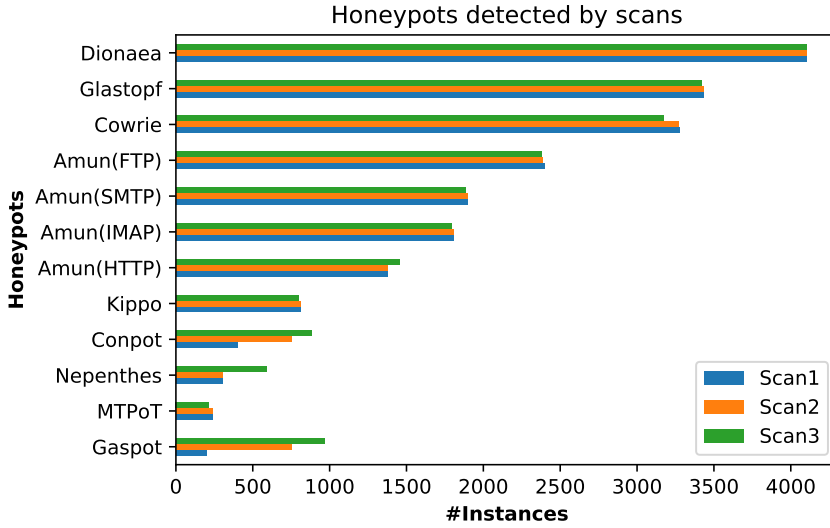
Fig. A.2: Honeypots detected per scan
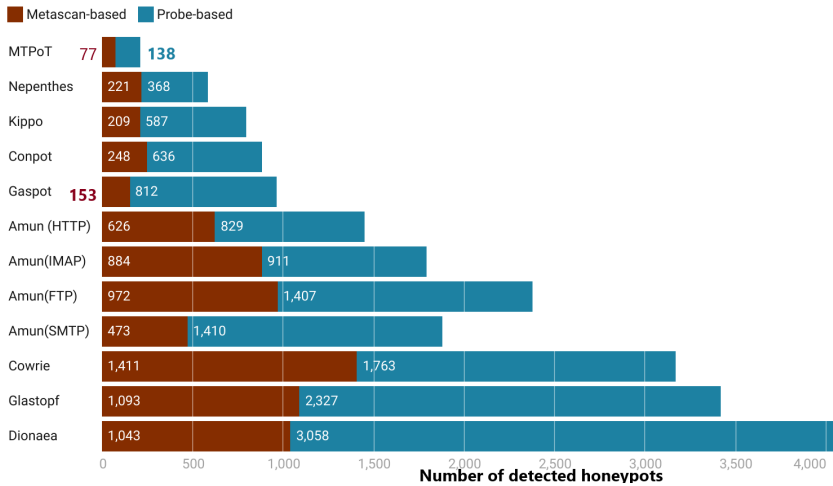
Fig. A.3: Honeypots detected by type and technique

ond, our results come more than one year after the aforesaid studies: this provided a relatively long period for honeypot administrators to react, while many honeypots (e.g. Conpot) have been updated to fix relevant vulnerabilities. Lastly, the multistage nature

of our framework suggests that, in contrast to related work, we should encounter a very small number of false positives. That is, IP addresses are only marked as honeypots when all (relevant) stages are confirmed.
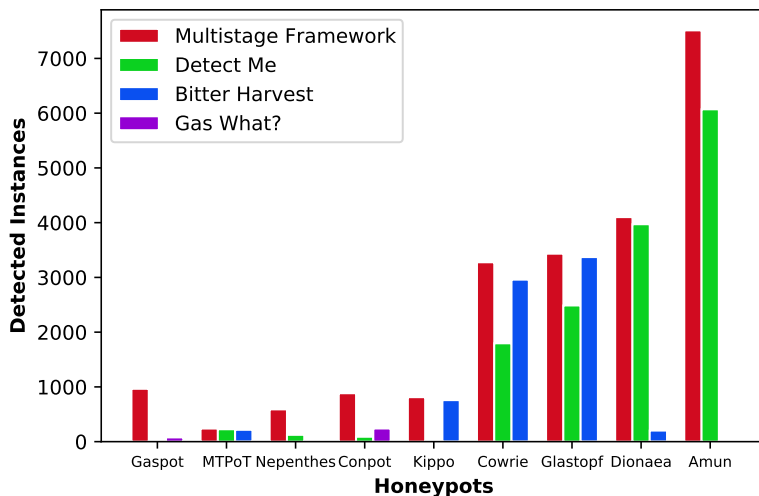
Fig. A.4: Comparison to previous measurements in related work

Honeypot Versions

We determine the versions of the instances detected as honeypots by examining specific changes added to the honeypots through patches released by the developers. However, versions could not be determined for some honeypots that do not maintain releases (i.e. MTPot and Gaspot). We find that the majority of the honeypots detected, have not been updated by the administrators even though there were patches released by the honeypot developers (e.g. for certain fingerprinting attacks). Furthermore, we detect instances running on honeypots that are no longer maintained by the developers. The developers of these honeypots disclose that the project has been discontinued and also suggest newer honeypots under active maintenance. We list the instances with the identified deployed versions in Table A.5.

Honeypots with Default Configuration

The honeypots considered in our tests can be deployed with a default configuration. Nevertheless, for some honeypots, the developers explicitly provide additional templates and

Honeypot	Deployed Version	#Instances
Conpot 0.5.2*	0.5.2	221
	0.5.0	496
	0.4.0	167
Cowrie 2.1.0*	2.1.0	17
	1.5.3	232
	1.5.1	2925
Glastopf 3.1.2*	3.1.2	4
	0.2.0	3416
Dionaea 0.8.0*	0.8.0	2259
	0.6.0	1782

Table A.5: Detected honeypot versions (* latest version)

guidelines to change the default settings. The usage of default honeypot configuration can be problematic as it makes fingerprinting significantly easier.

To determine this, we compare the cumulative results from the framework's *HTTP Static Response* and the *Static Command Response* stages to the default configuration of the deployed honeypots in our lab environment (see Section 3.2). Therefore, upon matching, we can infer that the instance is a honeypot deployed with its default configuration. We observe that the majority of the detected honeypots are running with default configurations that make primitive fingerprinting techniques like static http-content very successful. We list the number of honeypot instances running with default configurations in Table A.6.

Honeypots	#Instances with default configuration	#Instances without default configuration
Gaspot	925	40
MTPot	215	0
Conpot	777	107
Nepenthes	531	58
Kippo	773	23
Cowrie	3149	25
Amun	7455	57
Glastopf	3416	4
Dionaea	4064	37
Total	**21305**	**351**

Table A.6: Detected honeypots running on default configuration

Non-honeypot encounters

As a result of multistage checks from the framework, instances are filtered out at each
stage when they fail the matching criteria. We further analyze the non-honeypot in-
stances that were filtered out at multiple stages to determine the cause of filtration at a
particular stage and/or the success in other stages. Table A.12 in the Appendix shows
the non-honeypot instances determined at stages in our framework based on honeypot
types. Furthermore, we find a total of 355,054 vulnerable systems (see Table A.7) with
unpatched versions and default passwords among the non-honeypot systems identified.
Based on this, we derive the following findings.

Vulnerability	# Instances
Default passwords (SSH)	
root, root	216
admin, admin	124
root, 1234	23
admin, 1234	43
root, 123456	21
root, (no password)	18
admin, (no password)	28
Default passwords (FTP)	
root, root	94
admin, admin	29
root, 1234	19
admin, 1234	8
Vulnerable Banners (SSH)	
SSH-2.0-ROSSSH	263,516
SSH-2-0-libssh-0.7.0(5)	196
Vulnerable Banners (FTP)	
220 ProFTPD 1.3.5 Server	53,873
220 ProFTPD 1.3.1 Server	15,823
220 Serv-U FTP Server v6.2	21,023
Total	**355,054**

Table A.7: Vulnerable instances of identified non-honeypot instances

SSH and FTP Instances with Default Passwords   We find SSH instances running
on default passwords that met the initial criteria for SSH honeypot detection in our
framework, but fail in other stages (e.g. static command and library checks). These
instances' credentials match the ones of the default passwords accepted by Kippo and
Cowrie honeypots. Our conclusion is that these are either vulnerable devices with

default logins or high-interaction honeypots. We list the number of vulnerable SSH instances found with default passwords in Table A.7.

SSH and FTP Instances with Vulnerable Versions From the instances that were filtered out of the banner check stage (in the probe-based pipeline), we identify the number of instances that appear to contain vulnerable versions in their banners. In particular, we take into account banners that have a high severity vulnerability (by making use of the National Vulnerability Database [34]). We identify a total of 263, 712 instances with vulnerable versions as per the advertised banners. The banners and the number of instances identified are listed in Table A.7.

Experiment repetition: gain and blocked/offline instances

Due to the nature of our experiments (i.e. long time windows and rather aggressive fingerprinting scans) we expect that: *i)* we will observe some fluctuation in our results, *ii)* we will have some gain as new honeypots are introduced on the Internet, *iii)* we expect some of the networks to blocklist our scanners, and lastly *iv)* we anticipate some honeypots not to be responsive due to them taken down, maintenance and/or network errors.

We scan the Internet with a different scanning host that has different IP address and subnet. We compare the results from the different scanning periods to identify new and existing honeypot instances. In the next step, we analyze the IP address of the new honeypot instances detected against our framework and check the IP address for their subnet and their AS. If the IP address belongs to a different subnet but belongs to the same AS, and further matches to the properties of the honeypot identified in the previous period, we infer that the honeypots are the same but had some churn-related effects. Moreover, we further examine the gain vs. blocked trade-off by trying to connect to the new IP address of the honeypot instance from our previous scanning host. If the honeypot instance blocks the connection from the first connected host but was connected by the second scanning host, then it is very likely that the honeypot administrator has blocklisted the IP address of the first scanning host.

Table A.8 shows the number of new honeypot instances detected in the scans and the instances that were either blocked or offline. There was a significant number of new Nepenthes honeypots instances detected in the third scan. On tracing the IP addresses of the new instances, we find that all the new detected honeypots were hosted by a hosting provider which was traced earlier hosting Nepenthes instances on another subnet. We can infer that either the honeypots were configured to undergo some IP rotation logic or were simply offline for a certain period. Overall, we find that only 2.3% of the honeypot instances have changed their IP and only 1% are not offline after the first scan.

Honeypot	Scan-2 New Instances	Scan-2 Blocked/Offline	Scan-3 New Instances	Scan-3 Blocked/Offline
Gaspot	567	12	387	11
MTPoT	0	1	0	23
Nepenthes	0	3	573	16
Conpot	367	33	110	23
Kippo	0	4	0	13
Amun	0	3	63	51
Cowrie	0	4	0	98
Glastopf	3	2	0	13
Dionaea	0	0	0	0

Table A.8: Identification gain vs. blocked/offline instances

3.4 Shodan Honeyscore

The Shodan Honeyscore is a proprietary algorithm used to determine whether a crawled instance is a honeypot or not [17]. Shodan offers an API that provides a score for IPs detected as probable honeypots. The score ranges from $[0, 0.3, 0.5, 0.8, 1]$, with 0 denoting that the IP is not a honeypot and 1 that it is. The API also returns the value *NA* when no information is available for a specific IP address. Since the Honeyscore is not open source, not many conclusions can be derived by examining its output. In fact, it is not disclosed which honeypots can be identified by Shodan's Honeyscore. Nevertheless, we expect that there is some overlap with regard to the fingerprinting techniques used by our framework and Shodan's Honeyscore.

We fetch the Honeyscore for *all* the honeypot IP's determined by our framework and compare the results with Shodan. Figure A.5 depicts the Honeyscore assigned to honeypot instances detected through our framework (for the combination of both metascan and probe-based results). We observe that Shodan returns 0 as Honeyscore for many of the IPs. This suggests that the Honeyscore is not taking into account as many checks as our framework. Moreover, the high deviations observed with regard to Glastopf and Amun suggest that Shodan is not very effective in identifying such honeypots.

3.5 Validation

The absence of ground truth knowledge regarding honeypots creates a challenging landscape for measuring metrics such as precision or possible false positives. This is a fundamental problem in the area of honeypot fingerprinting that cannot be solved in its entirety. Hence, in the following we attempt to provide indications on why false positives are not a significant issue in our approach.
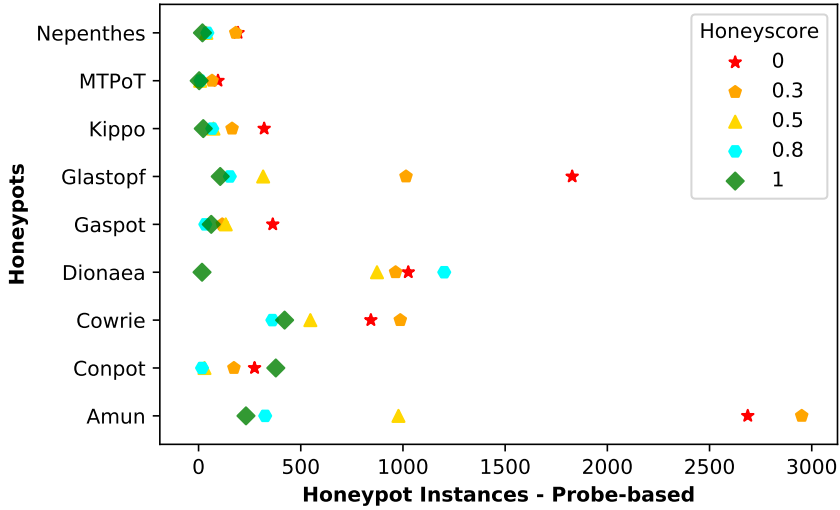
Fig. A.5: Comparison with Shodan's Honeyscore

First, in contrast to the state of the art, we propose a framework that requires multiple steps to be confirmed until an IP address is marked as a honeypot. These steps include a multitude of independent checks which, we argue, significantly decrease the probability of false positives. Looking at the SotA, Vetterl et al. [14] measure the detection accuracy using the responses received from the honeypots by generating a cosine similarity score and Morishita et al. [15] use the matching of honeypot signatures in four datasets. In contrast, our approach relies on multiple checks at each stage to minimize false positives.

Second, we replicate and extend the ground truth validation proposed by [15] and [35]. Morishita et al. argue that a honeypot IP address cannot be present in IP spaces that are known for their commercial usage. This argument obviously does not solve the absence of ground truth, but rather provides a minor indication that the identified IP addresses are not clear false positives. Vogt et al. also use a similar validation in their evaluation to check if the domain identified by their model is listed on sources providing web-statistics like the top 1 million domains [35]. In this context, we evaluate our results by comparing the identified honeypot IP addresses with the top 1 million domains from Alexa [36], Majestic [37] and Cisco-Umbrella [38] with known benign FTP servers, as well as known university SMTP domains. For this evaluation we fetch the Alexa top 1 million domains from Alexa, perform a DNS lookup and examine whether our results match them. Similarly we fetch the top 1 million domains from the Majestic and the Cisco-Umbrella websites. We confirm that none of IP address from these domains are

found in our results. We note that the IP addresses for some of the domains change based on the geo-location of resolution due to the Content Delivery Network (CDN). Hence, we repeated the experiments by connecting to many different geo-locations by using a Virtual Private Network (VPN) provider. Moreover, we fetch the list of official FTP mirrors from GNU [39], Apache [40], Ubuntu [41], Debian [42] and find $1,231$ unique domain names. Upon performing a DNS lookup, we get $2,784$ IP addresses. Once more, none of the identified honeypots match these IP addresses. Furthermore, we retrieve the list of university domain names from [43] for evaluating Amun (SMTP) honeypot. Upon performing a DNS lookup we find $12,012$ IP addresses. There were no honeypots detected in the domains from this list. To sum up, while the SotA uses a singular method to deal with false positives, our approach utilizes multiple stages. Moreover, we further test our results with an adaption and extension of the techniques employed by [15] to address the absence of ground truth knowledge.

4 Discussion

The evaluation of the multistage framework involved an experimental setup to reduce false positives and help in classification of honeypot instances. In this section we discuss the ethical considerations and experimental setup considered during the experimentation phase.

4.1 Ethical considerations

This section takes into account the various ethical considerations we had during our research.

Experiments

First, we inform the IT administrators of our organization about the ongoing research and seek their assistance for providing an approved setup for scanning the Internet. This is important as organizations tend to blocklist the IP addresses of sources that appear to be scanning them. Second, we setup a website on the IP address of our scanner that provides a disclosure/explanation of our research purpose. This assists in limiting the effects of blocklisting the IP addresses of our organization.

Results Disclosure

The list of honeypot instances obtained through our framework is not publicly shared. We only present here aggregated statistics and do not share any identifiers of the honeypot instances. We seek guidance from the privacy department of our organization for guidelines on storing the results of our experiments and being compliant to GDPR. We

followed the GDPR compliance by anonymizing the IP addresses after three months following the completion of our research.

Ethical disclosure: notifying Honeypot Developers

We contact the honeypot developers of all of the active honeypot implementations and provide them with the specifics of the honeypot fingerprinting methods that can be used against them. Moreover, we contacted members of the Honeynet project [44], an international security research organization that focuses on honeypot research, to further disclose the fingerprinting mechanisms that we have identified.

Ethical disclosure: notifying Honeypot Administrators

We take all $21,855$ IP addresses that were identified as honeypots and perform a WHOIS scan to find relevant contact information. Based on this, we identify 939 email addresses that correspond to all the IP addresses that we managed to find information about. We note that in many cases one email address corresponds to hundreds of honeypot instances, deployed in the same network. There are multiple benefits from this procedure. First and foremost, we notify honeypot administrators that their deployments are vulnerable to our fingerprinting methods. Second, we ask administrators to contact us in case they are confident that our finding is a false positive and no honeypot deployment has taken place in their networks. This acts as an additional false positive sanity check. Until the time of submission, we did not receive any false positive claim from the contacted administrators.

4.2   Limitations

As discussed in Section 3.5, the research field of low and medium interaction honeypot fingerprinting has the fundamental limitation that there is no global ground truth knowledge with regard to honeypot deployment. This translates to potential false positives. Our work is also influenced by this: while our findings come as the result of multiple stages and checks there may still be cases in which an instance is incorrectly labeled as a honeypot.

The proposed multistage framework leverages multiple checks to determine if the end instance is a honeypot. As part of the failed checks from the framework, 355,000 non-honeypot instances have been detected. While we argue that the majority of these are most likely vulnerable/misconfigured devices, it might be that some are high-interaction honeypots. Ideally, one could perform manual tests on a sample of these systems by logging into them and attempting to understand the presence of a honeypot environment. However, this would be illegal and therefore we could not perform such an action. Moreover, fingerprinting high-interaction honeypots requires extensive probing and analysis. Hence, this is considered to be out of the scope of this article.

Lastly, while direct comparisons to the state-of-the-art is considered the default evaluation methodology in many fields of cybersecurity, this is not possible in our setting. The combination of the aforementioned ground truth knowledge problem, along with the different time frames of the experiments make direct comparisons unreliable. We argue, that our work and results is not competing to the state-of-the-art. This is amplified by the fact that we are dealing with IP addresses and therefore topics such as static vs dynamic IP addresses, Network Address Translation (NAT), and churn need to be taken into account.

5 Countermeasures against fingerprinting

This section discusses potential countermeasures against fingerprinting. First, we want to emphasize that, due to their nature, low and medium interaction honeypots can always be identified upon continuous interaction and response analysis. Instead, we argue that the emphasis should be to reduce as much as possible fingerprinting vulnerabilities that can easily be automated.

5.1 Metascan countermeasures

Metascan-based methods rely on data that is obtained without interaction from the target system. This can be translated to a scenario in which malware uses Shodan's API to ask whether an IP address is a honeypot before contacting it (e.g. for propagation reasons). We argue that Shodan, Censys and other scanners must introduce limitations to their honeypot identification services. From the honeypot deployment and implementation perspective, Moving Target Defense (MTD) techniques could be employed by honeypot implementations to avoid a static IP identification. We also discourage the usage of cloud hosting providers for honeypots based on ICS protocols. Honeypots like RIoTPot [11] maintain an active list of IP addresses from known scanning-services and label all the traffic from these sources. This list can be further used to block all traffic from scanning-services and hence limit fingerprinting attempts.

5.2 Probe-based countermeasures

For *probe-based* methods, we suggest that the honeypots are made self-aware and dynamic each time an attack has been detected. Fingerprinting methods can be less effective if the honeypots contain non-static parameters while also choosing selective services periodically. In addition, honeypots rely heavily on protocol emulation libraries. It is important to refer to libraries that are regularly maintained. Furthermore, we suggest making additional tweaks to the references to modify default static responses by comparing the responses to an actual system. Default configurations must be avoided and dynamic configuration based on the attack and the environment is recommended.

Dynamic responses

Honeypot fingerprinting techniques exploit the limited exploitation capabilities of low-interaction honeypots for indicators of deception. The limited simulation entails reduced support and hard-coded responses for commands. Automated fingerprinting checks can be deceived by introducing dynamic response patterns and a degree of randomness. For example, the *date* command could respond with current date and time, or return changing time on sequential requests. Fingerprinting techniques could either check the response for the *date* command for static values or sophisticated techniques can compare the response with the timestamp received in the packet. We acknowledge that it is beyond the scope for low-interaction honeypots for enabling dynamic responses. However, we suggest to implement dynamic response for common commands used by bots and malware.

Maintenance and library support

Low-interaction honeypots use libraries for simulation of services. For example, Cowrie uses the Twisted library for implementing the SSH protocol simulation. Most of the libraries used in honeypot implementations are however not maintained. This entails that the honeypot implementations are vulnerable to any bugs affecting the libraries. Honeypot implementations must be periodically revised and maintained to prevent staleness. Fingerprinting research by Vetterl et al. [14] suggest that limited protocol emulation in honeypots that use poorly maintained libraries can be fingerprinted by examining the responses and calculating the effective deviation. The authors suggest short and long-term countermeasures from identification of fingerprinting probes to the development of new-generation honeypots that are similar in actual protocols.

High interaction components

High-interaction honeypots are actual or real systems that run a vulnerable service and log all the traffic. With high-interaction honeypots, the deceptive layer is the actual vulnerable service with the underlying system and hence provides the attacker with full interaction capabilities. High-interaction honeypots addresses some limitations of low-interaction honeypots like limited simulation and low resources. However, as high-interaction honeypots run on actual systems, there is a risk of them getting exploited to perform attacks on systems on the Internet. Such risks can be addressed by configuring network rules and using containerized, ephemeral instances.

6   Related Work

This section focuses on honeypot-specific fingerprinting research. We note here that besides honeypots, there has been research in the identification of intrusion detection

systems and network telescope sensors (e.g. [45, 46]). However, we consider this out of the scope of this article. Similarly, we will not discuss here fingerprinting of honeypot-like systems (such as honeytoken identification) [47]. We also note all papers in the SotA exclude high interaction honeypots from their analysis.

Authors & Year	Fingerprinting technique	IPv4 scan
Holz et al., 2005 [48]	Static command response check	No
Wang et al., 2010 [49]	Static command response check	No
Hayatle et al., 2012 [50]	Static command response check	No
Aguirre et al., 2014 [51]	Library dependency check, static command response check	No
Vetterl et al., 2018 [14]	Banner check, protocol handshake check, Library dependency check, static command response check	Yes
Vetterl et al., 2019 [52]	Banner check, Library dependency check	Yes
Huang et al., 2019 [53]	Banner check, static command response check	No
Morishita et al., 2019 [15]	Banner check, http static response	Yes
Zamiri et al., 2019 [31]	default config, static response, protocol handshake	Yes
Papazis et al., 2019 [28]	Banner check, http static response, static command check	No
Sun et al. , 2021 [54]	fuzzing, limited response	No

Table A.9: Overview of the related work

Techniques for fingerprinting honeypots were first proposed early in 2005 by Holz et al. [48]. The authors state that limited simulation and virtualization cause restricted

interaction on the honeypot system that leads to fingerprinting possibilities. Holz et al. propose fingerprinting techniques to detect User-mode Linux (UML) kernels by observing the process id's, virtualized environments by analyzing the ping response time, and debuggers by using *ptrace()*. The presented techniques are focused more on fingerprinting at the process and operating system level. This is mainly due to the limited availability of honeypots at the time of research.

Wang et al. present an approach to detect honeypots in advanced botnet attacks [49]. Their work is based on the assumption that security professionals deploying honeypots have a liability constraint; they cannot allow their honeypots to participate in attacks. Hence, botmasters can detect honeypots by checking whether compromised machines in their botnet can successfully send out unmodified malicious traffic. This approach is based on monitoring the traffic which is transmitted by the infected system through the bots. For example, the use of *iptables* command on Linux environments to list the port forwarding helps in the identification of honeypots because of outbound traffic rules. This information is transmitted by the bot to the botmaster. The authors also present fingerprinting techniques involving ping response time.

Vetterl et al. propose the detection of nine well known open source honeypots by constructing probes to fetch specific data and observe the deviation between the response from actual honeypots [14]. The deviation is measured as a cosine coefficient. This approach provides a good insight into the state of open source honeypots and their vulnerability to fingerprinting attacks. The methodology is evaluated and the authors identify $7,605$ honeypots on the Internet. In comparison, although our framework employs an approach to observe deviation in responses, we further extend the framework to include additional checks to reduce false positives.

Moreover, Morishita et al. [15] propose honeypot fingerprinting through signature-based detection. The authors develop signatures for 15 open source honeypots offering multiple services. The signatures are then matched against responses obtained through probes and mass-scan engines to determine if the system is a honeypot. The approach is evaluated and the authors detect $19,208$ honeypots. Our approach checks for known honeypot banners returned by the instances, although it does not rely solely on the banner check to flag the instance as a honeypot.

In addition, Zamiri et al. detect GasPot [32], an ATG-based ICS honeypot through probes designed to fetch information about the default configuration and limited emulation of the protocols [31]. The authors study ICS honeypots (specifically of Conpot and GasPot) list features, e.g. limited emulation static responses, and identify the underlying OS to eventually fingerprint them. They perform an Internet-wide scan to detect 17 GasPot and 240 Conpot instances.

Huang et al. probe remote systems and label the response data to train a machine-learning model to classify systems as honeypots [53]. The method follows a recursive probing process to obtain featured data for classification. The features include application-layer, network-layer, and system-layer properties. The authors train the

model for classification by providing data from known honeypot systems. However, the authors do not classify the responses from widely recognized honeypots like Kippo, Cowrie, or Dionaea.

Papazis et al. attempt to exploit some of the virtual network layers implemented in honeypots, using tools like NMap, to fingerprint them [28]. In addition, they demonstrate the identification of network and service anomalies like link latency and limited emulation that may also lead to honeypot detection. The authors discuss detection vectors for honeypots like Sebek, Artillery, BearTrap, KFSensor, HoneyD, Kippo and Dionaea.

Lastly, Sun et al. propose fuzzing-based technique for fingerprinting honeypots in industrial cyber-physical systems [54]. The technique is inspired by vulnerability mining and utilizes error handling to distinguish honeypots and real devices. The technique follows a two-step approach. In the first step mutation rules and security rules are setup to generate effective and secure probe packets. Then, these probe packets are used for scanning and identification in the second step. The authors test the method with a dataset and do not scan the Internet with the created probes.

Table A.9 summarizes the fingerprinting related work. We note that majority of the related work does not evaluate their proposed techniques by performing an active search for honeypots on the Internet. This is mainly due to the fact that Internet-wide scanning was not trivial until the emergence of ZMap [24]. That said, the fingerprinting techniques suggested by [14, 15, 31, 52] include a thorough evaluation. However, their core limitation is that they focus on a limited number of techniques for fingerprinting. In this article, we propose a multistage framework that combines probe-based techniques (targeting multiple system layers) with data available from Internet mass-scan search providers to systematically detect honeypots.

7 Conclusion

Honeypots are unique mechanisms for understanding attack methodologies, discovering new attack trends, as well as for early warning systems. In this article, we proposed a framework for honeypot fingerprinting that includes new and SotA components and is able to identify thousands of honeypot instances for *nine* of the most popular honeypot implementations. Our work reduces false positives by the utilization of multiple checks before determining that an instance is a honeypot. Our results also suggest that probe-based fingerprinting techniques are significantly more effective in detecting honeypots than the metascan techniques that utilize third-party systems like Shodan. Although metascan techniques are less invasive, using them exclusively could result in higher false-positives. We once more highlight that our work is in the direction of improving honeypots rather than arguing against them. With the availability of open honeypot identification APIs , such as Shodan's Honeyscore, it is only a matter of time that we see honeypot-evading malware. In this context, we contacted both the developers and

the administrators of the honeypots to make them aware of potential fingerprinting issues. However, based on the experience of previous work we are not over-optimistic with regard to the patching/updating of such systems. Instead, we argue that novel components must be added in new/old honeypots that are in the direction of Moving Target Defenses schemes. We plan to further investigate fingerprinting countermeasures in our future work.

Appendix

A   Multistage Framework for Honeypot Fingerprinting

Table A.10 shows the banners advertised by honeypots in our evaluation. Most honeypot implementations offer limited banners or custom banners.

Honeypot	Protocol	Banner
Kippo	SSH	Default: SSH-2.0-OpenSSH_5.1p1 Debian-5 # SSH-1.99-OpenSSH_4.3 # SSH-1.99-OpenSSH_4.7 # SSH-1.99-Sun_SSH_1.1 # SSH-2.0-OpenSSH_4.2p1 Debian-7ubuntu3.1 # SSH-2.0-OpenSSH_4.3 # SSH-2.0-OpenSSH_4.6 # SSH-2.0-OpenSSH_5.1p1 Debian-5 # SSH-2.0-OpenSSH_5.1p1 FreeBSD-20080901 # SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu5 # SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu6 # SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu7 # SSH-2.0-OpenSSH_5.5p1 Debian-6 # SSH-2.0-OpenSSH_5.5p1 Debian-6+squeeze1 # SSH-2.0-OpenSSH_5.5p1 Debian-6+squeeze2 # SSH-2.0-OpenSSH_5.8p2_hpn13v11 FreeBSD-20110503 # SSH-2.0-OpenSSH_5.9p1 Debian-5ubuntu1 # SSH-2.0-OpenSSH_5.9
Cowrie	SSH	Debian GNU/Linux 7
Cowrie	Telnet	\xff\xfd\x1flogin:
Glastopf	HTTP	Apache httpd
Dionaea	FTP	220 Welcome to the ftp service
Amun(SMTP)	SMTP	220 mail\.example\.com SMTP Mailserver
Amun(IMAP)	IMAP	a001 OK LOGIN completed
Amun(FTP)	FTP	220 Welcome to my FTP Server
Conpot	SSH	SSH-2.0-OpenSSH_6.7p1 Ubuntu-5ubuntu1.3
Conpot	Telnet	Connected to [00:13:EA:00:00:0]
Gaspot	ATG	Linux 3.X|4.X
Nepenthes	FTP	220 —freeFTPd 1\.0—warFTPd 1\.65—
MTPot	Telnet	\xff\xfb\x01\xff\xfb\x03\xff\xfc'\xff\xfe\x01 \xff\xfd\x03\xff\xfe\"\xff\xfd'\xff \xfd\x18\xff\xfe\x1f

Table A.10: Banners advertised by Honeypots (adapted from [14] and [15] see *Banner check* in Section 2.2)

Table A.11 shows the static content received as HTTP response from honeypots for specific requests. The static responses are either due to limited emulation or due to honeypots being deployed with default configuration.

Honeypot	HTTP Request	HTTP Response Contents
Glastopf	GET / HTTP/1.0	1. \<h2\>My Resource\</h2\> 2. \<h2\>Blog Comments\</h2\>\n \<label for=\"comment\"\>Please post your comments for the blog\</label\>\n \<br /\>\n \<textarea name=\"comment\" id=\"comment\" rows=\"4 \" columns=\"300\"\>\</textarea\>\n \<br /\>\n \<input type=\"submit \" name=\"submit\" id=\"submit_comment\" value=\"Submit\" /\>\n
Amun	GET / HTTP/1.0	\<!DOCTYPE HTML PUBLIC \"-//IETF//DTD HTML 2\.0//EN\"\>\<html\>\<head\>\<title\> It works!\</title\>\</head\>\<html\>\<body\>\<h1\>It works!\</h1\>\<br\>tim\.bohn@gmx\.net \<br\>johan83@freenet\.de\</body\>\</html\>\n\n
Dionaea	GET / HTTP/1.0	\<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML 3\.2 Final//EN\"\>\<html\>\n\<title\> Directory listing for /\</title\>\n\<body\>\n\<h2\>Directory listing for /\</h2\>\n
Conpot	GET /HTTP/1.0/ index.html	1. Last-Modified: Tue, 19 May 1993 09:00:00 GMT 2. Technodrome 3. Mouser Factory

Table A.11: HTTP Response from Honeypots (see *HTTP Static Response* in Section 2.2)

Table A.12 shows the non-honeypot instances determined at stages in our framework based on honeypot types. Limited emulation in honeypots cause identification at different levels that are determined by the stages in our framework.

Honeypot	Portscan	Failed Banner	Failed Static http response	Failed SSL/TLS Certificate check	Failed Protocol Handshake	Failed Library Dependency Check	Not a Honeypot
Kippo	4361857	4324502	NA	NA	34887	1656	4361045
Cowrie	4361857	4318645	NA	NA	37836	2100	4358581
Glastopf	57062712	56385819	673462	NA	0	0	57059281
Dionaea	43944853	43890588	49963	201	0	0	43940752
Nepenthes	10391953	10391645	NA	NA	3	0	10391648
Conpot	29950	28693	NA	NA	732	333	29758
Gaspot	222593	222393	NA	NA	0	0	222393
MTPot	2923651	2923412	NA	NA	0	0	2923412
Amun(SMTP)	6020828	6018931	NA	NA	0	0	6018931
Amun(IMAP)	4152084	4150278	NA	NA	0	0	4150278
Amun(FTP)	10391953	10389555	NA	NA	0	0	10389555
Amun(HTTP)	43944853	43942485	NA	NA	0	0	43943476
Total	**187809144**	**186986946**	**724416**	**201**	**73458**	**4089**	**187789110**

Table A.12: Non-honeypot encounters by stage. See also Section 3.3.

Table A.13 denotes the keywords used in Shodan and Censys to retrieve honeypots. The keywords are derived from banners and static content advertised by honeypots.

Honeypot	Shodan	Censys
Glastopf	<h2>My Resource</h2>	80.http.get.body: "<h2>My Resource</h2>/"
Dionaea	ssl:"Nepenthes"	443.https.tls.certificate.parsed.subject.common_name: "Nepenthes Development Team"
Conpot	port:"102" product:"Conpot"	80.http.get.body: "Technodrome"
Nepenthes	product:"Nepenthes HoneyTrap fake vulnerable ftpd"	21.ftp.banner.banner: "220 —freeFTPd 1\.0—warFTPd "
Amun	"220 Welcome to my FTP Server"	"80.http.get.body: tim.bohn@gmx.net" 21.ftp.banner.banner: "220 Welcome to my FTP Server" 25.smtp.starttls.banner: "220 mail\.example\.com SMTP Mailserver" 143.imap.starttls.banner: "OK LOGIN completed"
Gaspot	I20100 port: "10001"	"I20100"

Table A.13: Honeypot keywords search (see also paragraph *Keyword Search* in Section 2.2)

Table A.14 shows the static response returned by honeypots for specific commands requested by our probes. Limited emulation or default configuration lead to static response from the honeypots.

Honeypot	Command	Response
Conpot	S7_ID station name unit name	88111222 "STATOIL STATION" "Technodrome"
Kippo	nano vi	E558: Terminal entry not found in terminfo
Cowrie	arp	IP address HW type Flags HW address Mask Device 192.168.1.27 0x1 0x2 52:5e:0a:40:43:c8 * eth0 192.168.1.1 0x1 0x2 00:00:5f:00:0b:12 * eth0
Amun(FTP)	quit	221 Quit. 221 Goodbye!
Gaspot	I30100	9999FF1B

Table A.14: Overview of honeypot static responses. In reference to Section 2.2

Table A.15 provides an overview of the number of honeypot types and instances detected over the three scanning periods.

Honeypot	Scan 1	Scan 2	Scan 3	Total	Total (active)
Dionaea	4101	4101	4101	**4101**	4101
Glastopf	3431	3433	3420	**3433**	3420
Cowrie	3276	3272	3174	**3276**	3174
Amun(FTP)	2398	2388	2379	**2398**	2379
Amun(SMTP)	1897	1897	1883	**1897**	1883
Amun(IMAP)	1806	1806	1795	**1806**	1795
Amun(HTTP)	1377	1375	1455	**1455**	1455
Kippo	812	809	796	**812**	796
Conpot	399	751	884	**884**	884
Nepenthes	305	302	589	**589**	589
MTPoT	239	238	215	**239**	215
Gaspot	200	755	965	**965**	965

Table A.15: Honeypots detected per scan

B   Framework specific checks and pipeline

Algorithm 1, represents the pseudo-code block that checks an instance for Dionaea's default certificate parameters. In lines 3-5, the algorithm retrieves the certificate from the web server by accepting the IP and port of the instance and checks for common attributes subject organization, country, and issuer. These attributes have static values assigned by the honeypot developers. In steps 7-9, the algorithm checks if the values match the Dionaea honeypot certificate's static values. Upon match, the algorithm returns that the instance is a Dionaea honeypot.

Algorithm 2 shows the checks done in the Metascan-based pipeline. The algorithm checks or open ports, and performs keyword based check to list instances that match the static content delivered by honeypots. Furthermore, additional checks like FQDN and cloud hosting checks are performed for determining specific honeypot types.

Algorithm 3 represents the metascan search performed to determine instances with specific ports exposed to the Internet. The algorithm performs a search on Shodan and Censys mass scan engines for specific ports that are open on honeypots in our test.

Lastly, algorithm 4 represents the Protocol Handshake check procedure described in Section 2.2. The algorithm checks for a deviated response from the instances for specific negotiation parameters and response based on the port and the service of the instance.

Algorithm 1: Certificate Check

---

    input : ip, port
    output: isDionaea   `/* True if certificate from Dionaea */`
    begin
1 | *checkCert*(ip,port)
2 | isDionaea = false
3 | cert = $ssl.get\_server\_cert(ip, port)$
4 | X509 = `Crypto.X509.load_cert(cert)`
5 | org = `X509.subject.org`
6 | **if** *cert* **then**
7 | | **if** *org= "dionaea.carnivore.it"* **then**
8 | | | isDionaea = true
9 | | Return isDionaea
10 | **end**

Algorithm 2: Metascan-based Pipeline

---

    input : ports   `/* ports */`
    output: findHoneypot   `/* Honeypots on the Internet */`
    begin
1 | ip ← *metasearch*(ports)
                                           `/* Shodan and Censys Search */`
2 | **foreach** *ip* **do**
3 | | kw = *keywordSearch*(ip)
4 | | **if** *kw* **then**
5 | | | **foreach** *ip* **do**
6 | | | **if** *checkfqdn(ip)* **then**
7 | | | | return hasFqdn = $false$
8 | | | **if** *!hasfqdn & port=502|102* **then**
9 | | | | **if** *cloudCheck(ip)* **then**
10 | | | | | return isHoneypot = $true$
11 | | | | isHoneypot
12 | | | **if** *!hasfqdn* **then**
13 | | | | **if** *isResearch(ip)* **then**
14 | | | | | return isResearch = $true$
15 | | | | isHoneypot
16 | | | endIf
17 | | endFor
18 | **end**

Algorithm 3: Metascan Search

---

    input : port   `/* search parameter */`
    output: instanceIP   `/* Instances with open ports */`
    begin

1	instances`[]` = *null*
2	*shodanSearch*(port)
3	**foreach** *ip* **do**
4	instances`[]`.append(ip, port)
5	return instances`[]`
6	endFor
7	*censysSearch*(port)
8	**foreach** *ip* **do**
9	instances`[]`.append(ip, port)
10	return instances`[]`
11	endFor
12	end

Algorithm 4: Protocol Handshake Check

---

input : instance[],   /* Instance [ip, port, protocol, isDeviated] */
output: instance[isDeviated]   /* Handshake is deviated */
begin

1		*CheckHandshake*(instance[])
2		**foreach** *ip in Instance[]* **do** /* For each instance */
3		**if** *port=22/2222 & protocol="SSH"* **then**
4		*request*("SSH-2.0-OpenSSH\n\n\n\n\n\n\n\n\n")
5		if **response** = "bad Packet length" or "protocol mismatch"
6		return Instance[isDeviated] = *true*
7		**else** *request*("SSH-2.0-OpenSSH_6.0p1 Debian-4+deb7u2\n")
8		if **response** = "protocol mismatch\n"
9		return Instance[isDeviated] = *true*
10		**elseif** *port=102 & protocol="S7"* **then**
11		*request*(H,0300002102f08032070000...)
12		if **response** = "0x32"
13		return Instance[isDeviated] = *true*
14		**elseif** *port=502 & protocol="Modbus"* **then**
15		*request*(function_code:None, slave_id:0, request:0000000000050..)
16		if **session.state** = "Disconnected"
17		return Instance[isDeviated] = *true*
18		**elseif** *port=25 & protocol="SMTP"* **then**
19		*request*(PASS:Test)
20		if **response** = "220 OK"
21		return Instance[isDeviated] = *true*
22		**elseif** *port=143 & protocol="IMAP"* **then**
23		*request*(RCPT TO:TEST)
24		if **response** = "221 Bye Bye"
25		return Instance[isDeviated] = *true*
26		**elseif** *port=21 & protocol="FTP"* **then**
27		*request*(ftp (ip))
28		if **packet.windowSize**=4096 & **session.disconnect.timeout**=45
29		return Instance[isDeviated] = *true*
30		**elseif** *port=23/2323 & protocol="Telnet"* **then**
31		*request*(telnet (ip))
32		if **packet.response**= "You have connected to the telnet server"
33		return Instance[isDeviated] = *true*
34		**elseif** *port=80/8080/443/8443 & protocol="HTTP/HTTPS"* **then**
35		*request*(GET /HTTP/1.0 (ip))
36		if **response.packet.header.server**= "nginx" or "Apache/1.3.29" or "BaseHTTP/0.3 Python/2.5.1" or "Microsoft-IIS/5.0"
37		return Instance[isDeviated] = *true*

References

[1] L. Spitzner, "The honeynet project: trapping the hackers," *IEEE Security Privacy*, vol. 1, no. 2, pp. 15–23, 2003.

[2] M. Nawrocki, M. Wählisch, T. C. Schmidt, C. Keil, and J. Schönfelder, "A survey on honeypot software and data analysis," *arXiv preprint arXiv:1608.06249*, 2016.

[3] Decester, "An ssh honeypot," 2000. [Online]. Available: https://github.com/desaster/kippo

[4] M. Oosterhof, "Cowrie ssh/telnet honeypot," 2016. [Online]. Available: :https://github.com/micheloosterhof/cowrie

[5] L. Rist, "Glastopf project," 2009.

[6] D. Tools, "Web honeypot," 2010. [Online]. Available: https://github.com/DinoTools/dionaea/

[7] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, and M. Fischer, "Hostage: A mobile honeypot for collaborative defense," in *Proceedings of the 7th International Conference on Security of Information and Networks*, ser. SIN '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 330–333. [Online]. Available: https://doi.org/10.1145/2659651.2659663

[8] L. Rist, J. Vestergaard, D. Haslinger, A. Pasquale, and J. Smith, "Conpot ics/scada honeypot," *Honeynet Project (conpot. org)*, 2013.

[9] E. Vasilomanolakis, S. Srinivasa, C. G. Cordero, and M. Mühlhäuser, "Multi-stage attack detection and signature generation with ics honeypots," in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. Istanbul, Turkey: IEEE, 2016, pp. 1227–1232.

[10] L. Krämer, J. Krupp, D. Makita, T. Nishizoe, T. Koide, K. Yoshioka, and C. Rossow, "Amppot: Monitoring and defending against amplification ddos attacks," in *International Symposium on Recent Advances in Intrusion Detection*. Springer, 2015, pp. 615–636.

[11] S. Srinivasa, J. M. Pedersen, and E. Vasilomanolakis, "Riotpot: a modular hybrid-interaction iot/ot honeypot," in *26th European Symposium on Research in Computer Security (ESORICS) 2021*, Springer. Darmstadt, Germany: Springer, 2021.

[12] C. Zou and R. Cunningham, "Honeypot-aware advanced botnet construction and maintenance," in *International Conference on Dependable Systems and Networks (DSN'06)*. Philadelphia, PA, USA: IEEE, 2006, pp. 199–208.

[13] B. Botezatu, "New hide 'n seek iot botnet using custom-built peer-to-peer communication spotted in the wild," 2018. [Online]. Available: https://bit.ly/35QbK1P

[14] A. Vetterl and R. Clayton, "Bitter harvest: Systematically fingerprinting low- and medium-interaction honeypots at internet scale," in *12th USENIX Workshop on Offensive Technologies (WOOT 18)*. Baltimore, MD: USENIX Association, Aug. 2018, p. 9. [Online]. Available: https://www.usenix.org/conference/woot18/presentation/vetterl

[15] S. Morishita, T. Hoizumi, W. Ueno, R. Tanabe, C. Gañán, M. J. van Eeten, K. Yoshioka, and T. Matsumoto, "Detect me if you. . . oh wait. an internet-wide view of self-revealing honeypots," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. Arlington, VA, USA: IEEE, 2019, pp. 134–143.

[16] L. Spitzner, "Passive fingerprinting," pp. 1–4, 2000.

[17] SHODAN, "Shodan," 2021. [Online]. Available: https://www.shodan.io/

[18] G. Lyon, "Nmap network mapper," 2021. [Online]. Available: https://nmap.org/

[19] O. Arkin, F. Yarochkin, M. Kydyraliev, O. Arkin, F. Yarochkin, and M. Kydyraliev, "The present and future of xprobe2: The next generation of active operating system fingerprinting. sys-security group," 2003.

[20] F. Holik, J. Horalek, O. Marik, S. Neradova, and S. Zitta, "Effective penetration testing with metasploit framework and methodologies," in *2014 IEEE 15th International Symposium on Computational Intelligence and Informatics (CINTI)*, IEEE. Budapest, Hungary: IEEE, 2014, pp. 237–242.

[21] H. Van and K. Roland. (2021) Kali tools. THC.org. [Online]. Available: https://www.thc.org/thc-hydra/

[22] Censys, "Censys search," 2021. [Online]. Available: https://censys.io/

[23] Rapid7, "Recog," 2021.

[24] Z. Durumeric, E. Wustrow, and J. A. Halderman, "Zmap: Fast internet-wide scanning and its security applications," in *Proceedings of the 22nd USENIX Conference on Security*, ser. SEC'13. USA: USENIX Association, 2013, p. 605–620.

[25] R. D. Graham, "Masscan: Mass ip port scanner," 2014.

[26] L. Izhikevich, R. Teixeira, and Z. Durumeric, "Predicting ipv4 services across all ports," in *Proceedings of the ACM SIGCOMM 2022 Conference*, ser. SIGCOMM '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 503–515. [Online]. Available: https://doi.org/10.1145/3544216.3544249

[27] ZMap. (2020) Github. ZMap. [Online]. Available: https://github.com/zmap/zmap/wiki/Sample-Applications

[28] K. Papazis and N. Chilamkurti, "Detecting indicators of deception in emulated monitoring systems," *Service Oriented Computing and Applications*, vol. 13, no. 1, pp. 17–29, 2019.

[29] Domaintools. (2022) Domaintools whois lookup. Domaintools. [Online]. Available: https://whois.domaintools.com/

[30] T. Grudziecki, P. Jacewicz, Ł. JUSZCZYK, P. Kijewski, and P. Pawliński. (2012) Proactive detection of security incidents. ENISA. [Online]. Available: https://www.enisa.europa.eu/publications/proactive-detection-report

[31] M.-R. Zamiri-Gourabi, A. R. Qalaei, and B. A. Azad, "Gas what? i can see your gaspots. studying the fingerprintability of ics honeypots in the wild," in *Proceedings of the Fifth Annual Industrial Control System Security (ICSS) Workshop*, ser. ICSS. New York, NY, USA: Association for Computing Machinery, 2019, p. 30–37. [Online]. Available: https://doi.org/10.1145/3372318.3372322

[32] K. Wilhoit and S. Hilt. (2015) The gaspot experiment: Unexamined perils in using gas-tank-monitoring systems. Black Hat. USA.

[33] Cymmetria, "Mtpot," 2016. [Online]. Available: https://github.com/Cymmetria/MTPot

[34] H. Booth, D. Rike, and G. Witte, "The national vulnerability database (nvd): Overview," National Institute of Standards and Technology, Tech. Rep., 2013.

[35] R. Vogt, J. Aycock, and M. J. Jacobson Jr, "Army of botnets," in *NDSS*, Citeseer. San Diego, CA, USA: NDSS, 2007.

[36] Amazon. (2020) Alexa - an amazon company. Amazon. [Online]. Available: https://www.alexa.com/topsites

[37] Majestic. (2021) The majestic million. Majestic. [Online]. Available: https://majestic.com/reports/majestic-million

[38] Cisco. (2020) Umbrella popularity list. Cisco. [Online]. Available: https://umbrella-static.s3-us-west-1.amazonaws.com/index.html

[39] GNU.org. (2020) Gnu operating system. GNU.org. [Online]. Available: https://www.gnu.org/prep/ftp.en.html

[40] T. A. Foundation. (2020) The apache software foundation. Apache.org. [Online]. Available: https://www.apache.org/mirrors/

[41] O. A. M. for Ubuntu. (2020) Debian project. Ubuntu. [Online]. Available: https://launchpad.net/ubuntu/+archivemirrors

[42] Debian. (2020) Debian project. Debian Project. [Online]. Available: https://www.debian.org/mirror/list

[43] Hipo. (2020) Github. Hipo. [Online]. Available: https://github.com/Hipo/university-domains-list

[44] T. H. Project, "The honeynet project."

[45] J. Bethencourt, J. Franklin, and M. Vernon, "Mapping internet sensors with probe response attacks," in *14th USENIX Security Symposium (USENIX Security 05)*. Baltimore, MD: USENIX Association, jul 2005. [Online]. Available: https://www.usenix.org/conference/14th-usenix-security-symposium/mapping-internet-sensors-probe-response-attacks

[46] E. Vasilomanolakis, M. Stahn, C. G. Cordero, and M. Mühlhäuser, "On probe-response attacks in collaborative intrusion detection systems," in *2016 IEEE Conference on Communications and Network Security (CNS)*. Florence, Italy: IEEE, 2016, pp. 279–286.

[47] S. Srinivasa, J. M. Pedersen, and E. Vasilomanolakis, "Towards systematic honeytoken fingerprinting," in *13th International Conference on Security of Information and Networks*, ser. SIN 2020. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3433174.3433599

[48] T. Holz and F. Raynal, "Detecting honeypots and other suspicious environments," in *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*. West Point, NY, USA: IEEE, June 2005, pp. 29–36.

[49] P. Wang, L. Wu, R. Cunningham, and C. C. Zou, "Honeypot detection in advanced botnet attacks," *International Journal of Information and Computer Security*, vol. 4, no. 1, pp. 30–51, 2010.

[50] O. Hayatle, A. Youssef, and H. Otrok, "Dempster-shafer evidence combining for (anti)-honeypot technologies," *Information Security Journal: A Global Perspective*, vol. 21, no. 6, pp. 306–316, 2012.

[51] E. Aguirre-Anaya, G. Gallegos-García, N. S. Luna, and L. A. V. Vargas, "A new procedure to detect low interaction honeypots," *International Journal of Electrical and Computer Engineering*, vol. 4, pp. 848–857, 2014.

[52] A. Vetterl, R. Clayton, and I. Walden, "Counting outdated honeypots: Legal and useful," in *2019 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2019, pp. 224–229.

[53] C. Huang, J. Han, X. Zhang, and J. Liu, "Automatic identification of honeypot server using machine learning techniques," *Security and Communication Networks*, vol. 2019, 2019.

[54] Y. Sun, Z. Tian, M. Li, S. Su, X. Du, and M. Guizani, "Honeypot identification in softwarized industrial cyber–physical systems," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 8, pp. 5542–5551, 2021.

Paper B

Towards systematic honeytoken fingerprinting

Shreyas Srinivasa, Jens Myrup Pedersen, Emmanouil Vasilomanolakis

The layout has been revised.

Abstract

With the continuous rise in the numbers and sophistication of cyber-attacks, defenders are moving towards more proactive lines of defense. Deception methods such as honeypots and moving target defense paradigms, are nowadays utilized in a multitude of ways. A honeytoken is an umbrella term that describes honeypot-like entities/resources that can be inserted into a network or system. The moment an adversary interacts with a honeytoken, an alert is raised. Similar to honeypots, the value of honeytokens lies in their indistinguishability; if an attacker can detect them, e.g. via a fingerprinting tool, they can easily evade them. In this paper, we propose and discuss honeytoken fingerprinting methods. To the best of our knowledge, this is the first paper to examine honeytoken-specific fingerprinting. Furthermore, we showcase a proof of concept that is able to successfully detect a number of honeytoken types.

1 Introduction

Proactive defense mechanisms such as honeypots and moving target defense schemes have become a common additional line of defense. A honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource [1]. Over the years, a number of honeypot approaches have been proposed (e.g. [2–5]) for defending a multitude of protocols and systems (ranging from industrial control systems [6, 7] to IoT devices [8]).

Honeytoken is an umbrella term for a subset of honeypots in which there is no protocol or system emulation. Instead, a honeytoken usually emulates some resource (e.g. a file or a username/password) that is part of a real system and triggers an alert whenever it is accessed or used [9]. For example, a honeytoken can be a *.docx* file that contains an obfuscated script that is triggered when the file is opened.

An advantage of honeytokens over traditional honeypots is that they operate with lower system resources and are simpler to manage. In addition, they are easy to generate and deploy. Honeytokens can indirectly detect the presence of diverse attack vectors (e.g. malware) and identify direct attacks like unauthorised access attempts. Due to their simple design and flexibility, honeytokens are popular and are used by system administrators.

Over the years, there has been increase in honeytoken research including patents by commercial organizations [10–12]. Honeytokens can be modelled as files, directories, URLs, DNS entries, fake user accounts and fake data tuples in a database. While there is no limitation in the design, the core of honeytokens is to detect and notify users about unauthorized access. Some of the open-source and other research implementations include *Canarytokens* [13], *honeyλ* [14], *honeybits* [15], *HoneyGen* [16], *honeywords* [17] and lastly the *honeyfile* [18].

Recently, a number of researchers have discussed methods for fingerprinting honey-pots [19–21]. The purpose of these works is to generate some type of signature probe that is able to distinguish between a real system and a honeypot. While this research suggests that many traditional honeypots can be easily identified, it does not take honey-tokens into account. The key feature of honeytokens is that their alert logic is embedded within a real digital entity with fake contents. This makes honeytokens hard to identify as the only way of determining if an entity is a honeytoken is by utilizing it.

In this paper, we attempt a preliminary study on the possibility of fingerprinting honeytokens. We first classify the different honeytoken technologies in a systematic way and proceed by determining ways for their identification. Furthermore, we provide proof of concept experiments that demonstrate the feasibility of the proposed fingerprinting mechanisms. To the best of our knowledge this is the first paper to examine honeytoken fingerprinting.

The rest of the paper is structured as follows. Section 2 provides a background of honeytokens and honeytoken fingerprinting. In Section 3 we propose honeytoken fingerprinting techniques. We present a proof of concept by scanning of honeytokens using the proposed techniques in Section 4. We conclude our paper in Section 5 along with our future work goals.

2   Background

Since honeypots and honeytokens are flexible in their design and emulation approach, various concepts regarding their applicability and type have been proposed. Never-theless, the factor that distinguishes honeytokens from honeypots is their ability to detect threats by emulating low-level digital resources/entities like files, directories, user-accounts, and URLs. Honeypots operate at a higher level by emulating services and protocols that resemble a system or a service.

Fraunholz et al. survey deceptive technologies and provide a comprehensive overview of honeytokens as well [22]. The survey suggests that most proposals cover different types of entities and focus on the generation of deceptive digital twins. Furthermore, the authors present a classification that distinguishes between *server, database, authen-tication, and file* honeytokens. For example, the authors classify Honeyport [23] as a server-based honeytoken as it emulates an open network port within a server. Similarly, the honeytokens classified under database, authentication, and file, contain deceptive elements to emulate a data record, password, and a document respectively.

Han et al. also survey deception techniques in computer security [24]. The authors introduce a *multi-dimension* classification for honeypots, based on four orthogonal di-mensions: *goal, unit, layer, and deployment of deception.* Internal to the deployment dimension, a sub-class based on the deployment layer is relevant to honeytokens. The layer is further divided the into *network, system, application, and data* layers.

Based on the various honeytokens proposed in related research, we break down honeytokens' architecture into two primary mechanisms: *deception* and *alerting*. The deception mechanism is responsible for the emulation of the digital entity and the deception logic. The alerting mechanism focuses on the alert trigger mechanism responsible for notifying the user about the access attempt. The alerting mechanism is triggered when the adversary tries accessing the honeytoken or using the data generated as a honeytoken for an access attempt. Both deception and alerting mechanisms may vary based on the digital entity replicated.

Honeytoken	Deceptive Entity/Resource	Alerting Mechanism
Honeyentries [16], [25]	Table data set	DB Monitor
Honeyword [17]	Password	DB Monitor
Honeyaccount [26]	User-account	Event Logger
Honeyfile [18]	File-Google Sheets	Session Log
Honeyfile [23]	File	Event Logger
Honeypatch [27], [28]	Vulnerability	Session Log
HoneyURL [18]	URL	DNS Trigger
CanaryTrap [29]	Email	Email
Honeyport [23]	Network port	Session Log
CanaryToken [13]	File-pdf, docx	DNS Trigger
CanaryToken [13]	Directory	DNS Trigger
CanaryToken [13]	URL	DNS Trigger
Honeybits [15]	Email	DNS Trigger

Table B.1: Honeytoken-Mechanisms overview

Table B.1 provides an overview of the deception and alerting mechanisms employed in research-based and open-source honeytokens. The deceptive entity denotes the digital entity or resource that is emulated by the honeytoken. These vary from passwords, user-accounts, files, directories, email, software patches, URLs, network ports, etc. The alerting mechanism lists the triggering and notification technique employed by the honeytokens.

The *DB Monitor* monitors a dataset for changes and maintains an activity log. All changes and access information are logged respectively. The *Event Logger* operates at the system-level and maintains a log of all system events. User defined events can be logged on the *Event Logger*; this is supported by most modern operating systems. The *Session Log* operates at the application-level and logs all the events at user-defined log levels. These may vary from informational, debug, error and warning. *DNS Triggers* operate at the network-level by performing a name resolution query to a DNS server. The query includes a URL that triggers the alerting mechanism.

3 Honeytoken fingerprinting

We present generic techniques to detect honeytokens that operate at different levels. The proposed fingerprinting techniques leverage the gaps in both the deceptive entity and the honeytokens' alerting mechanism to determine if the entity is indeed a honeytoken. To understand the operating levels of the honeytokens, we classify the honeytokens (see Table B.1) into *Network Level*, *System Level*, *Application/File Level* and *Data Level*. Table B.2 provides an overview of the classification based on their operating level. The table also lists the fingerprinting techniques associated with each operational level corresponding to the alerting mechanism. In the following subsections we describe fingerprinting techniques on the basis of the various operating levels.

Alerting Mechanism	Operating Level	Fingerprinting Technique
DB Monitor	Data	Modified Date
Event Logger	System	Last Used, grep search
Session Log	Application	Grep search
DNS Trigger	Application, Network	Reverse Engineering Network Sniffing

Table B.2: Honeytokens Fingerprinting Overview

3.1 Network level

Honeytoken overview   Honeytokens operating at the network level are either replicating a networking entity or using the network for communicating the alerts to the administrator. For example, the Honeyport [23] emulates an open network port on a web server and uses the web server's session logs as the alerting mechanism. However, a honeytoken may operate at a different level (e.g. the *file level*) and use the network to communicate the alerts (e.g. Canarytoken [13]).

Network level fingerprinting   Considering the alerting mechanisms classified to operate at the network level in Table B.2, we observe the utilization of DNS. The honeytokens trigger a DNS resolution call made to a domain hardcoded within the embedded alerting mechanism upon detecting an access attempt. For example, a file-level Canarytoken contains an alerting mechanism that performs a DNS call upon opening the respective file. Fingerprinting these calls can be done by sniffing the DNS traffic on the compromised system. The DNS traffic will reveal the calls made to open-source honeytoken alert domains. However, inspecting the DNS traffic for calls made to honeytoken domains is a passive approach. Using this fingerprinting technique will notify the user

of the access attempt before identifying the honeytoken. In the following, we introduce active fingerprinting techniques that detect honeytokens without triggering an alert.

3.2 Application/File Level

Honeytoken overview The application/file level fingerprinting techniques focus on detecting honeytokens at the application or file level. These honeytokens operate by emulating a file of a specific format (e.g. *pdf* or *docx*) and obfuscating an alert mechanism within the file. The alert is triggered when the file is opened through specific applications like the *Adobe Reader*.

Application/File level fingerprinting File-level honeytokens using a network for alerting mechanisms can be fingerprinted by decomposing the files using reverse engineering techniques. For example, Canarytokens [13] that offer honeytokens as a *pdf* file format can be decomposed by file parsing techniques. On parsing the *pdf* file with a parsing tool from DidierStevensSuite we observe that the Canarytoken contains obfuscated DNS triggers to "canarytokens.net" in the /URI of the object stream [30]. We find similar obfuscated DNS triggers in other file formats like *docx*, which is offered from the open-source Canarytokens service. Adversaries can use file parsing techniques to explore the honeytokens for obfuscated code fragments that trigger the alert mechanisms. Similarly, a *honeydirectory*, a directory-emulating honeytoken from Canarytokens, can be identified by examining its meta-data.

3.3 System Level

Honeytoken overview Honeytokens that operate at the system level use the underlying operating system's features to facilitate the alert mechanism. Examples of the system features include *event-logs* and *inotify* alerts. Honeytokens like Honeyfile [18] and Honeyaccount [26] employ system-level triggers to alert the users.

System level fingerprinting Fingerprinting system-level alert mechanisms are complex because of their abstract calls and obfuscated deployments. Access monitors like *inotify* run as a background service that monitors a file or a directory for modifications. The *inotify* system calls are embedded within a C program and are initialized with a file descriptor, file path, and the mask modes as parameters. The mask modes offer options for the triggers like file accessed, modified, deleted, or created. The first step towards fingerprinting would be to check for *inotify* processes running in the background; i.e. the adversary has to list the background processes in the compromised system. Upon finding a process relevant to a C program execution, it is evident that there is an alerting mechanism setup. The adversary can open the C program's path, which calls *inotify* and check the file or directory path for changes.

3.4   Data level

Honeytoken overview   Data-level honeytokens work on the generation of fake data that resemble actual data. The generation of data-based honeytokens is complicated due to the requirement of high resemblance to real data. An example of this is the generation of employee data and access information. The honeytoken data that resembles the employee information and his access information must resemble a real employee record. Simultaneously, this data must be fake and attractive enough for an adversary. There have been many research proposals over algorithms and techniques for the generation of such data honeytokens [16, 26, 31].

Data level fingerprinting   The fingerprinting technique for detecting data-level honeytokens depends on the type of data emulated and the alerting mechanism used. Some research concentrate only on the generation of data honeytokens and not the alerting mechanism (e.g. HoneyGen [16]). Honeytokens like Honeyword [17] use a *Honeychecker* module that is responsible for comparing the password-hash used by the adversary with the list of passwords and triggering an alarm in case of unauthenticated access. While it is complicated to fingerprint honeywords, we propose using meta-data to determine if it is a real entity. For example, Honeyaccount [26] creates fake user-accounts for a system. On a compromised system that is running Windows and is attached to a domain, user accounts can be listed and checked for the last known activity of a user. In addition, the adversary can make use of specific scripts in the Windows PowerShell to retrieve meta-data about user accounts in the Active Directory. By observing the meta-data retrieved from the Active Directory, the adversary can identify if the user account is real or not.

4   Proof of Concept

This section demonstrates the applicability of some of the aforementioned honeytoken fingerprinting techniques. In particular, we demonstrate fingerprinting in one of the most used honeytoken implementation, the Canarytoken [13]. With respect to our classification we emphasize on *network* and *application/file level* fingerprinting. The source code and other screenshots of the proposed fingerprinting techniques can be found on our GitHub account*.

Firstly, we generate a *pdf* honeytoken by utilizing the Canarytoken service [13]. To support our claims (see Section 3.1) we manually monitor the network and open the generated pdf honeytoken. Figure B.1 shows the packets captured from a system when a Canarytoken is accessed in Wireshark. To avoid manual sniffing of all the network we implemented a honeytoken DNS sniffer (see GitHub for the implementation code)

---

*https://github.com/aau-network-security/tokengrabber
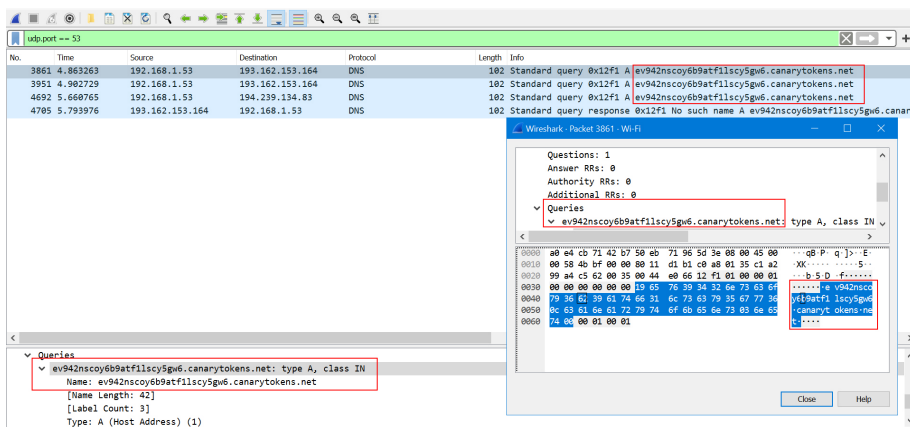
Fig. B.1: Network-level (DNS) Fingerprinting

that checks the DNS traffic of the system for calls made to known honeytoken services. We note here that if the adversary uses this method they risk triggering the honeytoken and therefore notifying the administrator.

For a stealthier option, the attacker may use file level fingerprinting techniques (see Section 3.2). We adopt the code of a pdf parser in [30], to identify honeytoken traces in a given pdf file. By using such an application-level fingerprinting technique, the *pdf* Canarytoken was parsed and the honeytoken was detected without triggering an alert to the administrator. A URI reference obfuscated in the pdf object 16 was detected. The URI referenced to [32] clearly indicates the call to a domain hosted by the Canarytokens service.

5 Conclusion

In this paper, we propose fingerprinting techniques against the majority of existing honeytoken proposals and implementations. Furthermore, as a proof of concept, we successfully fingerprint open-source honeytokens. This work provides a foundation to extend our research on honeytoken fingerprinting. In particular, for future work we plan to work on countermeasures against fingerprinting for the various honeytokens. Moreover, we will further examine the possible fingerprinting attacks against them, beyond the presented proof of concept.

References

[1] L. Spitzner, "Passive fingerprinting," pp. 1–4, 2000.

[2] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, and M. Fischer, "Hostage: a mobile honeypot for collaborative defense," in *Proceedings of the 7th International Conference on Security of Information and Networks*, 2014, pp. 330–333.

[3] M. Oosterhof, "Cowrie ssh/telnet honeypot," 2016. [Online]. Available: :https://github.com/micheloosterhof/cowrie

[4] D. Tools, "Web honeypot," 2010. [Online]. Available: https://github.com/DinoTools/dionaea/

[5] L. Rist, "Glastopf project," 2009.

[6] E. Vasilomanolakis, S. Srinivasa, and M. Mühlhäuser, "Did you really hack a nuclear power plant? an industrial control mobile honeypot," in *2015 IEEE Conference on Communications and Network Security (CNS)*.   IEEE, 2015, pp. 729–730.

[7] L. Rist, J. Vestergaard, D. Haslinger, A. Pasquale, and J. Smith, "Conpot ics/scada honeypot," *Honeynet Project (conpot. org)*, 2013.

[8] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "Iotpot: A novel honeypot for revealing current iot threats," *Journal of Information Processing*, vol. 24, no. 3, pp. 522–533, 2016.

[9] L. Spitzner, "Honeytokens: The other honeypot. 2003," *Internet: http://www. securityfocus. com/infocus/1713*, 2006.

[10] H. H. Neuvirth, T. Weinberger, Y. Zohar, C. A. Nelson, and A. E. Johnson, "Automated generation and deployment of honey tokens in provisioned resources on a remote computer resource platform," Sep. 10 2020, uS Patent App. 16/291,963.

[11] T. A. Be'ery and I. Grady, "Systems and methods for the detection of advanced attackers using client side honeytokens," Mar. 31 2020, uS Patent 10,609,048.

[12] S. Touboul, H. Levin, S. Roubach, A. Mischari, I. B. David, I. Avraham, A. Ozer, C. Kazaz, O. Israeli, O. Vingurt *et al.*, "Multi-factor deception management and detection for malicious actions in a computer network," Apr. 14 2020, uS Patent 10,623,442.

[13] Thinkst, "Canarytokens," https://github.com/thinkst/canarytokens.

[14] A. Karimi, "Honeylambda," https://github.com/0x4D31/honeyLambda.

[15] ——, "Honeybits," https://github.com/0x4D31/honeybits.

[16] M. Bercovitch, M. Renford, L. Hasson, A. Shabtai, L. Rokach, and Y. Elovici, "Honeygen: An automated honeytokens generator," in *Proceedings of 2011 IEEE International Conference on Intelligence and Security Informatics.* IEEE, 2011, pp. 131–136.

[17] A. Juels and R. L. Rivest, "Honeywords: Making password-cracking detectable," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 145–160.

[18] M. Lazarov, J. Onaolapo, and G. Stringhini, "Honey sheets: What happens to leaked google spreadsheets?" in *9th Workshop on Cyber Security Experimentation and Test ({CSET} 16)*, 2016.

[19] C. Huang, J. Han, X. Zhang, and J. Liu, "Automatic identification of honeypot server using machine learning techniques," *Security and Communication Networks*, vol. 2019, 2019.

[20] A. Vetterl and R. Clayton, "Bitter harvest: Systematically fingerprinting low-and medium-interaction honeypots at internet scale," in *12th {USENIX} Workshop on Offensive Technologies ({WOOT} 18)*, 2018.

[21] S. Morishita, T. Hoizumi, W. Ueno, R. Tanabe, C. Gañán, M. J. van Eeten, K. Yoshioka, and T. Matsumoto, "Detect me if you... oh wait. an internet-wide view of self-revealing honeypots," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM).* Arlington, VA, USA: IEEE, 2019, pp. 134–143.

[22] D. Fraunholz, S. D. Anton, C. Lipps, D. Reti, D. Krohmer, F. Pohl, M. Tammen, and H. D. Schotten, "Demystifying deception technology: A survey," *arXiv preprint arXiv:1804.06196*, 2018.

[23] D. Fraunholz, D. Krohmer, F. Pohl, and H. D. Schotten, "On the detection and handling of security incidents and perimeter breaches-a modular and flexible honeytoken based framework," in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS).* IEEE, 2018, pp. 1–4.

[24] X. Han, N. Kheir, and D. Balzarotti, "Deception techniques in computer security: A research perspective," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–36, 2018.

[25] M. G. Hoglund and S. M. Bracken, "Inoculator and antibody for computer security," Oct. 17 2017, uS Patent 9,792,444.

[26] C. D. Faveri and A. Moreira, "Visual modeling of cyber deception," in *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2018, pp. 205–209.

[27] F. Araujo, K. W. Hamlen, S. Biedermann, and S. Katzenbeisser, "From patches to honey-patches: Lightweight attacker misdirection, deception, and disinformation," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 2014, pp. 942–953.

[28] J. Avery and E. H. Spafford, "Ghost patches: Fake patches for fake vulnerabilities," in *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer, 2017, pp. 399–412.

[29] S. Farooqi, M. Musa, Z. Shafiq, and F. Zaffar, "Canarytrap: Detecting data misuse by third-party apps on online social networks," *Proceedings on Privacy Enhancing Technologies*, vol. 2020, no. 4, pp. 336–354, 2020.

[30] S. Didier. (2014) Pdf parser. [Online]. Available: https://github.com/DidierStevens/DidierStevensSuite/blob/master/pdf-parser.py

[31] J. White, "Creating personally identifiable honeytokens," in *Innovations and Advances in Computer Sciences and Engineering*. Springer, 2010, pp. 227–232.

[32] Canarytokens, "Canarytokens domain," https://ev942nscoy6b9atf1lscy5gw6.canarytokens.net/QBEINOXGQDLDQUOXILNWLUCAPCMWEAGOGJ.

Paper C

Honeysweeper: Towards stealthy Honeytoken fingerprinting techniques

Mohamed Msaad, Shreyas Srinivasa, Mikkel M. Andersen, David H. Audran, Charity U. Orji, Emmanouil Vasilomanolakis

The layout has been revised.

Abstract

The increased number of data breaches and sophisticated attacks have created a need for early detection mechanisms. Reports indicate that it may take up to 200 days to identify a data breach and entail average costs of up to $4.85 million. To cope with cyber-deception approaches like honeypots have been used for proactive attack detection and as a source of data for threat analysis. Honeytokens are a subset of honeypots that aim at creating deceptive layers for digital entities in the form of files and folders. Honeytokens are an important tool in the proactive identification of data breaches and intrusion detection as they raise an alert the moment a deceptive entity is accessed. In such deception-based defensive tools, it is key that the adversary does not detect the presence of deception. However, recent research shows that honeypots and honeytokens may be fingerprinted by adversaries. Honeytoken fingerprinting is the process of detecting the presence of honeytokens in a system without triggering an alert. In this work, we explore potential fingerprinting attacks against the most common open-source honeytokens. Our findings suggest that an advanced attacker can identify the majority of honeytokens without triggering an alert. Furthermore, we propose methods that help in improving the deception layer, the information received from the alerts, and the design of honeytokens.

1 Introduction

Cyber attacks have reached a record level in 2021, making it the highest in 17 years with a 10% increase from the previous year [1]. A $1.07 million cost increase is related to the spike in remote work due to the COVID-19 pandemic [2] in addition to the continuous growth of IoT devices [3, 4]. Further, the time needed to identify and contain a security breach may take up to 287 days [5]. To combat this, the cyber-defense community is moving toward more active lines of defense that leverage deception-based techniques. Deception techniques confuse and divert attackers from real assets by placing fake data and vulnerable systems across an organization's network. Any interaction with a deceptive entity may be considered an attack. In practice, there are two leading deception technologies: *honeypots* and *honeytokens*.

Honeypots are deceptive systems that emulate a vulnerable program [6–9], for instance, a vulnerable version of the Linux operating system (OS), an HTTP server, or an IoT device. They lure attackers and deflect them from real assets while gathering information about the techniques and tools used during the interaction. Honeypots differ by their low, medium or high-interaction level [10–12]. As the name implies, interaction refers to how much capabilities are offered to the adversary. The process of discovering the existence of a honeypot in a system is known as honeypot fingerprinting [11, 13]. The drawback of many honeypots is that their emulation of systems/protocols exposes

some artifacts that attackers can detect.

Honeytokens are digital entities that contain synthetic/fabricated data. They are usually stored in a system under attractive names as a trap for intruders, and any interaction with them is considered an attack. Honeytokens can be files such as PDFs, SQL database entries, URLs, or DNS records that embed a token. Once accessed they trigger and alert the system about the breach [14]. Additionally, honeytokens are less complex and easier to maintain when compared to honeypots.

The honeytokens' efficiency resides in their indistinguishability; hence, identifying that an entity is a honeytoken (known as fingerprinting), diminishes its value. In this paper, we explore and extend the research on honeytoken fingerprinting techniques and demonstrate a fingerprinting tool that can successfully fingerprint 14 out of 20 honeytokens offered by the most popular open-source honeytoken service. Our contributions in this work are as follows:

- We analyze the design of open-source honeytokens to identify potential gaps for fingerprinting purposes.

- We introduce additional techniques to detect open-source honeytokens without triggering alerts.

- We propose techniques to improve the deceptive capabilities of honeytokens and introduce features that can enhance the use of information received from alerts triggered by intrusions.

The rest of this paper is structured as follows. In Section 2, we discuss the background of the working mechanism and the fingerprinting mechanism of honeytokens. Section 3 summarizes the related work of honeytoken fingerprinting. Section 4 presents our proposed stealthy techniques for honeytoken fingerprinting. Moreover, in Section 5 we present a proof of concept for honeytoken fingerprinting. In Section 6, we discuss countermeasures against honeytoken fingerprinting. We conclude our work in Section 7.

2   Background

Cyber-deception is an emerging proactive cyber defense methodology. When well crafted, deception-based tools can be leveraged as source of threat intelligence data. Deception techniques have two correlated defense strategies: first, to diverge the attacker from tangible assets by simulating vulnerable systems to lure attackers and attract attention, protecting tangible assets from being attacked. Second, to notify about ongoing suspicious activities, which can minimize the impact of an attack.

Honeytokens are deceptive entities that work by essentially triggering a notification when the user initiates an action on them. The actions can vary depending on the

honeytoken type, such as read, write, query and others. The concept is to embed a token in the deceptive entity and rely on the deceptive layer to consume the token and trigger the alert. Figure C.1 shows the conceptual flow of a honeytoken. The honeytoken is deployed on a user's system at either OS, application, or network levels. On any attempt of access, the honeytoken triggers an alert to the user through the notification mechanism. The recipient's information is obtained by placing a request to the honeytoken service. The honeytoken service acts as an endpoint and provides a back-end for managing the honeytokens and the metadata of the deployed honeytokens. Upon obtaining recipient information, a notification is sent either as an email or a text message.
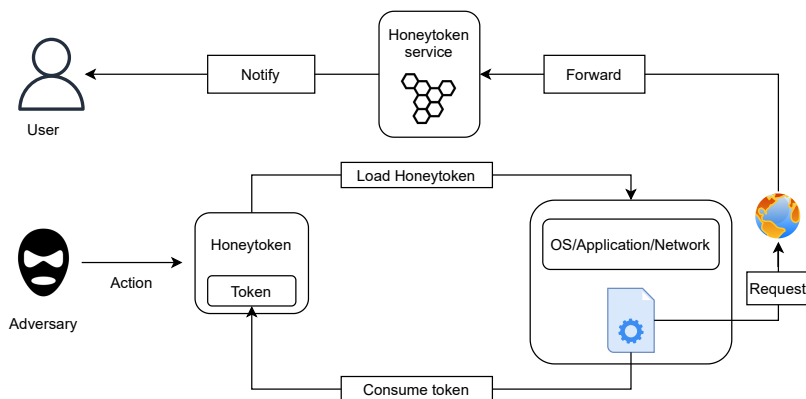
Fig. C.1: Honeytoken concept and alert mechanism

To explain the honeytoken mechanism in detail, we use the Canarytokens (honeytokens service) as a case study to provide concrete examples. Canarytokens is an open-source honeytoken provider that offers 20 different honeytoken types. All the honeytokens provided share the same deployment life-cycle as illustrated in Figure C.2.

To explain the deceptive layer and trigger mechanism, we use the PDF honeytoken from the Canarytoken service. The Adobe Acrobat Reader (AAR) offers a range of functionality for the PDF format to increase the document's interaction. One of these functionalities is the URI function, which allows linking a local URI to the world wide web via the AAR plugin Weblink [15]. The weblink plugin exposes its functionalities to other applications through the Host-Function-Table API. Once the honeytoken is accessed with AAR, the URL is loaded by the weblink plugin, which on its turn will start a DNS request to resolve the domain name. This DNS request will alert the owner of the PDF honeytoken.

Unlike honeypots, honeytokens are accessible only if the attacker is within the system where the honeytokens reside. The attacker can gain access through an attack or be an
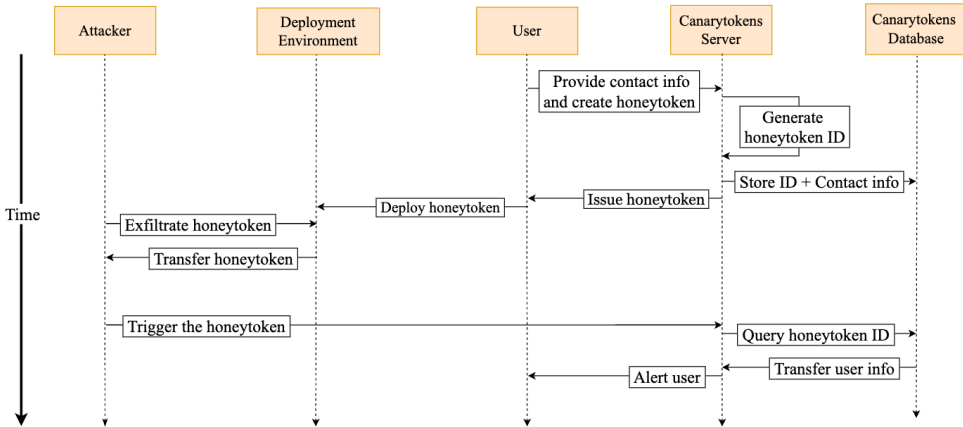
Fig. C.2: Canarytokens life-cycle

insider. In both cases, honeytokens are very useful as an early alarm against successful data exfiltration if triggered.

3   Related work

Since the invention of deception techniques, much research has been proposed for fingerprinting the deceptive entities [11, 16–18]. These fingerprinting techniques fall into two categories: passive and active fingerprinting. Passive techniques do not require interaction with the deceptive entity and focus on monitoring. However, active fingerprinting can be either stealthy or noisy. We define stealthy fingerprinting as the process of revealing a deceptive mechanism without triggering any alarm.

3.1   Honeypot fingerprinting

Holz et al. list some artifacts produced by the honeypot simulation to detect a honeypot [19]. For instance, by verifying the User-Mode-Linux (UML). UML is a way of having a Linux kernel running on another Linux. The initial Linux kernel is the host OS, and the other is the guest OS. By default, the UML executes in Tracing Thread mode (TT) and is not designed to be hidden and can be used to check for all the processes started by the host OS main thread. By executing the command: *"ps a"*, one can retrieve a list of processes and identify UML usage's existence. Another sign of UML is the usage of the TUN/TAP back-end for the network, which is not common on a real system and

can identify UML usage. Another place to look for artifacts is at the file *proc/self/maps* that contains the current mapped memory regions on a Linux system. On a real OS, the end of the stack is usually *0xc0000000*, which is not the case on a guest OS. These artifacts can be used against honeypots, rendering them visible to the attacker.

Other fingerprinting techniques, such as the network latency comparison, focus on the network layer. For instance, by calculating the differences between an HTTP server and a honeypot HTTP server. Mukkamala et al. utilized timing analysis to reveal if a program is a honeypot. Comparing the timing analysis of ICMP echo requests, they showcased that an HTTP-server honeypot will respond slower than a real HTTP-server [20]. In another work by Srinivasa et al., a framework for fingerprinting different honeypots is proposed. The utilized techniques include so-called probe-based fingerprinting (such as port-scans or banner-checks), and metascan-based fingerprinting (e.g., using data from the Shodan API) [13].

3.2   Honeytoken fingerprinting

Honeytokens can take the form of different data types, such as files, database entries, and URL/DNS records. The first step of fingerprinting is to classify honeytokens to build a standard fingerprint method for each type. Fraunholz et al. have classified honeytokens based on the entity type it emulates [21]. For instance, so-called honeypatches are classified as server-based honeytokens as they emulate a vulnerable decoy. The decoy may host monitoring software that collects important attack information and deceptive files that misinform the attackers. The attacker is redirected to a decoy once the system detects an exploit. Similarly, the database, authentication, and file honeytokens emulate data records and authentication credentials, such as passwords and documents. Similarly, Han et al. proposed a multi-dimensional classification of deception techniques based on the goal, unit, layer, and deployment of the deception [22]. The majority of the surveyed honeytokens are classified based on the detection goal. However, they differ in the four deception layers— the network, system, application, and data layer. In another work, Zhang et al. proposed a two-dimensional taxonomy, which eases the systematic review of representative approaches in a threat-oriented mode, namely from the domains of honeypots, honeytokens, and MTD techniques. They classify deception techniques depending on which phase of the Cyber Kill-Chain they can deceive an attacker. Honeytokens can be used in eight out of twelve phases to deceive attackers [23].

To the best of our knowledge, the only work that examines honeytoken-specific fingerprinting to date is by Srinivasa et al. [24]. The work showcases a proof of concept regarding fingerprinting a public honeytoken provider as a case study. Additionally, they suggest a honeytoken classification based on the four levels of operation and their fingerprinting technique, respectively:

- **Network level**: The honeytokens operating on this level emulate a network entity or use the network as the channel for delivering the alerts. The respective

fingerprinting technique for this deceptive layer relies on sniffing the network traffic to detect such calls. In their example with the PDF honeytoken, Srinivasa et al. observed the usage of DNS queries. However, this fingerprinting method remains passive and not stealthy as it leads to triggering the alert.

- **Application/File-Level**: These honeytokens take the format of a specific file, e.g., PDF or DOCX, and obfuscate an alert mechanism within the file. The alert is triggered if specific applications like Adobe Reader or Microsoft Word opens the honeytoken. The fingerprinting techniques relies on file decompression and obtaining the file honeytoken metadata.

- **System-Level**: These honeytokens utilize operating systems' features such as event logs and *inotify* calls as alert mechanisms. For fingerprinting these, Srinivasa et al. suggest monitoring background-running processes to check for the *inotify* call and to look out for changes in the file or the directory path.

- **Data-Level**: These honeytokens emulate data and can be hard to distinguish from actual data. The technique for fingerprinting honeytokens operating on the data level could vary depending on the data emulated and its alert mechanism. However, as mentioned by Srinivasa et al., viewing the file's meta-data can help an attacker determine whether the file is a possible honeytoken. For instance, Honeyaccount [25] creates fake user-accounts for a system to deceive attackers in using them and hence trigger the alert. On a compromised Windows machine, adversaries can list the user accounts to verify the last known activity. Additionally, adversaries can use Windows PowerShell scripts to recover meta-data about the accounts in Active Directory. This can assist in identifying fake user accounts.

Srinivasa et al. also present different fingerprinting techniques for each honeytoken type. For instance, to fingerprint a PDF honeytoken and determine its trigger channel, they monitored the network traffic when interacting with the file. This fingerprinting technique is noisy as the honeytoken triggers after the interaction. However, a stealthier fingerprinting approach for the same honeytoken was also applied. They used a PDF parser* to extract information from the PDF stream. The information consisted of a URL where the domain name belonged to the honeytoken provider. All their proposed fingerprinting techniques relied only on black box testing (i.e., triggering the honeytoken to find the deceptive layer and the alerting mechanism). Lastly, the authors did not consider multiple honeytokens but focused only on a few as a base for their proof of concept.

---

*https://github.com/DidierStevens/DidierStevensSuite/blob/master/pdf-parser.py

4   Methodology

To build the fingerprinting techniques, we used different methods to extract information from the honeytoken implementation. The methods include white box and black box testing.

4.1   Honeytoken Analysis

To analyze the honeytokens, we started by building a classification to help us create fingerprinting techniques for each honeytoken class. Srinivasa et al. have established a Canarytoken honeytoken classification, and we use it as a building block for our extended version [24].

In particular, we extend the previous classification and propose a new one that maps all the publicly offered honeytokens from Canarytokens, as shown in Table C.1. We added the dependency layer as a category of classification. The dependency can be at the application or the OS layer. The PDF, *.docx* honeytokens can only trigger when used with a specific application. For instance, *.docx* will only trigger with the application Microsoft Word and would not if opened with the online version Microsoft 365, concluding that it is an application-dependent honeytoken. In contrast, other honeytokens, such as the SQL-DUMP, will trigger with any query from an SQL-capable application. This classification also relates to the privileges needed to stop the triggering mechanism (e.g., the OS-dependent honeytokens will require higher privileges to interrupt the trigger process than the application-dependent ones).

The first analysis step is to classify the honeytokens based on their underlying operation. We leverage the syntax form of the token as the base for the classification. From all the 20 available honeytokens, we find four base usages: DNS, URL, SMTP, IP, and access keys base.

The second step is to classify the honeytokens based on the location of the honeytoken identifier in the token. After analyzing all the URL/DNS-based honeytokens, we observed that the token is a subdomain or a path identifier in the URL. This brought us to conclude the trigger channel based on the location. Subdomain honeytokens will use DNS as a trigger channel, while the URL honeytokens will use the HTTP protocol.

With the classification as a base, we focus on developing fingerprinting techniques that target the dependency layer and the trigger channel. We use white and black box testing in our methodology to identify the gap in the implementation of the honeytokens that can be leveraged for developing fingerprinting techniques.

White box testing

The Canarytokens (honeytoken provider) service is open source, and we used white box testing to investigate the implementation to find artifacts. In particular, we utilized

Honeytoken Base	Honeytoken Name	Trigger Channel	Alerting Entity	Dependency Layer
DNS Subdomain Based	Acrobat Reader PDF	DNS	Adobe Acrobat Reader & Others	Application
	Custom .exe/ Binary		Windows User Access Control	OS
	MySQL Dump		SQL Server	None
	SQL Server		SQL Server	None
	DNS		DNS Server	None
	Windows Folder		Windows File Explorer	OS
	SVN Server		SVN Server	None
URL Based	Windows Word Document	HTTP	Microsoft Word Desktop Application	Application
	Windows Excel Document		Microsoft Excel Desktop Application	Application
	QR Code Fast Redirect Slow Redirect URL Custom Image Web Bug Cloned Website		Web Browsers, Curl & others	None
SMTP Based	Email Address	SMTP	SMTP Server	None
IP Based	Kubernetes Config File	TLS	Kubernetes Application	
	Wireguard Config File	Wireguard Protocol	Wireguard Application	
Access Key Based	AWS Key	CloudWatch	CloudWatch	Application

Table C.1: Extended Canarytokens classification

manual static analysis to check the honeytokens' generation code for any predicted output or patterns that can be used as a fingerprinting base. From our testing, we discover the following:

- ID length: We identify the usage of a fixed length in the honeytoken ID.

- Hardcoded data: We analyzed the source code to search for hardcoded data in the honeytoken's generation process. For instance, upon analyzing the code for the *.exe* file honeytoken, we discover the usage of hardcoded data used to generate a certificate.

- Template file usage: Canarytokens use a template file to generate the PDF, *.docx* and *.xlsx* honeytokens. This template is not changed and leads to static metadata that can be fingerprinted.

- File size: This is a result of the template file usage and constant file size. We consider this an additional artifact to the template to enhance the probability of accurate fingerprinting.

Black box testing

The black box testing did not focus on testing the system's internals. Instead, we used it to extract additional information that is only available after the honeytoken generation and validate our findings. The black box included creating and interacting with the honeytoken to reveal the trigger channel and the entity responsible for triggering the alert. The implemented techniques are as follows:

- Extracting metadata from the honeytokens to inspect if there are any static metadata present.

- Monitoring the network traffic when triggering a honeytoken to discover the trigger channel and confirm the white box testing findings.

- Monitoring what sub-processes were started by the application or the OS that triggers the honeytoken. This gives us an idea of how to circumvent the trigger mechanism and stop the honeytoken alert if possible.

With the knowledge gained from the black box, we classify the honeytokens into three categories depending on the token base: URL/DNS, IP, and access key based. The URL/DNS-based honeytokens have a URL or a DNS subdomain directly in the data or the file's metadata. Regardless of the honeytoken type, they all have the same domain name, canarytokens.com, or the equivalent IP address. The access key is a simple AWS access key with an identifier to link the user information with the honeytoken.

4.2 Honeytoken Fingerprinting

The first step is to be able to fingerprint honeytokens generated from the official website of Canarytokens[†]. We create and download all possible honeytokens to familiarize ourselves and gain information about all the different honeytokens offered by the Canarytokens service. In particular, we are interested on the underlying trigger mechanism, the trigger channels, and the honeytoken dependency.

To begin, the fingerprinting technique was a simple keyword search in the honeytoken data. The keyword is usually related to the honeytoken provider or publicly known information. We searched for the "canarytokens" keyword in the data or the metadata of all the URL/DNS base honeytokens. Regarding the IP-based honeytokens, our initial fingerprinting method was to perform a reverse DNS lookup of the "canarytokens.com" domain name and compare it to the one in the honeytoken. Finally, we did not discover

---

[†]https://canarytokens.org/generate

any fingerprinting strategy for the access key-based honeytokens since all the information related to the access key, since the all the information is saved at the server of the access key provider, except for a repeated pattern in the AWS key ID as displayed in Listing C.1. The identifier has 12 constant characters *AKIAYVP4CIPP*, which can be used to fingerprint all the AWS keys originating from Canarytokens.

```
 1   # 1st key
 2   [default]
 3   aws_access_key_id = [AKIAYVP4CIPP]G6FXFYHS
 4   aws_secret_access_key = UDxJeQftE3ekx+KS7skayD8MuN6CVVxOuemuxBSB
 5   output = json
 6   region = us-east-2
 7
 8   # 2nd-key
 9   [default]
10   aws_access_key_id = [AKIAYVP4CIPP]CF45DQPM
11   aws_secret_access_key = 8iTskHJBDDnYpUt1a2KY/hTlbScFoAS51cJl4nO5
12   output = json
13   region = us-east-2
14
15   # 3rd-key
16   [default]
17   aws_access_key_id = [AKIAYVP4CIPP]A3TB575H
18   aws_secret_access_key = mb8HpotCq27p4rCsQGwYpXoOxx+oQcIMpjdT+qOJ
19   output = json
20   region = us-east-2
```

Listing C.1: Canarytokens AWS access key repeated characters

The second major milestone is fingerprinting the honeytokens regardless of the domain name. We use the Canarytokens source code to set up the honeytoken service on our private honeytoken server. The keyword search or the IP address comparison approach is ineffective with this setup. However, the keyword search is still valid for the *.exe/.dll* honeytoken files due to the hardcoded data found in the certificate generation source code.

As mentioned before, the white box testing revealed that the URL/DNS- based honeytokens follow a specific pattern. The DNS/URL contains a 25-character alphanumeric identifier (ID) as displayed in Table C.2, which is used to link the honeytoken with the user's contact information. The ID is the subdomain for the DNS-based honeytokens and is the path for the URL-based ones. The placement of the URL/DNS value in the honeytoken is known to us. However, there are other URLs/DNS in some honeytokens. For instance, the URL in the *.docx* honeytoken resides in the metadata, which already includes other URLs to microsoft.com. In order to determine the existence of a honeytoken URL, we loop through each URL and see if they have a 25-character alphanumeric string in the DNS/URL. If they do, we label it as a possible honeytoken URL.

Our analysis suggests that the file type honeytokens use a static template to generate the PDF, *.docx*, and *.xlsx* files. For instance, the *template.pdf* file in the source code leads

Table C.2: URL/DNS Honeytokens followed pattern

Identifier	uq3501pu9mo56obz6kn5auhpq
URL	http://domain.name/url/path/ uq3501pu9mo56obz6kn5auhpq/contact.php
DNS	uq3501pu9mo56obz6kn5auhpq.domain.name

to constant metadata in the PDF honeytoken. Normally, some metadata attributes, such as the Document UUID, should be unique for each file. A constant UUID will make it easy to identify any PDF file from Canarytokens, even if the domain name is private. Additionally, other data can make the attacker more confident that this is a honeytoken file (e.g., created and modified dates). However, the file creation and modification dates are old (7 years), and any data in it might not be valid anymore from the attacker's point of view. See Appendix Listing C.2 for more details.

The Canarytokens implementation uses template files to generate all the file type honeytokens, which results in fixed file sizes. We observe that all the PDF, *.docx*, and *.xlsx* have the same size of 5KB, 15KB, and 7.7KB respectively. This additional artifact can be used with the template static metadata to raise the confidence of our fingerprinting method. Additionally, this constant small file size indicates that the file is empty and may not lure the attacker into interacting with it.

5 Proof of concept: *honeysweeper*

This section demonstrates the applicability of our honeytokens' fingerprinting techniques based on the Canarytoken implementation [26]. The fingerprinting tool's, namely *honeysweeper*, source code is available at our GitHub repository[‡].

5.1 Overview

From all the information gained from the black/white box testing, we built an OS-independent tool that can successfully fingerprint 14 out of the 20 honeytokens offered by Canarytokens. The tool relies on a primary fingerprinting technique matching the 25-character string identifier. However, this fingerprint method introduces the problem of false positives. As we discussed earlier, some honeytokens (i.e., file-type ones) contain more than one URL/DNS. If by any chance, another link contains a 25 characters string, the tool will label it as a possible honeytoken. Nevertheless, from an attacker's perspective, we argue that false negatives are more critical since they would raise an alarm.

[‡] https://github.com/aau-network-security/canarytokens_finger_printer

Honeysweeper begins by revealing the honeytoken extension for the file-type ones and then extracting the DNS/URL. URL/DNS/Email honeytokens can be added in a text file and passed to the tool. As in the case of PDF, *.docx* and *.xlsx* files, the tool needs to decompress the file as shown in Appendix Listings C.3 - C.4, and loops through each file to extracts all the tokens. Once obtained, *honeysweeper* runs the ___*find__canarytoken(string)* to match any pattern that matches the 25-character string in the honeytoken content. The PDF, *.docx*, and *.exe/.dll* honeytokens have higher confidence due to the earlier additional artifacts, i.e., the static template as shown in Appendix Listing C.2 and the small file size as shown in Figure C.3. The tool includes checks for the PDF template as a proof of concept and can easily be enhanced to detect other files such as *.docx* and *.xlsx.*

Name	Size	Kind
9y2jr13ve8xjwg292euv36kiu.pdf	5 KB	PDF Document
79a4ust07x55z4knhjz4xofri.pdf	5 KB	PDF Document
jqntearfpyxonyk432xrdy4ut.pdf	5 KB	PDF Document
kkavvzp3ezlanse4dsumelz24.pdf	5 KB	PDF Document
ranmd5nmfnoij6ibbdoxs1use.pdf	5 KB	PDF Document
uekkk4dxseegb89l5hxo2ezak.pdf	5 KB	PDF Document
ulmmft7fw5wlji429w2c5sexj.pdf	5 KB	PDF Document
y0jkqbkr210dni9ivqo3eofel.pdf	5 KB	PDF Document
anht4ag0x24ob21artvndffeo.xlsx	8 KB	Spreadsheet
dwld1o5p9sjmdlk0ezg14orw4.xlsx	8 KB	Spreadsheet
ed23csngpg0qsnss5yfpeinaa.xlsx	8 KB	Spreadsheet
6hjjqoeaselut4wb2qhsgjxxw.docx	16 KB	Micros...(.docx)
cbyam7ftvi7wbnadni9ptnoik.docx	16 KB	Micros...(.docx)
s6b8enlmn9o3v27vg292qyxer.docx	16 KB	Micros...(.docx)

Fig. C.3: Honeytokens file-type constant size artifact

5.2   Limitations

The Wireguard and Kubernetes honeytokens are not included in *honeysweeper* as we found no possible way of fingerprinting them when deployed with a private IP. All the data in the honeytokens are randomly generated, e.g., the public and private keys. However, this technique remains effective if the honeytokens are deployed with a known honeytoken provider IP address. The fingerprinting techniques for SVN and SQL-server are not included in the fingerprinting tool since both honeytokens are not directly accessible to the attacker. A possible fingerprinting method for the SQL server can be

to check the size of the table where the honeytoken resides. If the table is empty, it may not deceive the attacker for any further interaction. The other honeytokens e.g., PDF, *.docx*, and SQL-dump are available directly on the system and the fingerprinting methods are covered in *honeysweeper*.

6   Countermeasures against fingerprinting

The fixed ID length is the primary artifact shared among the studied honeytokens. We propose that the honeytoken identifier should be randomized in length or set in a range. For instance, the ID length could be between 25 and 32 characters, making the fingerprinting process harder and removing the 25-character ID artifact. This mitigation is valid for all the honeytokens containing a URL/DNS with 25 character identifiers. However, this only solves one problem.

The following recommendations are valid for all the template-dependent honeytokens. The PDF honeytokens should have random metadata. In the case of PDF, the attacker can generate a PDF Canarytokens and compare it to any PDF exfiltrated. Even if the honeytoken administrator changes the domain name and removes the 25-character ID artifact, the metadata alone is enough to raise suspicion. To address this, we propose to randomize the PDF XMP metadata. There are a few rules to keep the metadata consistent and not leave a metadata-modification footprint [27]. We present our solution in Appendix Listing C.5.

Moreover, the honeytoken administrator should modify the content of the *.docx*, *.xlsx*, and PDF files before deployment to change the document size which are *.docx* files are always 15KB, the *.xlsx* files with 7.7KB, and the PDF files with 5KB. Once modified, the honeytokens will resemble an actual file with data and lure the attacker into opening it. Otherwise, the attacker can combine the honeytoken file size with other artifacts to ensure the existence of a trap.

The signing process for the *.exe/binary* honeytokens should be with certificates unrelated to any honeytoken provider. As seen in the Canarytokens source code, a new certificate is generated to sign the *.exe/.dll* files. We generate an executable honeytoken using the source code locally to investigate the generation process. We see that a private key and a certificate is generated to sign the honeytoken and are removed after the process is complete. Nevertheless, the information included in the signature is hard-coded. Figure [C.4] shows the hard-coded information in the certificate. This hard-coded information will be the same for all the *.exe*/binary honeytokens and can be an artifact.

When deploying the stored procedure for a table on the SQL server, the administrator can set explicit permissions on the stored procedure by denying the public users from viewing the stored procedure's definition. The same approach applies for the SQL functions as a honeytoken. The function permission can be fragmented. For example, allow the public to select the functions and views but disallow viewing the definitions

```
[ ca_dn ]
countryName             = "ZA"
organizationName        = "Thinkst Applied Research"
organizationalUnitName  = "Thinkst Applied Research CA"
commonName              = "Thinkst Root CA"
```

Fig. C.4: Certificate hardcoded data

(syntax). Additionally, the trap table should be populated with random fake data to lure the attacker into interacting with it.

The Wireguard and Kubernetes honeytokens should use an IP address not linked with a honeytoken domain name. If no domain name is available and there is no alternative but to use the Canarytokens servers due to development and maintenance costs, an administrator can use a local server IP and redirect the traffic to Canarytokens servers.

7 Conclusion

Deception techniques like honeytokens are an essential extra layer of defense, and deploying them is becoming more and more common. However, for the deception technique to achieve its goal, it should be well crafted to deceive and should not include easy to exploit fingerprinting artifacts. This paper proposes fingerprinting techniques against most existing Canarytokens' honeytokens proposals and implementations. We analyze all the publicly offered honeytokens and propose countermeasures against the suggested techniques. As ethical disclosure, we informed Canarytokens of our findings. For future work, we plan on exploring other fingerprinting methods. For instance, the signature verification of the *.exe/.dll* files and other techniques. Additionally, we consider improving the honeytoken ID generation process by including a non-repudiation concept.

Appendix

Static data on PDF Canarytoken

Listing C.2 shows the static data identified on parsing the composite XML file of the PDF Canarytoken. We can observe static data on the modify date, create date and metadata date.

```
1  <x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="Adobe XMP Core 5.6-c015
       81.157285, 2014/12/12-00:43:15         ">
2    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
3      <rdf:Description rdf:about=""
4            xmlns:xmp="http://ns.adobe.com/xap/1.0/"
5            xmlns:dc="http://purl.org/dc/elements/1.1/"
6            xmlns:xmpMM="http://ns.adobe.com/xap/1.0/mm/"
7            xmlns:pdf="http://ns.adobe.com/pdf/1.3/">
8        <xmp:ModifyDate>2015-07-22T16:41:31+02:00</xmp:ModifyDate>
9        <xmp:CreateDate>2015-07-22T16:38:51+02:00</xmp:CreateDate>
10       <xmp:MetadataDate>2015-07-22T16:41:31+02:00</xmp:MetadataDate>
11       <xmp:CreatorTool>Acrobat Pro 15.8.20082</xmp:CreatorTool>
12       <dc:format>application/pdf</dc:format>
13       <xmpMM:DocumentID>uuid:a2364080-b5a8-1b46-b156-ea05c4972d03</
              xmpMM:DocumentID>=
14       <xmpMM:InstanceID>uuid:7656c56e-b1e6-f444-801f-06e28a50831f</
              xmpMM:InstanceID>
15       <pdf:Producer>Acrobat Pro 15.8.20082</pdf:Producer>
16     </rdf:Description>
17   </rdf:RDF>
18 </x:xmpmeta>
```

Listing C.2: PDF honeytoken static metadata

Fingerprinting of PDF Canarytoken

Listing C.3 shows the pseudo code for fingerprinting of PDF Canarytoken. The method checks for URLs embedded in the PDF and against a list of known honeytoken service URLs.

```
1  def find_token_in_pdf(file_location):
2      check_template(file_location) # check for template artifact
3      # List for URLs found
4      list_of_urls = []
5      pdf = open(file_location, "rb").read()
6      stream = re.compile(b'.*?FlateDecode.*?stream(.*?)endstream', re.S)
7      for s in re.findall(stream, pdf):
8          s = s.strip(b'\r\n')
9          line = ""
10         try:
```

```
11              line = zlib.decompress(s).decode('latin-1') # changed this
                    from UTF-8 to latin-1 as it throws errors. We
12              # want the app to be silent :)
13          except Exception as e:
14              print(e)
15          token = Tokenfinder.find_tokens_in_string(line)
16          if token:
17              list_of_urls.extend(token)
18      if len(list_of_urls) == 0:
19          print("No canaries detected")
20          return None
21      else:
22          print(str(len(list_of_urls)) + " canary URLs detected in the
                file")
23          for url in list_of_urls:
24              print("Canary detected!: ", url)
25              print()
```

Listing C.3: PDF fingerprinting

Fingerprinting of .docx and .xlsx Canarytokens

Listing C.4 shows the pseudo code for fingerprinting of .docx and .xlsx Canarytokens. The techniques unzips the composite file formats to check for URLs embedded in the files.

```
1   def check_office_files(file_location):
2       list_of_urls = [] # List to hold all urls in the file
3       try:
4           # Unzip the office file without saving to folder
5           unzipped_file = zipfile.ZipFile(file_location,"r")
6           # List of all the content of the zip
7           namelist = unzipped_file.namelist()
8           # Reads every file in the zip file and looks if it contains the
                string you wish to search for
9           for item in namelist:
10              content = str(unzipped_file.read(item))
11              token = Tokenfinder.find_tokens_in_string(content)
12              if token:
13                  list_of_urls.extend(token)
14      except OSError as e:
15          print(f"Exception: {e}")
16      # If no results of the search
17      if len(list_of_urls) == 0:
18          return None
19      else:
20          print(str(len(list_of_urls)) +" canary URLs detected in the
                file")
21          for url in list_of_urls:
22              print("Canary detected: ", url)
```

```
23                    print ()
```

Listing C.4: .docx and .xlsx fingerprinting

Mitigation of metadata in Canarytoken

Listing C.5 shows the mitigation by randomization of the file creation date and time. The randomness avoids static creation dates that is implemented by Canarytokens.

```
1  from pikepdf import Pdf
2  import  uuid , random , datetime , os
3
4  # make creation date with random Time-Zone [+1 to +3]
5  def creation_date ():
6      time = datetime.datetime.now ()
7      rand_region =str (random.randint (1, 3))
8      stamp = time.strftime ('2022-%m-%d')+'T'+ time.strftime ('%H:%M:%S')
           + '+0'+ rand_region+ ':00'
9      return stamp
10
11
12 def modification_date ():
13      time = datetime.datetime.now ()
14      return time.strftime ('%Y-%m-%d')+'T'+ time.strftime ('%H:%M:%S')
15
16 def add_metadata (source_pdf , out_dir ):
17      mod_date = modification_date ()
18      with Pdf.open (source_pdf) as pdf:
19          with pdf.open_metadata (set_pikepdf_as_editor=False) as meta:
20              meta['xmp:CreatorTool'] = 'Acrobat Pro 22.001.20112'
21              meta['xmpMM:DocumentID'] = str (uuid.uuid4 ())
22              meta['xmpMM:InstanceID'] = str (uuid.uuid4 ())
23              meta['xmp:CreateDate'] = creation_date ()
24              meta['xmp:ModifyDate'] = mod_date
25              meta['xmp:MetadataDate'] = mod_date
26              meta['pdf:Producer'] = 'Acrobat Pro 22.001.20112'
27      pdf.save (os.path.join (out_dir , os.path.basename (source_pdf )))
28      print ('Done!')
29
30 source_pdf = "/Users/mm/Downloads/pdftoken.pdf"
31 out_dir = '/Users/mm/Desktop/'
32 add_metadata (source_pdf , out_dir)
```

Listing C.5: Metadata mitigation

References

[1] IBM, "Insights into what drives data breach costs," 2021. [Online]. Available: https://www.ibm.com/account/reg/uk-en/signup?formid=urx-51643

[2] ——, "Key findings," 2021. [Online]. Available: https://www.ibm.com/downloads/cas/OJDVQGRY

[3] K. Ghirardello, C. Maple, D. Ng, and P. Kearney, "Cyber security of smart homes: Development of a reference architecture for attack surface analysis," in *Living in the Internet of Things: Cybersecurity of the IoT - 2018*, March 2018, pp. 1–10.

[4] S. Srinivasa, J. M. Pedersen, and E. Vasilomanolakis, "Open for hire: Attack trends and misconfiguration pitfalls of iot devices," in *Proceedings of the 21st ACM Internet Measurement Conference*, ser. IMC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 195–215. [Online]. Available: https://doi.org/10.1145/3487552.3487833

[5] IBM, "How much does a data breach cost?" 2021. [Online]. Available: https://www.ibm.com/security/data-breach

[6] I. Mokube and M. Adams, "Honeypots: Concepts, approaches, and challenges," in *Proceedings of the 45th Annual Southeast Regional Conference*, ser. ACM-SE 45. New York, NY, USA: Association for Computing Machinery, 2007, p. 321–326. [Online]. Available: https://doi.org/10.1145/1233341.1233399

[7] Q. D. La, T. Q. S. Quek, J. Lee, S. Jin, and H. Zhu, "Deceptive attack and defense game in honeypot-enabled networks for the internet of things," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 1025–1035, Dec 2016.

[8] E. Vasilomanolakis, S. Karuppayah, M. Fischer, M. Mühlhäuser, M. Plasoianu, L. Pandikow, and W. Pfeiffer, "This network is infected: Hostage - a low-interaction honeypot for mobile devices," in *Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, ser. SPSM '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 43–48. [Online]. Available: https://doi.org/10.1145/2516760.2516763

[9] D. Tools, "Web honeypot," 2010. [Online]. Available: https://github.com/DinoTools/dionaea/

[10] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, and M. Fischer, "Hostage: a mobile honeypot for collaborative defense," in *Proceedings of the 7th International Conference on security of information and networks*, ser. SIN '14, vol. 2014-. ACM, 2014, pp. 330–333.

[11] A. Vetterl and R. Clayton, "Bitter harvest: Systematically fingerprinting low- and medium-interaction honeypots at internet scale," in *12th USENIX Workshop on Offensive Technologies (WOOT 18)*. Baltimore, MD: USENIX Association, Aug. 2018. [Online]. Available: https://www.usenix.org/conference/woot18/presentation/vetterl

[12] J. D. Guarnizo, A. Tambe, S. S. Bhunia, M. Ochoa, N. O. Tippenhauer, A. Shabtai, and Y. Elovici, "Siphon: Towards scalable high-interaction physical honeypots," in *Proceedings of the 3rd ACM Workshop on Cyber-Physical System Security*, ser. CPSS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 57–68. [Online]. Available: https://doi.org/10.1145/3055186.3055192

[13] S. Srinivasa, J. M. Pedersen, and E. Vasilomanolakis, "Gotta catch 'em all: a multistage framework for honeypot fingerprinting," 2021.

[14] A. Čenys, D. Rainys, L. Radvilavicius, and N. Goranin, "Database level honeytoken modules for active dbms protection," in *Advances in Information Systems Development*, A. G. Nilsson, R. Gustas, W. Wojtkowski, W. G. Wojtkowski, S. Wrycza, and J. Zupančič, Eds. Boston, MA: Springer US, 2006, pp. 449–457.

[15] Acrobat, "Acrobat api reference," August 2021. [Online]. Available: https://opensource.adobe.com/dc-acrobat-sdk-docs/acrobatsdk/html2015/Acro12_MasterBook/API_References_SectionPage/API_References/Acrobat_API_Reference/AV_Layer/Weblink.html

[16] E. Aguirre-Anaya, G. Gallegos-García, N. S. Luna, and L. A. V. Vargas, "A new procedure to detect low interaction honeypots," *International Journal of Electrical and Computer Engineering*, vol. 4, pp. 848–857, 2014.

[17] R. N. Dahbul, C. Lim, and J. Purnama, "Enhancing honeypot deception capability through network service fingerprinting," *Journal of Physics: Conference Series*, vol. 801, p. 012057, jan 2017. [Online]. Available: https://doi.org/10.1088/1742-6596/801/1/012057

[18] X. Fu, W. Yu, D. Cheng, X. Tan, K. Streff, and S. Graham, "On recognizing virtual honeypots and countermeasures," in *2006 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing*, Sep. 2006, pp. 211–218.

[19] T. Holz and F. Raynal, "Detecting honeypots and other suspicious environments," in *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*, June 2005, pp. 29–36.

[20] S. Mukkamala, K. Yendrapalli, R. Basnet, M. K. Shankarapani, and A. H. Sung, "Detection of virtual environments and low interaction honeypots," in *2007 IEEE SMC Information Assurance and Security Workshop*, June 2007, pp. 92–98.

[21] D. Fraunholz, S. D. Antón, C. Lipps, D. Reti, D. Krohmer, F. Pohl, M. Tammen, and H. D. Schotten, "Demystifying deception technology: A survey," *CoRR*, vol. abs/1804.06196, 2018. [Online]. Available: http://arxiv.org/abs/1804.06196

[22] X. Han, N. Kheir, and D. Balzarotti, "Deception techniques in computer security: A research perspective," *ACM Comput. Surv.*, vol. 51, no. 4, jul 2018. [Online]. Available: https://doi.org/10.1145/3214305

[23] L. Zhang and V. L. Thing, "Three decades of deception techniques in active cyber defense-retrospect and outlook," *Computers & Security*, vol. 106, p. 102288, 2021. [Online]. Available: https://arxiv.org/abs/2104.03594

[24] S. Srinivasa, J. M. Pedersen, and E. Vasilomanolakis, "Towards systematic honeytoken fingerprinting," in *13th International Conference on Security of Information and Networks*, ser. SIN 2020. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3433174.3433599

[25] C. D. Faveri and A. Moreira, "Visual modeling of cyber deception," in *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2018, pp. 205–209.

[26] T. A. Research, "Canarytokens." [Online]. Available: https://github.com/thinkst/canarytokens

[27] A. Gungor, "Pdf forensic analysis and xmp metadata streams," 2017. [Online]. Available: https://www.meridiandiscovery.com/articles/pdf-forensic-analysis-xmp-metadata/

Paradigms in Cyber Deception

Paper D

RIoTPot: a modular hybrid-interaction IoT/OT honeypot

Shreyas Srinivasa, Jens Myrup Pedersen, Emmanouil Vasilomanolakis

The layout has been revised.

Abstract

Honeypots are often used as a proactive attack detection mechanism and as a source of threat intelligence data. However, many honeypots are poorly maintained and cumbersome to extend. Moreover, low-interaction honeypots are prone to fingerprinting attacks due to their limited emulation capabilities. Nonetheless, low-interaction honeypots are essential for environments with limited resources. In this paper, we introduce RIoTPot, a modular and hybrid-interaction honeypot for Internet-of-Things (IoT) and Operational Technology (OT) protocols mainly used in Industrial Control System environments. RIoTPot's modularity comes as a result of plug-n-play container services while its hybrid-interaction capability enables users to switch between low- and high-interaction modes. We deploy RIoTPot on the Internet, receive a large amount of attacks and discuss the results received on both low- and high-interaction modes.

1 Introduction

Honeypots are deceptive systems that simulate a seemingly vulnerable system to gather attacks. Over the years, many honeypot solutions have been proposed that are commonly classified to low-, medium- and high-interaction based on the level of interaction they offer to the adversary [1, 2]. Low- and medium-interaction honeypots, due to their limited emulation capabilities, are prone to honeypot fingerprinting that may limit their scope [3]. Honeypot fingerprinting refers to adversarial methods that allow for the identification of the honeypot nature of a system. Nevertheless, these two classes of honeypots are the most commonly deployed ones. Other common issues with honeypots include the lack of flexibility in extending/adapting them, the absence of support, and limited documentation.

Despite the aforementioned limitations, honeypots are an excellent defensive toolkit, especially with regard to the increasing number of IoT and OT attacks. With such protocols being consistently attacked, in both consumer [4] and commercial environments [5], deception mechanisms like honeypots offer an early warning system and a method to analyse adversaries' techniques [6–8].

Traditional honeypot simulations may run on virtualized environments like VMs, virtual containers (LXC), or even language-based virtual environments. Kedrowitsch et al. made a first effort to propose the use of containers for honeypots [9]. The authors propose the usage of Linux containers as a platform to develop honeypots and compliment their proposal by comparing the detection methods of popular virtualization platforms against containers. Kedrowitsch et al. conclude that limitations exist in the use of either containers or virtual machines as a honeypot platform. A much recent proposal by Reti et al. introduces the use of container-based deception for honeypots [10]. The authors investigate the possibilities of container-based honeypots and introduce the

concept of simulating container-escapes (fake network pivoting outside a container) as a deception technique. Both approaches suggest the use of container systems to achieve ease of deployment. Moreover, many open-source honeypots offer the possibility of a containerized deployment for ease of installation. Nevertheless, besides the aforesaid academic work there are not many actual honeypot implementations that make use of containers. Furthermore, all existing honeypots have a binary interaction level: they are either low-, medium-, or high-interaction [1].

In this paper, we present RIoTPot*, a honeypot that: *i.)* breaks the traditional binary interaction paradigm, *ii.)* focuses on IoT and OT protocols, and *iii.)* is designed with a modular-by-design architecture. First, the hybrid-interaction level of RIoTPot aims at providing defenders flexibility by giving them the ability to utilize the appropriate interaction level based on their needs and capabilities. For instance, low constrained environments scale better with low interaction components while high interaction comes handy when deeper analysis of attack is required. Second, RIoTPot supports many IoT and OT protocols (i.e., Telnet, SSH, CoAP, Modbus, MQTT), with more to be implemented in the immediate future. At the moment, there are only a few real world honeypot implementations that focus on IoT [8, 11] and even fewer for OT [7, 12]. Lastly, the modularity of the honeypot comes from its architecture; each functionality of the honeypot is a plug-n-play component that can be edited, activated or deactivated based on the user's preferences.

2   RIoTPot Design

RIoTPot features a modular architecture that facilitates quick integration of new protocol simulation modules. A modular software architecture is a structural approach of building software components as modules by separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality [13]. Figure D.1 shows the high level architecture of RIoTPot. The prominent modules in the architecture are the *RIoTPot core* module, the *packet capture and noise filter* module, the *low-interaction* modules, the *high-interaction* modules, and the *attack database.*

The *RIoTPot core* consists of the required components for the configuration, administration, and orchestration of the honeypot. In particular, the core module provides RIoTPot with all the required parameters at startup. This includes user preferences for specific protocols, profile simulation, and the desired interaction level. In addition, the core is responsible for the network management for the high-interaction protocol services simulated on containers. The received attack traffic is forwarded to the respective container that hosts the protocol on which the attack was targeted. Furthermore, the core also facilitates the communication between itself and the containers, if hosted on a

---

*https://github.com/aau-network-security/riotpot
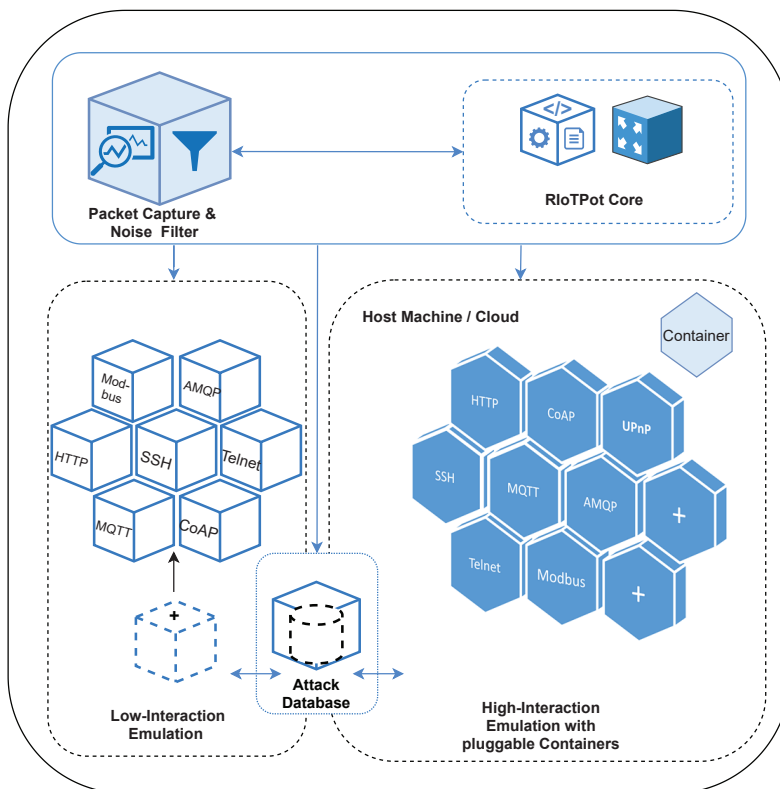
Fig. D.1: High level architecture of RIoTPot

cloud environment.

For the *Packet capture and noise filter* module the attack capture component is responsible for storing the attack packets as *pcap* files, using *tcpdump*, which can be used for detailed analysis (e.g., deep packet inspection). The noise filter component filters out the traffic received from Internet-wide scanners like Shodan [14] and Censys [15]. This helps the honeypot administrator to concentrate on attacks that matter by removing the noise traffic generated by such services.

The *low-interaction mode* is achieved through independent packages, with each package simulating a specific protocol. RIoTPot is implemented in Go language [16] and facilitates the development of a modular architecture through packages. The packages act as plug-ins that can be added to the honeypot to extend the protocols simulated. For example, the *fakeshell* package emulates a system shell that can be leveraged by the SSH and the Telnet packages. The *fakeshell* package can be extended to include

emulation of specific commands. Furthermore, RIoTPot provides a template that can be used for integration of additional protocols. The *high-interaction mode* is achieved by emulating the protocols as services in container images. Hence, since a container implements the full protocol the honeypot provides the attacker with high interaction capabilities. The containers act as high-interaction modules that offer a full implementation of a protocol. Additional protocol services can be added by integrating containers with the desired protocol services. The hybrid-interaction mode further allows the user to emulate selective protocols on low or high-interaction levels. For example, the user can choose to have SSH in low-interaction mode and MQTT in high-interaction mode.

The *attack database* stores all the attack traffic received on the honeypot. The database is setup as an independent module to ensure data availability even if a honeypot module is down (e.g., due to a crash or DDoS attack). The database is accessible from the low-interaction and high-interaction modules for attack storage.

To sum up, the design of RIoTPot facilitates modularity through packages and containers as plugins. Furthermore, the modular architecture assists the hybrid-interaction model of RIoTPot.

3 Preliminary Results

The honeypot was deployed in both low and high interaction modes on two hosts in our lab. The hosts were assigned a public IP each, under an unfiltered network. We define an attack as any interaction with the honeypot as there is no production value whatsoever. However, we differentiate incoming traffic from well-known crawlers (e.g. Shodan). The attacks on the honeypots were recorded for a period of one week. In the low-interaction variant, the protocols SSH, Telnet, HTTP, MQTT, CoAP and Modbus were simulated through the plug-in packages, while the high-interaction variant simulated the MQTT protocol in a container. In addition to recording the attacks in the database, the hosts also had the *tcpdump* service running in the background to capture the attack packets for comprehensive analysis. A total of $7,587$ attacks were observed across all the protocols simulated by RIoTPot.

Figure D.2 shows the number of unique attacks received per protocol for a period of one week. MQTT-HI indicates the high-interaction mode of the MQTT protocol. We observe a trend in the number of attacks for all protocols. Furthermore, the number of attacks on the MQTT protocol in the high-interaction mode is higher in comparison to the low-interaction mode. Moreover, we observe recurring sessions from same suspicious actors on the high-interaction mode, that included topic creation, subscription and deletion, and modification of existing messages in topics which have not been observed on the low-interaction mode.

Figure D.3 depicts the percentage of attacks from Internet-scanning engines (e.g., Shodan, Censys, Project Sonar [17], and ShadowServer [18]) in comparison to the attacks from suspicious sources. We observe an average of 25% of the total traffic originating
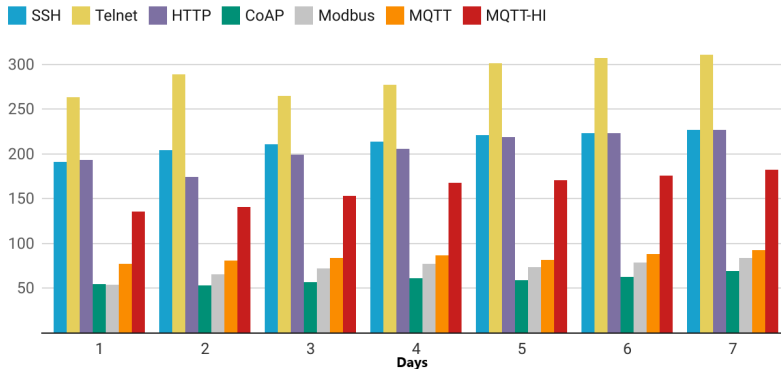
Fig. D.2: Number of attacks on protocols per day
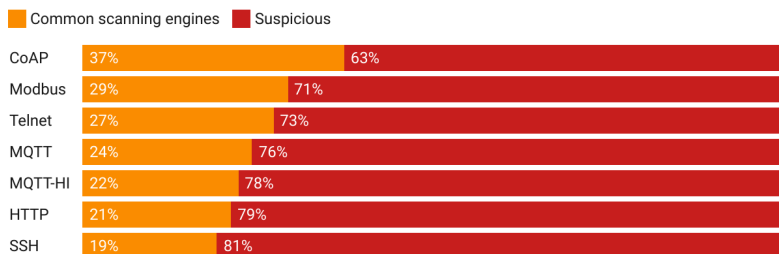
Fig. D.3: Attack noise classification in percentage

from 19 common scanning engines[†]. Filtering out such traffic reduces noise and alert data fatigue for the administrators.

4   Conclusion

In this paper, we introduce RIoTPot, a honeypot that features a hybrid-interaction model with a modular design for IoT and OT protocols. RIoTPot addresses the issue of limited interaction and flexibility, in addition to ease of deployment. Our preliminary results suggest that the honeypot is attractive to adversaries and is able to distinguish between suspicious traffic (traffic originating from attackers) and common scanning engines (traffic likely coming from Shodan-like systems). As future work, we aim to extend RIoTPot to support more IoT and OT protocols like UPnP, AMQP, XMPP, S7, DNP3, Fieldbus and Profibus. Furthermore, we intend to integrate threat intelligence

---

[†]For a complete list of the supported scanning engines see: https://github.com/aau-network-security/riotpot#12-Noise-Filter

reporting through STIX to facilitate structured sharing of threat data [19]. Finally, we plan to perform a more extensive evaluation of RIoTPot with an emphasis on ICS environments.

References

[1] L. Zhang and V. Thing, "Three decades of deception techniques in active cyber defense - retrospect and outlook," *Computers & Security*, vol. 106, p. 102288, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404821001127

[2] M. L. Bringer, C. A. Chelmecki, and H. Fujinoki, "A survey: Recent advances and future trends in honeypot research," *International Journal of Computer Network and Information Security*, vol. 4, no. 10, p. 63, 2012.

[3] A. Vetterl and R. Clayton, "Bitter harvest: Systematically fingerprinting low- and medium-interaction honeypots at internet scale," in *12th USENIX Workshop on Offensive Technologies (WOOT 18)*. Baltimore, MD: USENIX Association, Aug. 2018. [Online]. Available: https://www.usenix.org/conference/woot18/presentation/vetterl

[4] A. Mangino, M. S. Pour, and E. Bou-Harb, "Internet-scale insecurity of consumer internet of things: An empirical measurements perspective," *ACM Transactions on Management Information Systems*, vol. 11, no. 4, Oct. 2020. [Online]. Available: https://doi.org/10.1145/3394504

[5] X. Jiang, M. Lora, and S. Chattopadhyay, "An experimental analysis of security vulnerabilities in industrial iot devices," *ACM Trans. Internet Technol.*, vol. 20, no. 2, May 2020. [Online]. Available: https://doi.org/10.1145/3379542

[6] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, and M. Fischer, "Hostage: A mobile honeypot for collaborative defense," in *Proceedings of the 7th International Conference on Security of Information and Networks*, ser. SIN '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 330–333. [Online]. Available: https://doi.org/10.1145/2659651.2659663

[7] L. Rist, J. Vestergaard, D. Haslinger, A. Pasquale, and J. Smith, "Conpot ics/scada honeypot," *Honeynet Project (conpot. org)*, 2013.

[8] M. Oosterhof, "Cowrie ssh/telnet honeypot," 2016. [Online]. Available: :https://github.com/micheloosterhof/cowrie

[9] A. Kedrowitsch, D. D. Yao, G. Wang, and K. Cameron, "A first look: Using linux containers for deceptive honeypots," in *Proceedings of the 2017 Workshop on Automated Decision Making for Active Cyber Defense*, ser. SafeConfig '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 15–22. [Online]. Available: https://doi.org/10.1145/3140368.3140371

[10] D. Reti and N. Becker, "Escape the fake: Introducing simulated container-escapes for honeypots," 2021.

[11] E. Vasilomanolakis, S. Karuppayah, M. Fischer, M. Mühlhäuser, M. Plasoianu, L. Pandikow, and W. Pfeiffer, "This network is infected: Hostage-a low-interaction honeypot for mobile devices," in *Proceedings of the Third ACM workshop on Security and privacy in smartphones & mobile devices*, 2013, pp. 43–48.

[12] E. Vasilomanolakis, S. Srinivasa, and M. Mühlhäuser, "Did you really hack a nuclear power plant? an industrial control mobile honeypot," in *2015 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2015, pp. 729–730.

[13] M. Mohammed, M. Elish, and A. Qusef, "Empirical insight into the context of design patterns: Modularity analysis," in *2016 7th International Conference on Computer Science and Information Technology (CSIT)*, 2016, pp. 1–6.

[14] SHODAN, "Shodan," 2021. [Online]. Available: https://www.shodan.io/

[15] Censys, "Censys search," 2021. [Online]. Available: https://censys.io/

[16] Golang, "Go language," 2021. [Online]. Available: https://golang.org/

[17] R. Research, "Project sonar," 2021. [Online]. Available: https://www.rapid7.com/research/project-sonar/

[18] ShadowServer.org, "Shadowserver.org," 2021. [Online]. Available: https://www.shadowserver.org/

[19] S. Barnum, "Standardizing cyber threat intelligence information with the structured threat information expression (stix)," *Mitre Corporation*, vol. 11, pp. 1–22, 2012.

Paper E

Interaction matters: a comprehensive analysis and a dataset of hybrid IoT/OT honeypots

Shreyas Srinivasa, Jens Myrup Pedersen, Emmanouil Vasilomanolakis

The layout has been revised.

Abstract

The Internet of things (IoT) and critical infrastructure utilizing operational technology (OT) protocols are nowadays a common attack target and/or attack surface used to further propagate malicious actions. Deception techniques such as honeypots have been proposed for both IoT and OT but they either lack an extensive evaluation or are subject to fingerprinting attacks. In this paper, we extend and evaluate RIoTPot, a hybrid-interaction honeypot, by exposing it to attacks on the Internet and perform a longitudinal study with multiple evaluation parameters for three months. Furthermore, we publish the aforementioned study in the form of a dataset that is available to researchers upon request. We leverage RIoTPot's hybrid-interaction model to deploy it in three interaction variants with six protocols deployed on both cloud and self-hosted infrastructure to study and compare the attacks gathered. At a glance, we receive 10.87 million attack events originating from 22,518 unique IP addresses that involve brute-force, poisoning, multistage and other attacks. Moreover, we fingerprint the attacker IP addresses to identify the type of devices who participate in the attacks. Lastly, our results indicate that the honeypot interaction levels have an important role in attracting specific attacks and scanning probes.

1 Introduction

The number of cyber attacks targeting mission-critical infrastructure has increased steadily [1]. Recent attacks on industrial systems like the US Colonial Pipeline [2] and the Florida Water Treatment plant [3] have shown the impact caused on both the public and the government. Mission-critical systems in Operational Technology (OT) rely on sensors and connected devices to automate industrial control processes. However, a study of recent attacks reveals a lack of security on these devices [4]. Furthermore, the vast adoption of IoT devices in both consumer and industrial applications has increased the attack space. Research shows a large number of vulnerable, misconfigured IoT devices exposed to the Internet and sophisticated malware that can exploit them [5, 6]. Moreover, the ENISA Threat Landscape Report 2020 states that malware continues to be the most challenging attack vector and up to 230,000 variants are distributed daily [7].

Honeypots are deception systems that act as a trapping mechanism for attackers. They have low false positives as there is no justification to communicate with a honeypot, and hence, all communication can be considered suspicious. Honeypots are classified into low, medium, and high interaction based on the interaction capabilities they offer to the attackers. Over the years, many honeypots have been proposed for various protocols or device profiles. Well known examples include Cowrie [8], Conpot [9], HosTaGe [10], Glastopf [11], Dionaea [12], and T-Pot [13].

Honeypots come with some limitations as a result of either the lack of interaction, their static responses, or poor maintenance. Honeypot fingerprinting is the process of determining if the system in communication is a honeypot through probing mechanisms. Successful fingerprinting attacks can undermine the value of honeypots as their identity is exposed. A number of techniques for fingerprinting honeypots have been proposed in recent research [14–16]. Moreover, many open-source honeypot projects are abandoned or depend on libraries that are not in active maintenance; this leads to a lack of extensibility and reduced scope [14].

In this paper, we extend RIoTPot, a modular and hybrid-interaction honeypot that addresses the gap between extensibility and interaction to simulate application-layer protocols used in IT, IoT, and OT environments [17]. The contributions of this paper are as follows.

- We extend RIoTPot and provide the source code for IoT and OT device profiles that emulate the Telnet, SSH, MQTT, Modbus, CoAP, and HTTP protocols.

- We perform a longitudinal study of RIoTPot imposing many evaluation parameters based on interaction-level, simulation environment, deployment infrastructure, and geolocation. We report our findings and discuss the impact of the evaluation parameters on the operation of honeypots. In addition, we examine how RIoTPot performs compared to another popular state of the art honeypot.

- We offer the attack dataset to the research community.

The rest of the paper is structured as follows. Section 2 provides an overview of the related work in the use of honeypots for analyzing attack trends. We discuss RIoTPot, the extensions we implemented and the offered dataset in Section 3. In Section 4, we outline the methodology of our study and the experimental setup and evaluate our approach in Section 5. We discuss our findings in Section 6 and conclude in Section 7.

2   Related Work

Honeypots are commonly classified into low, medium, and high-interaction based on the interaction level they offer to attackers. Low-interaction honeypots offer limited simulations of a protocol or service and are easy to manage. Medium-interaction offer an extended interaction level to low-interaction and involve emulating a device that constitutes multiple protocols. High-interaction honeypots are real (or virtual) systems/devices with customized logging, limited egress traffic rules, and ephemeral configurations to prevent misuse. At a glance, various well-known low/medium interaction honeypots include: Cowrie [8], Conpot [9], HosTaGe [10], Glastopf [11], Dionaea [12] and TPot [13]. In addition, there are some high-interaction honeypots proposed such as Honware [18], Siphon [19], and Sarracenia [20]. All of the aforementioned honeypots operate at low

or high interaction modes thus offering a binary interaction capability. Moreover, they vary a lot on how much extensibility they offer for integrating additional protocols or targeting specific environments.

Another factor that is important to be taken into account is fingerprinting; adversaries may be able to detect if a target system is a honeypot using techniques that leverage minimal data obtained from the target system. On the one hand, recent research reveals that while low/medium-interaction honeypots are easy to manage and are minimal in risk, they are more vulnerable to fingerprinting attacks [14, 15]. On the other hand, high-interaction honeypots risk a compromised environment that can be leveraged for malicious activities entailing higher maintenance.

Litchfield et al. propose HoneyPhy to address the issues of limited simulation of honeypots that simulate cyber-physical environments [21]. The authors argue that the honeypots fail to emulate the actual behavior of cyber-physical systems which can lead to fingerprinting. The authors extend their work by proposing HoneyBot; a hybrid-interaction honeypot designed explicitly for networked robotic systems [22] that is capable of switching interaction based on the attack requests. However, HoneyBot is limited to a specific environment and cannot be extended for diverse operations.

The evaluation of honeypots in research studies that aim at presenting an overview of the attack landscape is not uncommon [23–25]. Dang et al. performed a longitudinal study of 12 months in a combined approach with both hardware and software honeypots [26]. The experiment involved the deployment of four hardware and 108 software IoT honeypots in gathering attacks on the IoT landscape. The software/virtual honeypots are deployed on eight public cloud providers and various geolocations. The authors present an overview of the attacks received on the honeypots and the implications. Minn et al. propose *IoTPOT*, a honeypot that simulates Telnet services from various IoT devices, to study the attack trends on the Telnet protocol [27]. The *IoTPOT* honeypot consists of a low-interaction front-end connected with a high-interaction back-end called *IoTBOX* that simulates the Telnet service from various IoT device profiles. The authors deployed the honeypot for 39 days and collected 43 distinct malware samples.

Srinivasa et al. find a large number of misconfigured IoT devices on the Internet and deploy six open-source IoT honeypots to study attack trends [5]. Tabari et al. present a multi-faceted IoT-honeypot ecosystem with extendable sophistication by observing real-world attacks [28]. The authors develop a honeypot for IoT cameras to observe the attack landscape around them. Furthermore, the authors propose *ProxyPot*, a honeypot proxy that sits between IoT devices and external networks to observe inbound and outbound traffic. The IoT camera honeypots were deployed for two years and an increase in the number of attacks was observed.

Vetterl et al. propose *Honware*, a high-interaction, virtual honeypot framework that can emulate a wide range of IoT device firmware without the hardware [18]. The authors evaluated the honeypot by deploying four device profiles of ADSL modems and found various attacks targeting vulnerabilities specific to emulated devices. Further-

more, Guarnizo et al. present *Siphon*, a scalable high-interaction honeypot architecture which utilizes IoT devices that are physically deployed at a geographical location and connected to the Internet for simulation [19]. The authors deploy 85 honeypot instances, in various locations, that utilize five physical cameras, an NVR, and an IP printer. An analysis of the attacks revealed that honeypots in certain cities received more attack traffic compared to others. Valeros et al. provide *Hornet 40*, a network dataset of geographically placed honeypots to study the impact of geolocation [29]. The data consists of 118 features, including 480 bytes of content for each flow. The dataset does not contain interactive attack traffic as only passive honeypots were used in the study. The attack data is collected from honeypots deployed at eight locations and contain 4.7 million network flows collected over a period of 40 days.

Barron et al. performed a four-month study that involved 102 medium-interaction honeypots [30]. In addition to deploying the honeypots at multiple locations, the authors experiment with parameters like the difficulty in the break-in and file generation. They observe how the differences in these parameters caused deviance in the attackers' behavior. In addition, the authors leak the access information of the honeypots on hacking forums and paste sites to monitor the attackers' actions. The authors find that the differences in the parameters introduced affected the human-based attackers and list key takeaways from the experiment. We consider this work the closest to our approach, where the authors introduce specific parameters in the study and how they influence the attacker.

Appendix Table E.6 provides an overview of the qualitative comparison of honeypots proposed in related work. The honeypots are compared based on their source-code availability, supported protocols, interaction level, operational environments and known fingerprinting techniques. Most of the proposed honeypots are available as open source and support multiple protocols. However, we observe that there are no high-interaction honeypots are available as open source. Furthermore, we observe that for open source honeypots, certain fingerprinting methods have been proposed. Most of the honeypots are designed to be deployed as virtual environments except IoTPot, which runs on hardware. While most of the honeypots operate at binary interaction levels, i.e. low or medium or high, RIoTPot is capable of operating in either low, high or hybrid interaction levels.

The related work listed in Table E.1 summarizes research that involves the deployment of honeypots to study different attack surfaces. Nevertheless, none of the studies compare attacks by deploying honeypots with varied interaction levels and operating environments for multiple protocols. Furthermore, there are no public datasets that offer diverse data based on interaction levels and protocols. We aim to address this gap by deploying multiple honeypot instances with varied interaction levels and operating environments that are geographically distributed. In addition, we analyze and compare the attacks received on multiple interaction levels simulated on different IoT and OT application protocols.

Study	Interaction level	Study period	Geographically distributed	Deployment
Honeycloud [26] (2019)	Medium	12 months	Yes	hardware, cloud
IoTPOT [27](2015)	Low	39 days	No	physical
Open for hire [5] (2021)	Low, Medium	1 month	No	physical
Muti-faceted Honeypot [28](2020)	Low	2 years	No	physical
Honware [18] (2019)	High	14 days	No	physical
Siphon [19](2017)	High	2 months	Yes	physical, cloud
Hornet 40 [29](2021)	Passive	40 days	Yes	cloud
Picky Attackers [30] (2017)	Medium	4 months	Yes	physical, cloud
RIoTPot (2022)	**Low, High, Hybrid**	**3 months**	**Yes**	**physical, cloud**

Table E.1: Overview of related work & evaluation parameters

3 Extending RIoTPot

RIoTPot aims to address the limitations of extensibility, interaction, and operational environments [17]. It follows a modular architecture and offers hybrid interaction levels. We extend RIoTPot* to adapt to this longitudinal study along with various enhancements (see also Figure D.1). RIoTPot offers multiple features such as extensibility, mode of operations, and compatibility with diverse deployment environments.

The motivation for RIoTPot is to offer administrators with the ease of deploying honeypots in their infrastructure, especially high-interaction honeypots in the form of containers. RIoTPot is open-source and can run on virtual infrastructure, unlike other high-interaction honeypots. Furthermore, RIoTPot offers administrators with the flexibility of changing the interaction level based on the resources. RIoTPot is able to start being deployed as low-interaction and switch to high-interaction on the fly. RIoTPot is designed to focus on modularity to facilitate the integration of additional protocols and maintenance. Many open-source projects are often abandoned due to the lack of extensibility. Modular architecture addresses this gap by providing extensibility in the form of modules. For low-interaction mode, honeypot administrators can use the default template for adding new simulations and easily integrate them with the startup configuration. Similarly, for the high-interaction mode, the path for the container image is provided for simulation. Through a modular implementation, administrators can add any relevant scenarios to the RIoTPot simulation portfolio.

Many honeypot projects face the threat of fingerprinting (see Section 2). Fingerprinting enables an adversary to detect honeypots based on elementary information obtained through crafted request probes. This is particularly harmful for low and medium interaction honeypots that are often utilizing specific libraries or hard-coded responses

---

*https://github.com/aau-network-security/riotpot

that can be fingerprinted. Nevertheless, such honeypots are of low maintenance and risk and thus an attractive deception mechanism. The hybrid interaction feature of RIoTPot provides honeypot administrators with a choice of operating the emulation in either low, high, or hybrid interaction. Hybrid allows some protocols to run in low and some on high interaction. This allows defenders to lure attackers into specific protocols in a breadcrumb-like approach. For instance, the administrators can run a protocol that interests them on high interaction and some other (that would be expected by the attacker) in low. The attacker would then eventually spend most of their time and effort on the high interaction protocol.

While a number of honeypot projects are developed focusing on containerized deployment, RIoTPot also offers a hybrid deployment scenario where administrators can choose to run the high-interaction containers on remote hosts or local infrastructure. This feature is beneficial in resource-constrained environments. For example, RIoTPot can be deployed on Raspberry Pi, and the high-interaction containers can be deployed in a cloud environment. Furthermore, if the simulation profile requires specific hardware (e.g., sensors), RIoTPot simulation containers can be deployed on the supporting infrastructure.

3.1   RIoTPot extended architecture

The architecture of RIoTPot follows modularity that enables quick integration of protocols and emulation modules [17]. Modular software architecture is a structural approach to building software components as modules by separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality [31]. Figure E.1 shows the extended architecture adapted from RIoTPot [17]. In addition to the quick integration of additional modules, the architecture facilitates the integration of extensions to support extended analysis or configuration. The prominent modules in the architecture are the *RIoTPot core* module, the *low-interaction* modules, the *high-interaction* modules, and the *attack database*. The *packet capture and noise filter* modules serve as extensions to support further analysis of the attack data.

3.2   Extended components

RIoTPot Core

The *RIoTPot core* consists of the essential components that enable the configuration, administration and orchestration services. The core module facilitates the administrators to configure the parameters like specifying the protocols for emulation, the desired interaction-level and the path for the loading the images in case of high interaction mode. The startup configuration allows users to choose the desired protocols and the interaction-level. In addition to the configuration, the core facilitates the network management between the containers in the high interaction mode. The traffic is forwarded
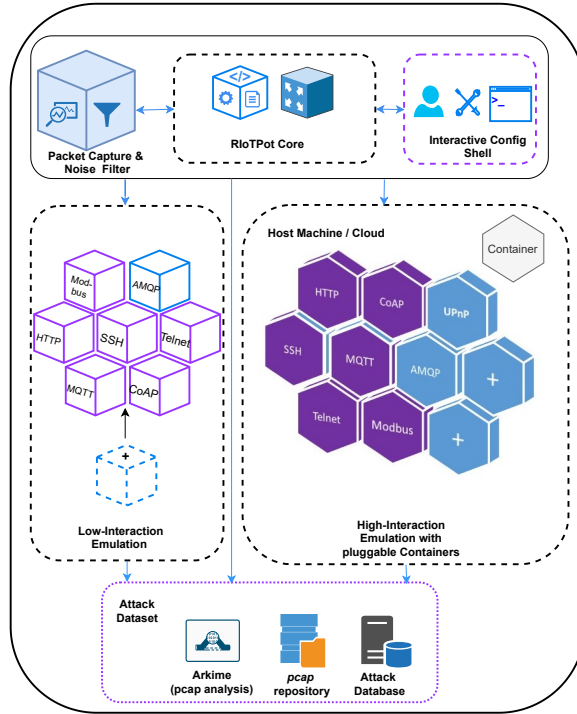
Fig. E.1: RIoTPot architecture adapted from [17], purple components suggest novel or enhanced parts

to the containers based on the simulated protocols on which the attacks are targeted. Furthermore, the core is responsible for communication with remote containers, in case of a cloud-based deployment. We extend the configuration module of the core by enhancing the static file-based configuration to a shell-based interactive configuration. The shell-based configuration provides interactive prompts where the desired startup configuration can be chosen by the administrator. The prompts include selection of protocols for emulation, choosing the interaction levels as operation modes, providing the parameters for remote database, *pcap* repository and the container images in case of operation in high interaction mode.

Low-interaction modules

The *low-interaction mode* is enabled by the implementation of packages that simulate individual protocols. RIoTPot is implemented in Go language [32] that enables development of independent packages. The modular architecture of RIoTPot is achieved

by developing the protocols as independent packages that can further be integrated as plug-ins. For example, the Telnet and the SSH protocols use the *fakeshell* package that emulates a system shell and responds to a list of commands. The *fakeshell* package can be extended to include more commands. A default template is provided with RIoTPot that can be leveraged for the integration of additional protocols. By default, RIoTPot supports emulation of seven protocols that include Telnet, SSH, HTTP, MQTT, AMQP, Modbus and CoAP. We extend RIoTPot by enhancing the Telnet, SSH, HTTP, MQTT, Modbus and CoAP modules for adapting to our study. The changes include extending the shell emulation capabilities for the *fakeshell* module and enhancing the emulation capabilities for the Modbus, MQTT and CoAP protocols.

High-interaction modules

RIoTPot operates in *high-interaction mode* by emulating protocols as services running on container images. For each configured protocol, the administrator provides a relevant image that will be deployed on a container for emulation. As the protocols operate as full services on the containers, they act as high interaction modules that provide a full implementation of a protocol and thereby provides the attackers with high interaction capabilities. Additional protocols can be integrated by specifying the protocol and the image path in the startup configuration. The high-interaction modules have been extended in this work by leveraging container images provided through the Docker Hub repository [33]. An advantage of using Docker Hub is that it employs a verification procedure for most of its images (e.g., verifying that the Apache Foundation is the publisher of the httpd image and BusyBox for the busybox image). We use the Busybox [34], OpenSSH [35], HTTPD [36], Modbus-Server [37], Eclipse-Mosquitto [38] and CoAP-Gateway [39] images for high-interaction of protocols.

Hybrid interaction

The hybrid interaction mode allows to choose the desired interaction level for selective protocols. Through the hybrid interaction mode, a specific protocol like SSH can run in low interaction mode while another (e.g., HTTP) can run in high interaction mode. This facilitates administrators to set up device profiles that constitute a collection of protocols with less resource requirements and can choose the hybrid operation mode through the interactive shell configuration prompts during startup.

Noise filter and packet capture

RIoTPot has two default extensions - the attack capture component and the noise filter. The attack capture component stores all the traffic received on RIoTPot as *pcap* files using *tcpdump* to facilitate comprehensive analysis. Through the attack capture extension, the users can further specify the required rotation levels for the packet capture.

The attack capture component is responsible for storing the attack packets as *pcap* files, using *tcpdump*, which can be used for detailed analysis (e.g., deep packet inspection). The noise filter component filters out the traffic received from Internet-wide scanners like Shodan [40] and Censys [41]. The component can filter traffic from 19 Internet scanning services. The attack sources are labeled accordingly in the attack database. This helps the administrator concentrate on attacks that matter by removing the noise generated by such services.

Attack dataset

The traffic received on RIoTPot is stored in the *attack database*. It is provisioned as an independent container to ensure no disruption in logging in the event of a crash or failure. The attacks received on both low and high interaction modes are stored in the database. The database instance is accessible from the low and high interaction containers. To facilitate this longitudinal study we extend RIoTPot to log all the attacks to a remote database in the cloud that ensures scalability and simplified backup process. We further enhance RIoTPot by integrating *Arkime*, an open-source indexed packet capture and searching tool [42]. We leverage Arkime to search through the *pcap* files generated from the honeypot deployments. Arkime imports the *pcap* files from the *pcap* repository and stores them in its back-end (Elasticsearch) for indexed searching capability. Furthermore, Arkime supports querying based on attributes and integration with Virustotal [43] that helps in identification of malicious events and sources.

4   Methodology

Our work aims to capture attacks on the IoT and OT landscape and evaluate RIoT-Pot by leveraging its hybrid-interaction operational feature. Furthermore, we impose six evaluation parameters to observe their influence on our experiment and compare the results. We describe the evaluation parameters and the experimental setup of our longitudinal study in the following sections.

4.1   Evaluation parameters

Interaction levels

As RIoTPot can operate in low, high, and hybrid interaction levels, we study and compare the attacks received on each interaction level. The analysis based on interaction levels will provide an insight into the effectiveness of deception used on each interaction level.

Multiple honeypot instances

By deploying multiple instances of honeypots, we get a better understanding of the observed attacks. For instance, an attack from a specific IP identified on all honeypot instances can construe that the attack source is either an Internet-wide scanning service or part of a reconnaissance process from a bot. On the contrary, unique attack sources identified in specific instances provide insight into distinct attack types and approaches.

Deployment infrastructure

The honeypots' deployment infrastructure plays a significant role in the deceptive layer. While it is common for some application protocols to be open in cloud environments (e.g., Telnet, SSH, MQTT, HTTP), it is peculiar to have protocols like Modbus on the cloud. The cloud infrastructure that includes the containers for simulation, attack database, and the *pcap* file repository are provisioned on the Digital Ocean cloud with configurations to limit the egress traffic. We evaluate RIoTPot in self-hosted lab as well as cloud infrastructure to study the influence of deploying honeypots in different settings.

Geographical distribution

There is research on the impact of deploying honeypots in different geographical locations [44] and we therefore take this into consideration by deploying honeypots on different continents and countries. In particular, our experiments are conducted at four geographical locations, namely: New York City (cloud), Frankfurt (cloud), Singapore (cloud), and Denmark (lab infrastructure) to review region-specific attacks. This way the attack data can be analyzed to discover attacks that are region-specific or potential region-specific malware variants.

Protocol emulation

We study six application protocols, namely: Telnet, SSH, HTTP, MQTT, Modbus, and CoAP. The reason for choosing these protocols is to have a mixed emulation portfolio that includes the most commonly used application protocols in both self-hosted and cloud infrastructure and IoT and ICS environments. The protocols are simulated as both low-interaction in the form of modules and high-interaction in dedicated ephemeral containers. The analysis of attacks provides protocol-specific threats and attack trends resulting from misconfiguration.

Period of study

The evaluation of RIoTPot is performed for three months, both for self-hosted and cloud environments that result in a dataset of a large volume of attack traffic captured

on each RIoTPot instance. The attacks gathered over time provide an overview of the attack trends on each protocol simulation. Furthermore, we run the Conpot honeypot to compare the attacks received from RIoTPot. The study was carried out from December 10, 2021 - to March 10, 2022.

4.2   Experimental setup

We intend to deploy RIoTPot in diverse environments, interaction modes, geographical locations, and simulation environments to get a comprehensive view of the attacks. The experiment was distributed across our lab and the cloud infrastructure to facilitate the evaluation and the parameters. We deploy RIoTPot in tertiary interaction modes - low, high, and hybrid interaction for further evaluation. We describe our lab's experimental setup and cloud infrastructure in the following sections.

Lab setup

The experimental setup in our lab is depicted on the Appendix Figure E.8. RIoTPot was deployed on three hosts R1 (high-interaction), R2 (low-interaction) and R3 (hybrid-interaction). In addition to RIoTPot, the Conpot [9] honeypot is deployed on host C1 (medium-interaction). All four hosts are connected to the Internet and configured with a public IP address on an unfiltered network. However, the hosts are configured with limited egress traffic to avoid misuse of honeypots. The containers spawned by RIoTPot run as ephemeral instances that are re-spawned periodically to avoid infection spread and recover from availability crashes. The attack data from all the hosts are stored as partitioned, individual, and rotated files on a remote file repository to facilitate further analysis. All the attack traffic is stored in the attack database to facilitate querying and analysis. The attack database and the file repository are provisioned on remote systems to avoid disruption in the logging in situations of system failure. Host R3 operates in a hybrid interaction mode where the SSH, MQTT, Modbus, and CoAP operate in high-interaction mode, and Telnet and HTTP operate in low-interaction modes.

Cloud setup

The experimental setup on the cloud infrastructure is shown in the Appendix Figure E.9. Similar to the lab setup, the cloud instances are provisioned on the Digital Ocean as Droplets and has 12 honeypot instances (R4-C4). The 12 honeypot instances are distributed across three geographical locations - New York City, Frankfurt, and Singapore and configured with a public IP address accordingly. Similar to the lab setup, the attack traffic from all the instances is stored as both *Pcap* files and in a database running on dedicated remote systems. The containers are re-spawned periodically and re-provisioned with a static configuration file. The egress traffic from all the containers

is limited for potential misuse of the vulnerable environments. The database is provisioned with an elastic model to support the large volumes of attack traffic collected from the honeypot instances. Digital Ocean droplet monitoring helps in tracking the state of the honeypot instances that helps in identifying any failure situations [45].

Summary

Table E.2 summarizes the experimental setup of the evaluation. To summarize the evaluation parameters of the longitudinal study described in Section 4.1, we deploy RIoTPot in three interaction levels (low, high, hybrid), two deployment environments (lab, cloud), twelve independent hosts per interaction-level, four geographical locations(Denmark(lab) , New York City, Frankfurt, and Singapore), six application protocol emulations (Telnet, SSH, HTTP, MQTT, Modbus, CoAP), comparison with one honeypot in medium interaction (Conpot) and an evaluation period of three months (10Dec,2021-10Mar,2022).

Host	Environment	Geo-Location	Interaction-level	Protocols Emulated
R1	Lab	Denmark	High	Telnet, SSH, HTTP, MQTT, Modbus, CoAP
R2	Lab	Denmark	Low	Telnet, SSH, HTTP, MQTT, Modbus, CoAP
R3	Lab	Denmark	Hybrid	High - SSH, MQTT, Modbus, CoAP Low - Telnet, HTTP
C1	Lab	Denmark	Medium	Telnet, SSH, HTTP, Modbus, S7
R4	Cloud	New York City	High	Telnet, SSH, HTTP, MQTT, Modbus, CoAP
R5	Cloud	New York City	Low	Telnet, SSH, HTTP, MQTT, Modbus, CoAP
R6	Cloud	New York City	Hybrid	High - SSH, MQTT, Modbus, CoAP Low - Telnet, HTTP
C2	Cloud	New York City	Medium	Telnet, SSH, HTTP, Modbus, S7
R7	Cloud	Frankfurt	High	Telnet, SSH, HTTP, MQTT, Modbus, CoAP
R8	Cloud	Frankfurt	Low	Telnet, SSH, HTTP, MQTT, Modbus, CoAP
R9	Cloud	Frankfurt	Hybrid	High - SSH, MQTT, Modbus, CoAP Low - Telnet, HTTP
C3	Cloud	Frankfurt	Medium	Telnet, SSH, HTTP, Modbus, S7
R10	Cloud	Singapore	High	Telnet, SSH, HTTP, MQTT, Modbus, CoAP
R11	Cloud	Singapore	Low	Telnet, SSH, HTTP, MQTT, Modbus, CoAP
R12	Cloud	Singapore	Hybrid	High - SSH, MQTT, Modbus, CoAP Low - Telnet, HTTP
C4	Cloud	Singapore	Medium	Telnet, SSH, HTTP, Modbus, S7

Table E.2: Experimental setup overview

4.3 Dataset

The traffic received on all the honeypot instances from the study are stored in the attack database and as *pcap* files in the *pcap cloud*. In this longitudinal study, we collect data from 12 honeypot instances of RIoTPot and four instances of Conpot over a period of three months. The dataset is a collection of the database dumps and the *pcap* files. The

pcap files capture the ingress and egress traffic from the honeypot instances. The dataset is segregated based on honeypot instance, protocol, geolocation and interaction-levels. The filtering of the scanning-service from the ingress traffic is performed by labeling the traffic in the database. The labeled dataset of the labeled events of scanning-services can be exported from the attack database. Currently, the dataset is checked for 19 scanning-services. The traffic captured on the *pcap* files are "packet-buffered", so that the output is written to *pcap* file at the end of each packet rather than at the end of each line. The administration traffic is excluded from the *pcap* files and the attack database. The *pcap* files are periodically rotated (daily). The dataset will be provided to academic researchers upon request and following a non disclosure agreement[†].

5   Evaluation

To provide a comprehensive overview of the findings during the study, we break down the results based on the evaluation parameters. The findings are discussed in the following sections.

5.1   Interaction levels

We discuss our findings based on interaction-levels in the following sections.

Total Events

Figure E.2 shows the total number of events on all RIoTPot instances based on interaction levels low, high and hybrid. Compared to the low and hybrid interaction, the high-interaction level received higher events. A total of 10.87 million events were received on all instances, of which 32% $(3,487,877)$ were from low, 35% $(3,788,435)$ from high, and the remaining 33% $(3,600,823)$ from hybrid interaction. The total events are inclusive of the probing traffic received from Internet-wide scanning probes (e.g., [41], [40]).

    The Appendix Figure E.10 shows the percentage of events received per day by interaction level. From the outset, we see a rise in the events received on the high-interaction level compared to low and hybrid interaction levels. We see sharp differences in the number of events between December 13-15,2021, and February 13-20,2022. A possible explanation for such uncertainty could be that the hybrid-interaction level involves both low and high interaction levels on the simulated protocols. We discuss the possible reasons for the deviations in Section 6.3.

[†]https://doi.org/10.11583/DTU.21088651
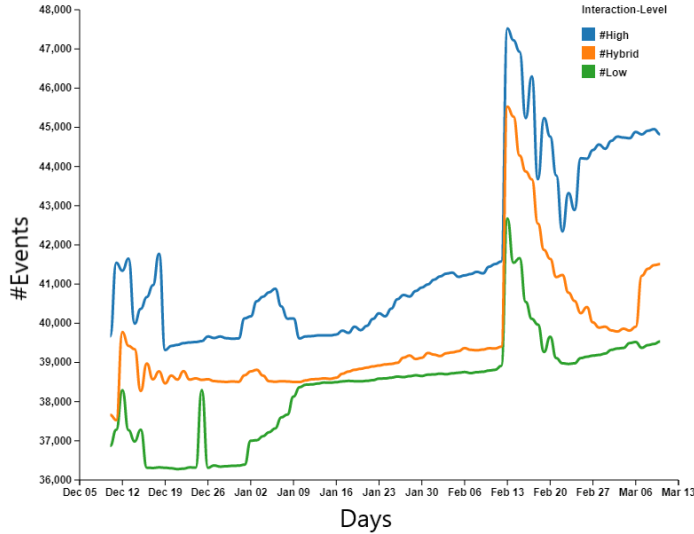
Fig. E.2: Total events by interaction

Event classification

A total of 10.87 million events were received on all the RIoTPot deployments. Table E.3 summarizes the event classification by type and interaction level. Of the total events, 56% of the traffic was identified from Internet scanning-services (e.g., Shodan [40], Censys [41]). We consider the traffic from scanning-services as benign because of the known intent behind their scans. Filtering out benign traffic simplifies the analysis process of focusing on the traffic with malicious intent. A total of 19 unique scanning-services[‡] were identified and labeled by RIoTPot's noise filter, which has a database of benign scanning-services. While it is common to observe recurring traffic from scanning-services like Shodan and Censys, some scans occur multiple times per day while other services follow a different pattern that ranges from days to weeks. Note that we simulate six protocols in RIoTPot and some scans target specific port ranges and some target specific ports with custom requests [46]. Lastly, we did not detect any deviations in the received scanning-services traffic on the basis of the interaction.

The traffic that is not labeled as scanning-services is classified as malicious. The malicious classification includes both suspicious traffic and traffic with clear malicious intent in the requests or payloads. The suspicious traffic includes probing traffic from unknown sources and probable backscatter noise. As honeypots do not have any production value, we consider any communication, excluding the aforementioned scanning

---

[‡] https://github.com/aau-network-security/riotpot#12-Noise-Filter

services, towards them suspicious. 4.8 million events are marked as malicious based on our criteria. Note that the number of malicious events stated here is not unique by attack source. We observe multiple attacks from the same attack source in the traffic volume. Further classification of malicious traffic volume received based-on interaction level is shown in Table E.3. We observe that the high-interaction instances received higher volume than low and hybrid interaction levels.

Interaction-level	Even-type	Count
Low-interaction	Scanning-service	2.02 M
High-interaction	Scanning-service	2.02 M
Hybrid-interaction	Scanning-service	2.02 M
Low-interaction	Malicious	1.46 M
High-interaction	Malicious	1.76 M
Hybrid-interaction	Malicious	1.57 M
Total scanning-services events		**6.07 M**
Total malicious events		**4.8 M**
Total events		**10.87 M**

Table E.3: Total events by type and interaction level

Attack sources by interaction-level

Figure E.3 shows the number of unique IP addresses identified from the malicious traffic over days and interaction level. We observe a steady increase in the total unique IP addresses over days, with a peak from 13 February 2022 and a subsequent decline for the next four days. Furthermore, we observe that the high-interaction level instances received attacks from more unique sources than the other interaction levels.

Interaction Level	#Malicious Events	#Unique IPs
High-Interaction	$1,763,395$	$18,431$
Hybrid-interaction	$1,575,807$	$12,618$
Low-interaction	$1,463,883$	$8,635$
Distinct IPs from all interaction levels	$22,518$	

Table E.4: Summary of malicious events and unique IPs

Table E.4 summarizes the distinct cumulative total number of unique source IP addresses by interaction-level. The maximum number of unique IPs were detected on the high-interaction instances. We identify $22,518$ unique IPs across all the malicious

events. We want to emphasize that in our study, RIoTPot emulates six protocol services, and the unique attack sources summarized in Table E.4 are based on traffic received across these protocol emulations.
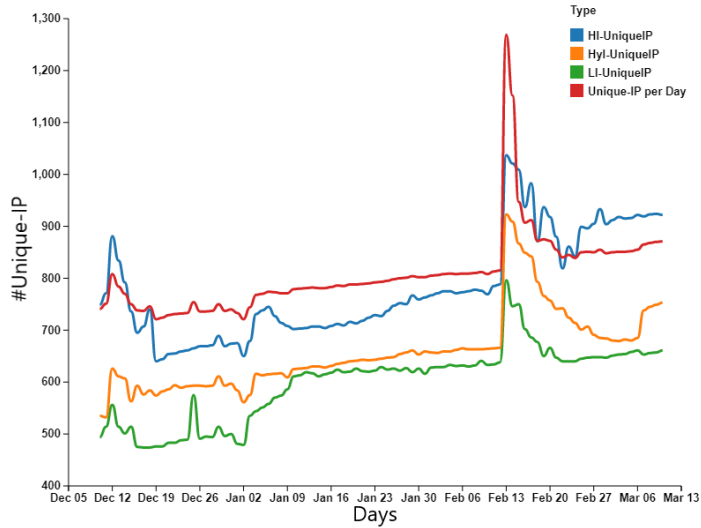

Fig. E.3: Unique-IPs over day and interaction

5.2 Deployment infrastructure

RIoTPot is deployed on both lab (self-hosted) and cloud infrastructure for evaluation. The lab infrastructure hosts three instances, whereas the cloud infrastructure hosts nine instances of RIoTPot. Figure E.4 shows the distribution of malicious events received on RIoTPot instances based on the hosted infrastructure. The number of events on the cloud infrastructure is high due to higher instances of RIoTPot deployed (9 instances) in comparison to the lab (3 instances). Furthermore the figure shows the number of malicious events per instance in lab and cloud infrastructure. We observe that the cloud instances have a higher number of malicious events than the lab instances. This could be because of any suspicious scans or malicious requests that are region-specific.

In Figure E.4 the number of malicious events per interaction level on the lab and cloud infrastructure is summarized. Although there are deviations in the traffic volume, the number of malicious events across all interaction levels is increasing over time. The high interaction instance received more attacks than low and hybrid interaction levels. Note that the number of malicious events on the cloud is higher than in the lab, as there are more instances of RIoTPot deployed on the cloud infrastructure compared to

the lab environment. We observed minor variations in the trend of malicious events in both operating environments.
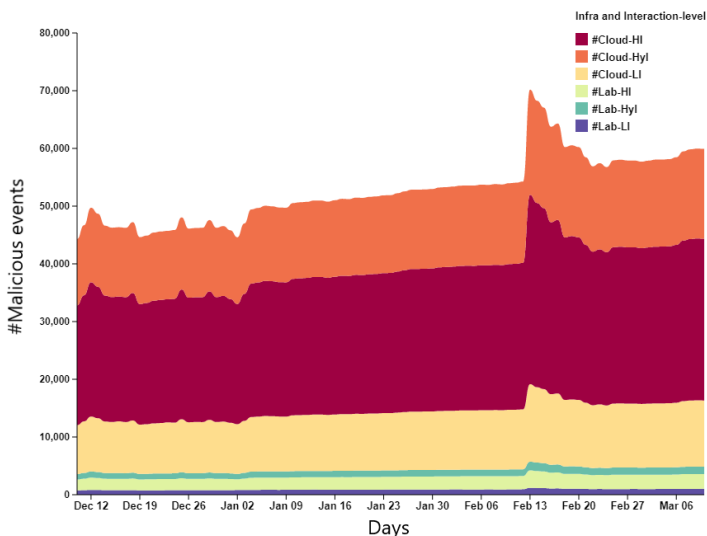
Fig. E.4: Comparison of total malicious events by infrastructure and interaction

5.3 Geographical location

To study the influence of geo-location and region-specific attack distribution, we deploy RIoTPot in four locations - New York City, Frankfurt, Singapore, and Denmark (Lab). Figure E.5 shows the distribution of malicious events from each location and interaction levels. The interaction levels are color-coded, and the solid sphere represents the number of daily events by interaction. The sphere's radius is proportional to the number of events denoted in the legend.

The lowest number of attacks received per interaction per day is 743, while the highest is 13, 287. Compared with the cloud instances, the lab instances received significantly lower malicious events. Initially, the New York instances received higher traffic; however, the Frankfurt instances received the highest traffic overall. The instances deployed in Singapore reported the lowest traffic compared to the other cloud deployments for the whole duration of the evaluation. We observed suspicious events specific to the region that were not seen in other cloud instances. The suspicious events consisted of port scans, brute-force attempts, and attacks specific to protocols emulated by RIoTPot. We find region-specific benign scans from known entities like educational institutions and government-aided organizations other than the malicious events. In the Appendix

Figure E.14 we further discuss how the location and the cloud vs. lab deployment is connected to the number of attacks.
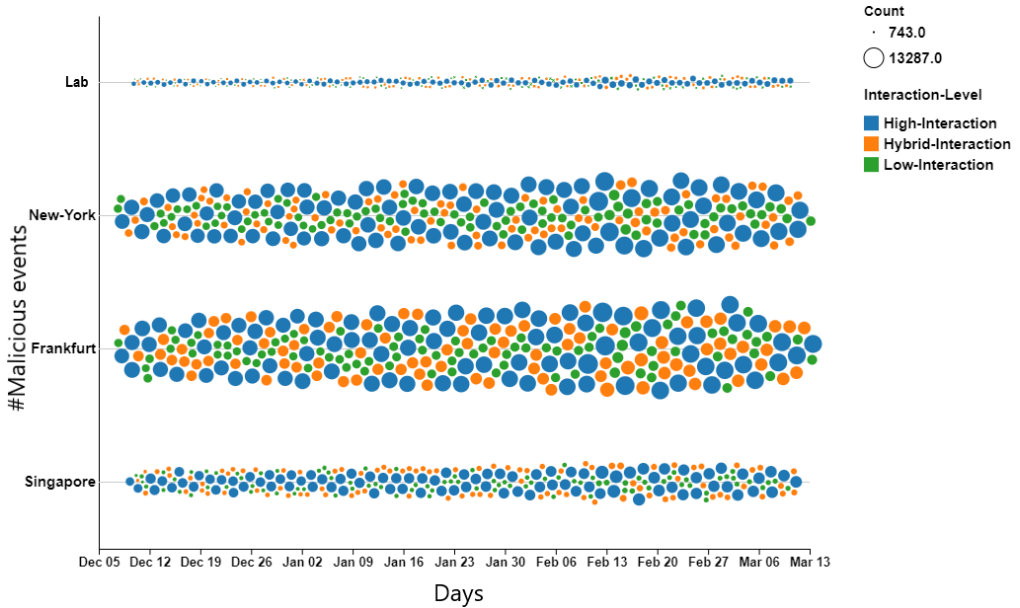
Fig. E.5: Distribution of malicious events across lab and cloud deployments

5.4 Emulated protocols

RIoTPot emulates six protocols - Telnet, SSH, HTTP, Modbus, MQTT, and CoAP. The protocols are emulated in diverse interaction levels across the deployments. Table E.2 summarizes the interaction level of the individual protocols emulated on the instances. Figure E.6 shows the number of malicious events recorded on each protocol. We observed the highest number of events on the SSH protocol, followed by HTTP, Telnet, MQTT, Modbus, and CoAP. Note that the number of events is not unique per source IP and is the total count of the events observed across all interaction levels.

Appendix Figure E.13 summarizes the malicious events received per protocol by interaction level. We observe that the highest number of malicious events in all protocol emulations are received on the high-interaction instances. In protocols like Telnet, SSH, MQTT, and Modbus, we observed a gradual decrease in the number of events on the low-interaction instances. Many attack types like brute-force, poisoning, pivoting and reflection attacks were observed. The attack types are discussed further in Section 6.1.
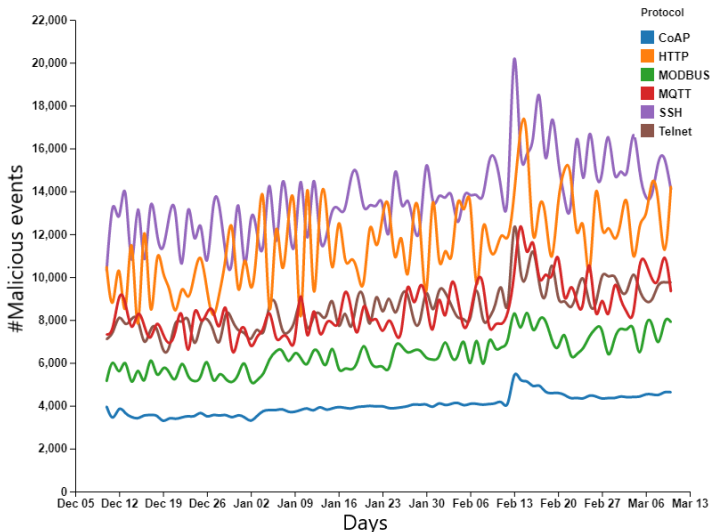
Fig. E.6: Total malicious events by protocol

5.5 Comparison with Conpot

To compare the attacks received on RIoTPot instances, we deploy Conpot [9], a medium-interaction honeypot that can emulate the SSH, Telnet, HTTP, Modbus, and S7 protocols. We deploy Conpot on both the lab and cloud infrastructure, including the geo-locations at which RIoTPot instances are deployed. Appendix Figure E.11 shows the comparison of the malicious events received on RIoTPot and Conpot instances by interaction-level, protocol, and hosted infrastructure. The figure lists the deployed instances (see Table E.2) and the total malicious events received per instance. The Conpot instances simulated four protocols (Telnet, SSH, HTTP, and Modbus) that can be compared with the protocols emulated by RIoTPot instances. In comparison to Conpot, we observe that RIoTPot received a higher number of events on the high and hybrid interaction instances and a similar number of events with low interaction-level instances. The figure also compares the number of malicious events observed on each RIoTPot instance deployed by hosted infrastructure, location interaction-level, and emulated protocols. We observe that the instances deployed on the Frankfurt cloud infrastructure (R7) received the highest number of malicious events. The instances R1,R2,R3,C1 were deployed in the lab; R4,R5,R6,C2 in New York City; R7,R8,R9,C3 in Frankfurt and R10,R11,R12,C4 in Singapore. We suspect that the difference in the number of malicious events between Conpot and RIoTPot could be as a result of limited interaction capabilities of Conpot in comparison to RIoTPot.

6   Discussion

This section discusses the attack types observed during the evaluation process and our findings on the varied malicious events received based on the evaluation parameters. We further state the limitations in our approach, and the ethical considerations followed in our methodology.

6.1   Malicious events

We received a total of 4.8 million malicious events on all the instances. Note that all the events that are not labeled as scanning-services are classified as malicious. The malicious events further include traffic with malicious intent in the requests or payloads. We observe diverse attack types in the malicious traffic received on all instances. The attack types and the exclusive attacks observed during the evaluation are discussed in the following sections.

Attack type by interaction-level

Figure E.7 shows an overview of the attack types observed during our evaluation by percentage and interaction level. We observe attack types like brute-force, poisoning, reflection, and portscans from unknown scanners. The brute-force attacks were the most common observed attacks across all instances and targeted all the emulated protocols. The emulated protocols were configured with weak access controls and credentials to capture advanced attack types. A persistent volume of brute-force attacks was observed at all interaction levels. Furthermore, we see brute-force attacks from the same actors (IPs) across all the interaction levels. However, some regional attacks were observed in specific instances where the attack source appeared to be from the same continent. The poisoning attacks focused on modifying data following unauthorized access. For example, we discover messages on the CoAP protocol to modify data. A larger volume of poisoning attacks was observed on the high-interaction level. In addition, we observe that the attacks from the same attack source interrupted the connection on low-interaction instances while pursuing the connection on high-interaction instances. With this, we entail that the threat actors use specific information from the sessions to determine the pursuit of the attack.

Reflection attacks were detected on the CoAP protocol. We identified reflection attacks when the destination address port is port 80 and the source port is 5832. A larger volume of reflection attacks was again observed in the high-interaction instances. However, we acknowledge that the reflection attacks may be a part of backscatter traffic. We observe malware injection attacks where an attacker tries to download malware from malicious links. The malicious links are identified by analyzing the attacks logged in the *pcap* files. Upon finding a suspicious link in the payload, we check the link with Virustotal [43] to determine the maliciousness. We observe multiple variants of
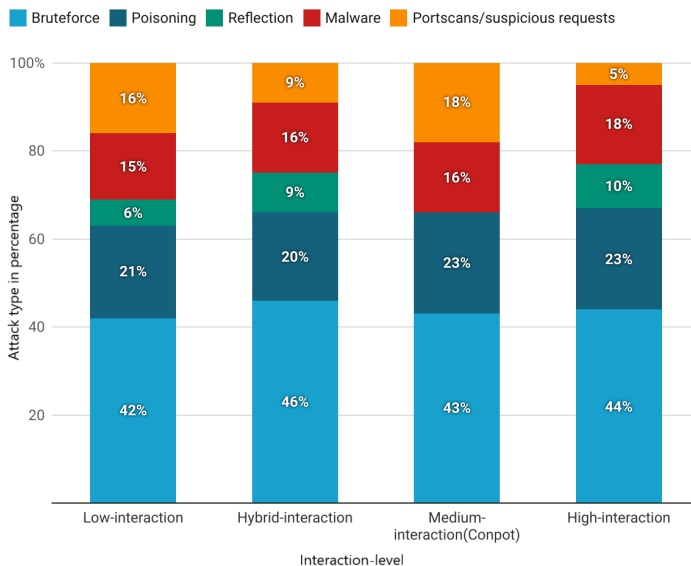
Fig. E.7: Attack types by interaction

the Mirai [47] malware along with LuaBot, Mozi [48] and BrickerBot, among others. Lastly, we observed a specific volume of non-recurring portscanning traffic that was not identical to known scanning-services or attack types. We group such kind of traffic under portscans and suspicious requests.

Attack type by protocol

Appendix Figure E.12 shows the attack types in percentage received by emulated protocols. The attack traffic was stored both as pcap files and the session logs in the attack database. We summarize the attack types found on each emulated protocol.

Telnet and SSH   The Telnet and the SSH protocols received high volumes of bruteforce attacks. The Telnet protocol further received certain malware injections on successful brute force attempts. Upon checking with Virustotal, the malware links observed on the Telnet attacks were detected to be variants of the Mirai family. Similarly, several variants of Mirai were detected on the SSH protocol. In addition to the trivial bruteforce attacks and malware injections, many port scans were observed. This entails that there are still many actors looking for vulnerable Telnet and SSH instances.

HTTP   The HTTP protocol emulated a static login page for a Siemens LOGO 230 RCEo Modbus controller. The protocol received a large number of brute-force attempts. In addition to the brute-force attacks, the HTTP protocol received many Log4j attacks, although the Apache webserver [36] was used for the emulation. The attacks tried injection attacks through RMI (Remote Method Invocation) calls from remote servers. Lastly, a large volume of web scrapers was identified along with unknown scanning services.

MQTT   The MQTT protocol was emulated using the Eclipse-Mosquitto [38] image for the high-interaction and the library for the low-interaction. Although the protocol was configured to allow anonymous logins, there was a large volume of brute-force attacks. Moreover, several data poisoning attempts were detected where the attackers tried to modify data in the queues. In addition, we detected that some attacks created new topics and tried accessing the $SYS\$$ topic specifically. The protocol was scanned mainly by benign scanning services, however, some regional suspicious scans were detected in the instances deployed in Frankfurt and Singapore.

Modbus   The Modbus protocol received a large volume of poisoning attacks to read and modify the data from the registers and coils. The attacks were observed to target three function codes of the nineteen available to fetch information on the device, the reporting server, and the holding register. Furthermore, we observe that most of the attacks used invalid function codes to access the data in the registers. This entails that the scans search for a device with specific function codes for a known exploit.

CoAP   The CoAP protocol was configured to serve on UDP port 5683. Many brute-force attacks were detected that tried to access the CoAP service. Furthermore, we identify data poisoning attacks aimed at modifying values through publishing messages. In addition to the data poisoning attacks, the CoAP protocol saw reflection flooding attacks where the attackers tried to spoof the source packets to divert all the response traffic to a victim. Such attacks were identified by observing the destination port. We observed 27 victim IPs, of which 12 of them were located in Brazil, 4 in South Korea, 4 in Russia, 3 in China, 1 in France and 1 Germany. However, we found that the victim IPs did not have a valid domain and served blank HTML pages.

Region-specific attacks

RIoTPot instances were deployed in four geo-locations. We detect attacks and attack sources of the targeted instances in specific regions. Appendix Table E.7 lists the attack type and volume in percentage of the source of malicious traffic, observed exclusively in specific regions. We observe attacks from specific attack sources on several protocols and attack types. The unique source IPs listed in the table denote the source of attacks

that targeted that region exclusively. While Internet scanning-services are known to use regional hosts for scanning specific locations, the unique IPs listed in Table E.7 are from malicious events. To filter our results, we check if the IPs are Tor relays [49] or from a VPN [50] and find that they are neither. We check the IPs sources on Internet scanning-services like Shodan [40] to find that the hosts had SSH ports open. Furthermore, upon looking up the IPs history, we find that they were recently moved from an ASN.

6.2 Attack sources

A total of $22,518$ unique IPs were identified from the malicious events. To get an understanding of the attack sources, we try to identify the attack sources using banner-based fingerprinting techniques. We send connection attempts on Telnet (port 23) and HTTP requests on port 80, 8080 and 443 on the IPs by using Lift[§], an open-source low-impact fingerprinting tool. We then check the banners and the response for potential identifiers for the attack sources. We take care to send a minimal number of packets in our probes to limit the traffic with the attack sources. Table E.5 shows the device types identified by the banner grabbing checks. A total of 5264 (23%) devices were identified through the banner checks. We suspect that these are compromised devices that are causing attacks on the Internet. In addition to the infected devices, we notice that a vast majority of the HTTP response contained default test websites from Apache, NGinX and Tengine web servers. A total of 4218 (19%) of such responses were identified. We perform a reverse-DNS lookup to identify if there were any domains associated with the IP address ranges and found 21 domains. The domains were associated with some generic top-level domains that include *.art* (5), *.games* (6), *.love* (3), *.website* (2) and *.webcam* (5). Lastly, the rest (58%) of the devices could not be determined.

Device type	Protocol	Count
Router	HTTP	1819
DVR	HTTP	1621
Router	Telnet	721
IP Phone	HTTP	311
Switch	HTTP	287
Switch	Telnet	211
IP Printer	HTTP	176
NAS	HTTP	118
Total		**5264**

Table E.5: Attack-source types

[§]https://github.com/trylinux/lift

6.3   Impact of interaction-levels in honeypots

Our evaluation and findings reveal that the attacks on low-interaction honeypots decreased gradually for some protocols (see Figure E.10), while the high-interaction instances received a higher number of attack events. Hence, our results suggest that the interaction levels play an essential role in attracting specific attacks. Our observation of a gradual decrease in non-recurring scanning probes indicates that modern scanning probes may have checks that help in characterizing if the scanned system is a honeypot[¶]. The malicious events received on the hybrid-interaction model reveal that a combination of low and high-interaction emulation indeed attracts more attacks and successfully deceives the checks from suspicious scanning probes. To summarize the impact of interaction levels, low-interaction honeypots are still effective in capturing scanning and bot traffic. However, we suggest that deploying high-interaction honeypots with limited network configuration on some protocols is more effective to achieve a higher deception layer.

6.4   Limitations

We acknowledge the following limitations in our approach. First, the lab infrastructure is limited to one location, while the cloud deployments range to three locations. This limitation causes an uneven comparison directly between the lab and the cloud deployments. The comparison between the instances deployed in the lab and cloud would be descriptive if the number of deployed instances are the same. Second, we deploy RIoTPot in four cities limiting the scope to three continents. Deploying RIoTPot in all continents would provide a broader perspective of the attack landscape. Third, we limit the number of emulated protocols to six. We acknowledge that more protocols would provide us with an extensive dataset for analysis. However, this work aims to visualize the impact of many evaluation and design parameters that can affect the purpose of honeypots. Fourth, we consider each event as a connection and this entails some limitations in terms of over-counting. As the connection terms vary across protocols, we generalize counting by events and not as connections. Lastly, the dataset does not group the attack data as Netflow formats. Storing the data as Netflow formats facilitates wider integration possibilities with analysis platforms.

7   Conclusion

In this work, we extend RIoTPot, a modular and hybrid-interaction honeypot and facilitate a longitudinal study. To ascertain the impact of parameters like the interaction levels of honeypots, we perform an extensive longitudinal evaluation of RIoTPot by measuring the malicious events gathered based on parameters like interaction level,

---

[¶]In fact, services like Shodan already have such capabilities [51]

deployed infrastructure, geographical location, and emulated protocols. Our results indicate that these parameters are essential in honeypot studies and can provide a broader overview of the attack landscape. The results suggest that high-interaction honeypots receive more sophisticated attacks in comparison with the low-interaction honeypots. Moreover, we observe attacks specific to the hosting environment and geolocation. Compared with Conpot, RIoTPot's high interaction instances received a higher volume of malicious events on all evaluation parameters. We observe diverse attacks like reflection, data poisoning and malware on the honeypots. Lastly, we observe large volumes of traffic from scanning-services that may cause alert fatigue and are false positives.

A   Appendix

A.1   Qualitative comparison

Table E.6 provides an overview of the qualitative comparison of honeypots proposed in related work. The honeypots are compared based on their source-code availability, supported protocols, interaction level, operational environments and known fingerprinting techniques. Most of the proposed honeypots are available as open source and support multiple protocols. However, we observe that there are no high-interaction honeypots are available as open source.

Honeypot	Opensource	Supported protocols	Interaction levels	Virtual vs. Hardware	Known fingerprinting methods
Conpot [9]	Yes	SSH, Telnet, Modbus, BACNet, HTTP	medium	virtual	Yes
Cowrie [8]	Yes	SSH, Telnet	medium	virtual	Yes
Glastopf [11]	Yes	HTTP, HTTPS	medium	virtual	Yes
IoTPot [27]	Yes	No	low	hardware	No
Dionaea [12]	Yes	Yes	medium	virtual	Yes
Honware [18]	No	image based	high	virtual	No
RIoTPot [17]	Yes	image based	low, hybrid. high	virtual	No

Table E.6: qualitative comparison of honeypots

A.2   Appendix: Experiment Overview

Lab setup

The lab setup of our experimental setup is shown in Figure E.8. Three instances of RIoTPot R1,R2,R3 and one instance of Conpot C1 were deployed and assigned a public IP each. All the traffic received and sent from the honeypots are stored in a remote file system as a repository in addition to storing the session parameters in the attack database.
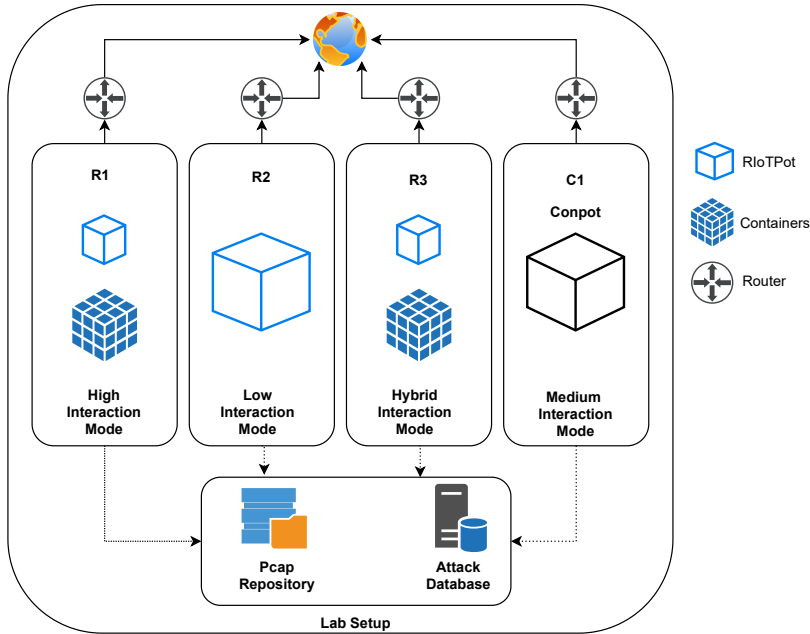
Fig. E.8: RIoTPot evaluation - lab setup

Cloud setup

The cloud setup of the methodology is shown in Figure E.9. The cloud instances are provisioned at three geographical nodes, Frankfurt, New York city and Singapore.

A.3   Appendix: supplementary results

Percentage of events by interaction-level

Figure E.10 shows the percentage of daily events received on RIoTPot instances based on interaction level. We observe a sharp decrease in the percentage of events over time on the low interaction when compared with events on high and hybrid interaction instances. We suspect that this could be because of limited interaction levels.

Comparison by interaction-level, location, honeypot and emulated protocols

Figure E.11 shows the comparison of the number of attacks received by honeypot instance of RIoTPot(R) and emulated protocols with Conpot(C). We observe a high number of malicious events on the high interaction instances of RIoTPot in comparison to

Fig. E.9: RIoTPot evaluation - cloud setup
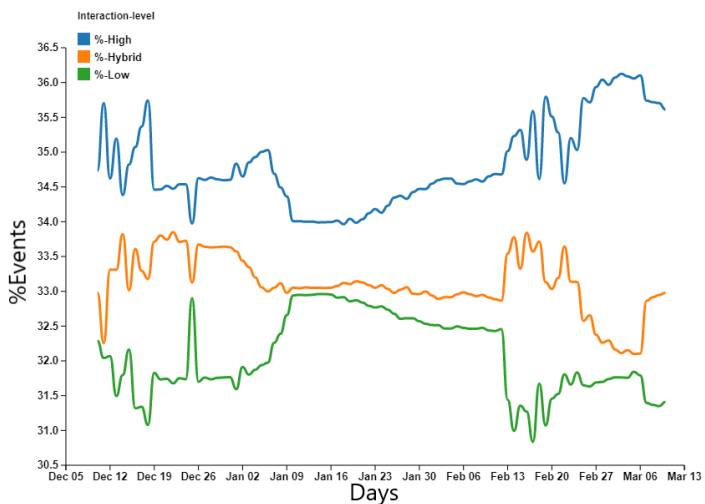
Fig. E.10: Percentage of events by interaction-level and percentage

the other deployments.
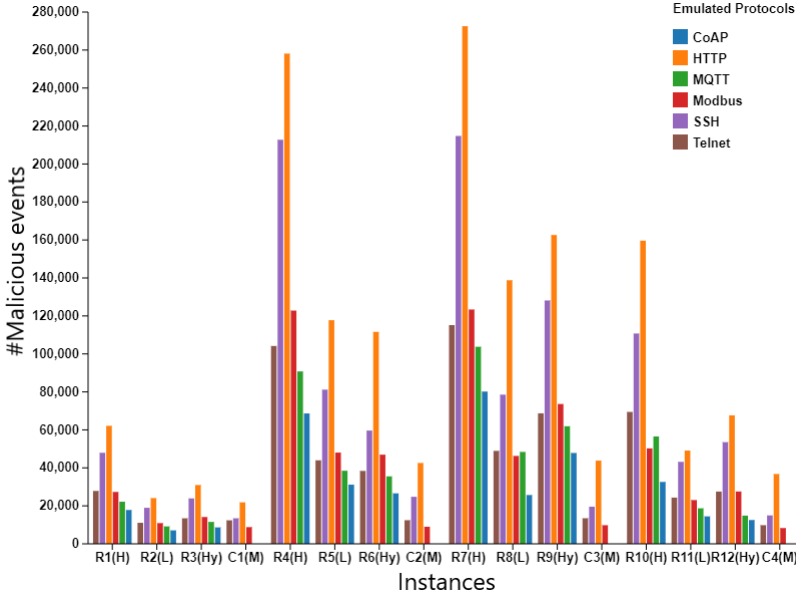
Fig. E.11: Total malicious events by instances
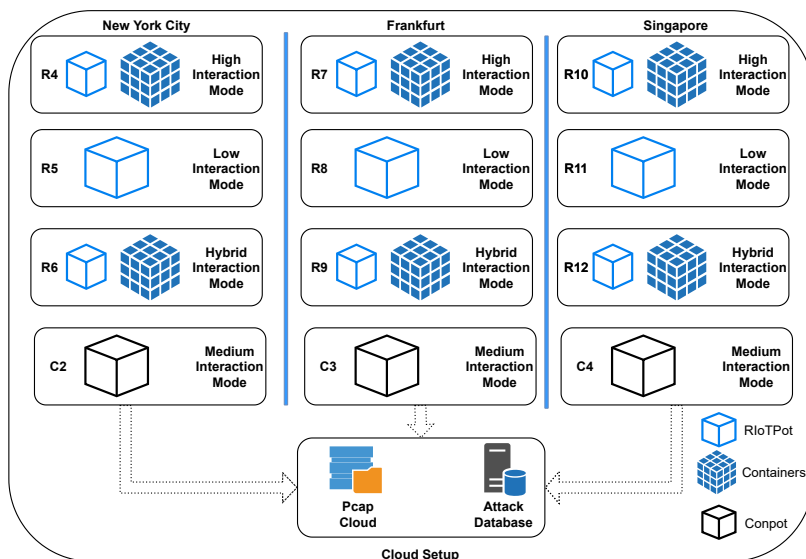
Attack Types by Protocol

Figure E.12 shows the percentage of attacks types on the emulated protocols. We observe multiple attack types that include bruteforce, poisoning, reflection, malware and portscans. Attacks like bruteforce and portscans are observed across all protocol emulations and attacks like malware are observed with Telnet and SSH emulations.

Malicious events received per protocol by interaction level

Figure E.13 summarizes the malicious events received per protocol by interaction level. We observe that the highest number of malicious events in all protocol emulations are received on the high-interaction instances. In protocols like Telnet, SSH, MQTT, and Modbus, we observed a gradual decrease in the number of events on the low-interaction instances. Many attack types like brute-force, poisoning, pivoting and reflection attacks (CoAP) were observed.

Attacks and geographical distribution

Figure E.14 shows the aggregation of the maximum and the minimum number of malicious events obtained per interaction and city. The instances in Frankfurt city recorded
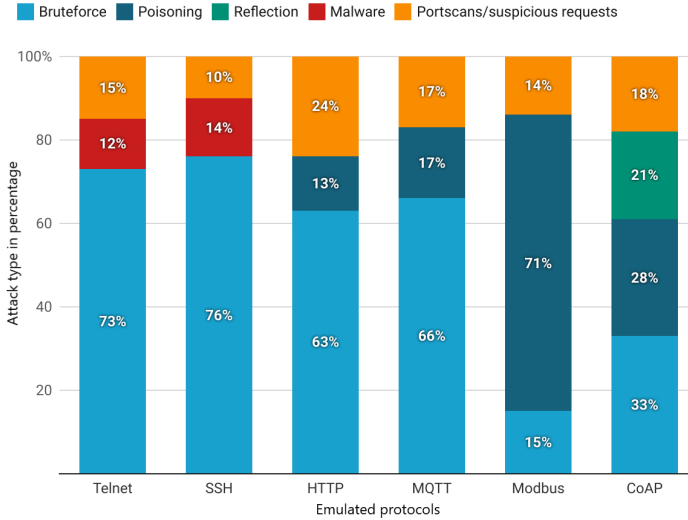
Fig. E.12: Attack types in percentage by emulated protocols
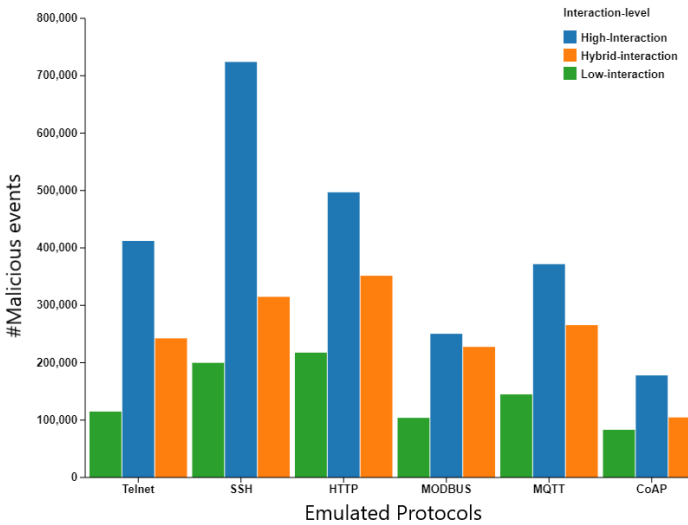
Fig. E.13: Total malicious events by protocol and interaction

the maximum number of events on each interaction level, while the lowest number of events were recorded at the lab infrastructure daily. The high-interaction instances re-

ceived more malicious events, regardless of infrastructure or location, followed by the events on hybrid-interaction instances.
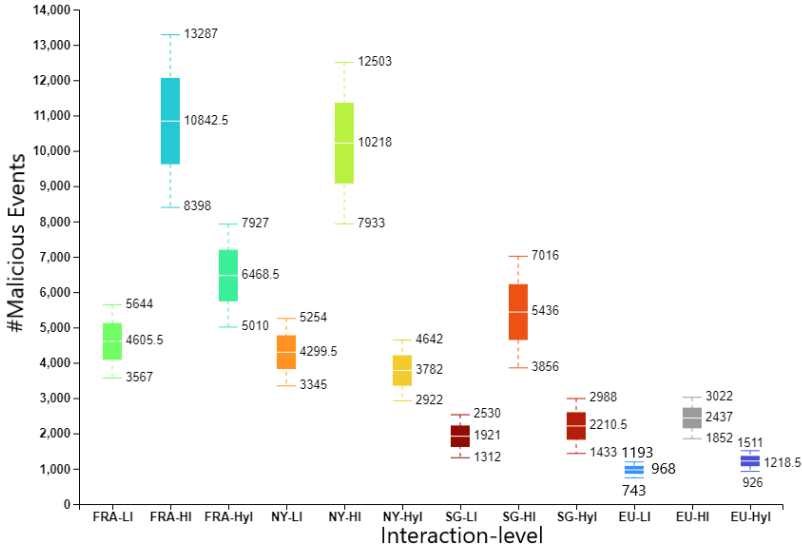
Fig. E.14: Total malicious events by interaction-level and city: lowest, average and highest per day

Region specific attack types

Table E.7 lists the attack type and volume in percentage of the source of malicious traffic, observed exclusively in specific regions.

Instance	Region	Attack-type	Protocol	Unique attacker IP	Volume
R1	Denmark(lab)	Brute-force	Telnet	19	7%
R4	New-York	Brute-force	SSH	36	11%
R7	Frankfurt	Brute-force	Telnet	27	14%
R10	Singapore	Brute-force	Modbus	7	14%
R5	New-York	Brute-force	HTTP	33	17%
R7	Frankfurt	Poisoning	MQTT	21	18%
R10	Singapore	Poisoning	MQTT	13	12%
R10	Singapore	Reflection	CoAP	6	16%

Table E.7: Summary of region-specific attack types

Multistage attacks

Among the attack types discussed above, we observe multistage attacks on the instances. Multistage attacks are attacks that are from the same adversary and sequentially target multiple protocols emulated on the target system. A total of 4786 multistage attacks were detected across all the RIoTPot deployments. Figure E.15 shows the protocols targeted sequentially by adversaries. The start node denotes the protocol first attacked, and the nodes step-2 and step-3 denote sequential attacks on the other protocols carried out by the same adversary. The numbers below the protocols denote the volume of requests received on the protocols used in the attack. Although such behavior is typical in scanning-services, in this case, the attacks are classified as multistage attacks exclusively when malicious content is observed in the requests. A majority of the requests start from the Telnet and SSH protocols. The MQTT protocol is observed to have received the highest volume of subsequent attacks.
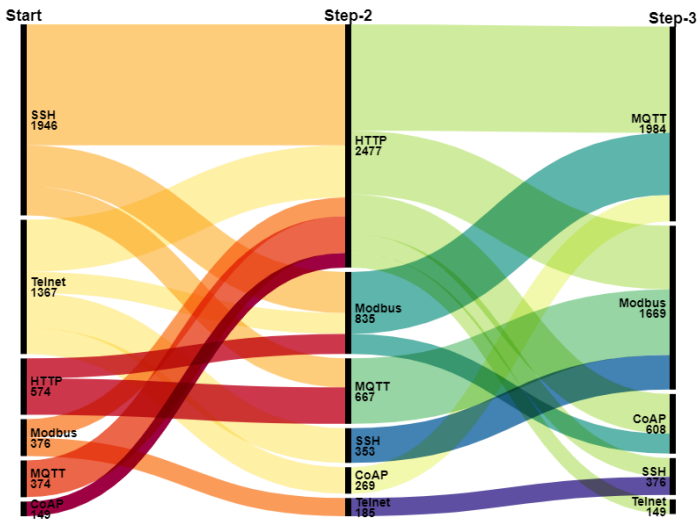
Fig. E.15: Multistage attacks

A.4   Labeling benign traffic

RIoTPot has a database of known Internet-wide scanning services. However it is currently limited to 19 services and thus it may be missing benign services. To further classify the unique source IPs identified in our dataset, we check them with Greynoise API [52]. Greynoise provides a classification of suspicious IPs whether they are benign, malicious or unknown. Figure E.16 shows the classification as retrieved from Greynoise.

Upon correlating the classification from Greynoise to the IPs from which malicious traffic was observed on our honeypot instances, we find that all the IPs were either classified as malicious or unknown from Greynoise.
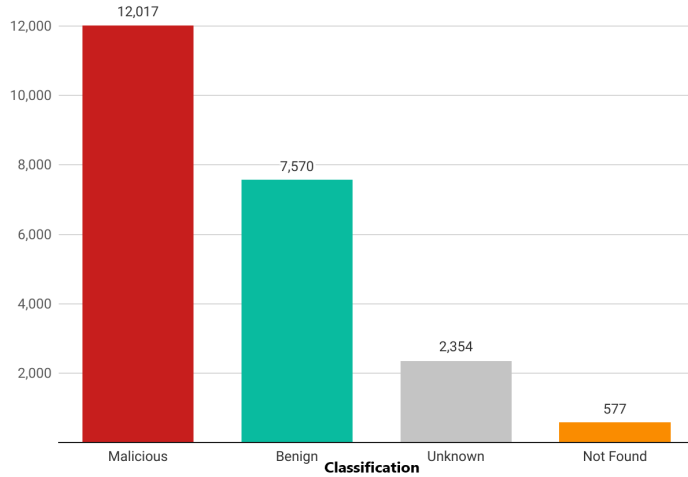
Fig. E.16: Greynoise classification

A.5   Appendix: supplementary discussion

Ethical Considerations

We deploy 12 instances of RIoTPot in varied interaction levels. As honeypots are configured to appear as vulnerable systems, they are prone to be used for causing attacks on the Internet. We configure egress rules on all our deployments to limit the traffic leaving our instances to prevent such misuse. In addition, to avoid the infection spread by any malware attacks, we use ephemeral container instances for our honeypot deployments. New instances are spawned regularly to avoid the spread of any infections.

References

[1] T. Miller, A. Staves, S. Maesschalck, M. Sturdee, and B. Green, "Looking back to look forward: Lessons learnt from cyber-attacks on industrial control systems," *Int. J. Crit. Infrastruct. Prot.*, vol. 35, no. C, dec 2021. [Online]. Available: https://doi.org/10.1016/j.ijcip.2021.100464

[2] L. O. Monaco. (2021) Dag monaco delivers remarks at press conference on darkside attack on colonial pipeline. The United States Department of Justice. [Online]. Available: https://www.justice.gov/opa/speech/dag-monaco-delivers-remarks-press-conference-darkside-attack-colonial-pipeline

[3] F. Robles and N. Perlroth. (2021) 'dangerous stuff': Hackers tried to poison water supply of florida town. The New York Times. [Online]. Available: https://www.nytimes.com/2021/02/08/us/oldsmar-florida-water-supply-hack.html

[4] J. Wang, M. K. Lim, C. Wang, and M.-L. Tseng, "The evolution of the internet of things (iot) over the past 20 years," *Computers & Industrial Engineering*, vol. 155, p. 107174, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360835221000784

[5] S. Srinivasa, J. M. Pedersen, and E. Vasilomanolakis, *Open for Hire: Attack Trends and Misconfiguration Pitfalls of IoT Devices.* New York, NY, USA: Association for Computing Machinery, 2021, p. 195–215. [Online]. Available: https://doi.org/10.1145/3487552.3487833

[6] X. Jiang, M. Lora, and S. Chattopadhyay, "An experimental analysis of security vulnerabilities in industrial iot devices," *ACM Trans. Internet Technol.*, vol. 20, no. 2, may 2020. [Online]. Available: https://doi.org/10.1145/3379542

[7] ENISA. (2020) Enisa threat landscape 2020 - malware. ENISA. [Online]. Available: https://www.enisa.europa.eu/publications/malware

[8] M. Oosterhof, "Cowrie ssh/telnet honeypot," 2016. [Online]. Available: :https://github.com/micheloosterhof/cowrie

[9] L. Rist, J. Vestergaard, D. Haslinger, A. Pasquale, and J. Smith, "Conpot ics/scada honeypot," *Honeynet Project (conpot. org)*, 2013.

[10] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, and M. Fischer, "Hostage: A mobile honeypot for collaborative defense," in *Proceedings of the 7th International Conference on Security of Information and Networks*, ser. SIN '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 330–333. [Online]. Available: https://doi.org/10.1145/2659651.2659663

[11] L. Rist, "Glastopf project," 2009.

[12] D. Tools, "Web honeypot," 2010. [Online]. Available: https://github.com/DinoTools/dionaea/

[13] D. T. A. H. Project, "T-pot: A multi-honeypot platform," 2022.

[14] S. Srinivasa, J. M. Pedersen, and E. Vasilomanolakis, "Gotta catch 'em all: a multistage framework for honeypot fingerprinting," 2021.

[15] A. Vetterl and R. Clayton, "Bitter harvest: Systematically fingerprinting low- and medium-interaction honeypots at internet scale," in *12th USENIX Workshop on Offensive Technologies (WOOT 18)*.   Baltimore, MD: USENIX Association, Aug. 2018, p. 9. [Online]. Available: https://www.usenix.org/conference/woot18/presentation/vetterl

[16] S. Morishita, T. Hoizumi, W. Ueno, R. Tanabe, C. Gañán, M. J. van Eeten, K. Yoshioka, and T. Matsumoto, "Detect me if you... oh wait. an internet-wide view of self-revealing honeypots," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, IEEE.   Washington DC, USA: IEEE, 2019, pp. 134–143.

[17] S. Srinivasa, J. M. Pedersen, and E. Vasilomanolakis, "Riotpot: a modular hybrid-interaction iot/ot honeypot," in *26th European Symposium on Research in Computer Security (ESORICS) 2021*, Springer.   Darmstadt, Germany: Springer, 2021.

[18] A. Vetterl and R. Clayton, "Honware: A virtual honeypot framework for capturing cpe and iot zero days," in *2019 APWG Symposium on Electronic Crime Research (eCrime)*.   Pittsburgh, PA, USA: IEEE, 2019, pp. 1–13.

[19] J. D. Guarnizo, A. Tambe, S. S. Bhunia, M. Ochoa, N. O. Tippenhauer, A. Shabtai, and Y. Elovici, "Siphon: Towards scalable high-interaction physical honeypots," in *Proceedings of the 3rd ACM Workshop on Cyber-Physical System Security*, ser. CPSS '17.   New York, NY, USA: Association for Computing Machinery, 2017, p. 57–68. [Online]. Available: https://doi.org/10.1145/3055186.3055192

[20] S. Sentanoe, B. Taubmann, and H. P. Reiser, "Sarracenia: Enhancing the performance and stealthiness of ssh honeypots using virtual machine introspection," in *Secure IT Systems*, N. Gruschka, Ed.   Cham: Springer International Publishing, 2018, pp. 255–271.

[21] S. Litchfield, D. Formby, J. Rogers, S. Meliopoulos, and R. Beyah, "Poster: Rethinking the honeypot for cyber-physical systems," in *Poster at IEEE Symposium on Security and Privacy*.   San Jose, California: IEEE, 2016.

[22] C. Irvene, D. Formby, S. Litchfield, and R. Beyah, "Honeybot: A honeypot for robotic systems," *Proceedings of the IEEE*, vol. 106, no. 1, pp. 61–70, 2018.

[23] M. Dodson, A. R. Beresford, and M. Vingaard, "Using global honeypot networks to detect targeted ics attacks," in *2020 12th International Conference on Cyber Conflict (CyCon)*, vol. 1300.   Estonia: IEEE, 2020, pp. 275–291.

[24] W. Z. Cabral, C. Valli, L. F. Sikos, and S. G. Wakeling, "Analysis of conpot and its bacnet features for cyber-deception," in *Advances in Security, Networks, and Internet of Things*, K. Daimi, H. R. Arabnia, L. Deligiannidis, M.-S. Hwang, and F. G. Tinetti, Eds.   Cham: Springer International Publishing, 2021, pp. 329–339.

[25] P. D. Ali and T. G. Kumar, "Malware capturing and detection in dionaea honeypot," in *2017 Innovations in Power and Advanced Computing Technologies (i-PACT)*.   Vellore, India: IEEE, 2017, pp. 1–5.

[26] F. Dang, Z. Li, Y. Liu, E. Zhai, Q. A. Chen, T. Xu, Y. Chen, and J. Yang, "Understanding fileless attacks on linux-based iot devices with honeycloud," in *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '19.   New York, NY, USA: Association for Computing Machinery, 2019, p. 482–493. [Online]. Available: https://doi.org/10.1145/3307334.3326083

[27] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "IoTPOT: Analysing the rise of IoT compromises," in *9th USENIX Workshop on Offensive Technologies (WOOT 15)*.   Washington, D.C.: USENIX Association, Aug. 2015. [Online]. Available: https://www.usenix.org/conference/woot15/workshop-program/presentation/pa

[28] A. Ziaie Tabari and X. Ou, *A Multi-Phased Multi-Faceted IoT Honeypot Ecosystem*.   New York, NY, USA: Association for Computing Machinery, 2020, p. 2121–2123. [Online]. Available: https://doi.org/10.1145/3372297.3420023

[29] V. Valeros and S. Garcia, "Hornet 40: Network dataset of geographically placed honeypots," *Data in Brief*, vol. 40, p. 107795, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352340922000075

[30] T. Barron and N. Nikiforakis, "Picky attackers: Quantifying the role of system properties on intruder behavior," in *Proceedings of the 33rd Annual Computer Security Applications Conference*, ser. ACSAC '17.   New York, NY, USA: Association for Computing Machinery, 2017, p. 387–398. [Online]. Available: https://doi.org/10.1145/3134600.3134614

[31] M. Mohammed, M. Elish, and A. Qusef, "Empirical insight into the context of design patterns: Modularity analysis," in *2016 7th International Conference on Computer Science and Information Technology (CSIT)*, 2016, pp. 1–6.

[32] Golang, "Go language," 2021. [Online]. Available: https://golang.org/

[33] Docker. (2022) Dockerhub. Docker. [Online]. Available: https://hub.docker.com/

[34] Busybox. (2022) Busybox dockerhub. BusyBox. [Online]. Available: https://hub.docker.com/_/busybox

[35] Linuxserver.io. (2022) Openssh dockerhub. OpenSSH. [Online]. Available: https://hub.docker.com/r/linuxserver/openssh-serve

[36] T. A. H. S. Project. (2022) Httpd dockerhub. The Apache Project. [Online]. Available: https://hub.docker.com/_/httpd

[37] OITC. (2022) Modbus-server dockerhub. OITC. [Online]. Available: https://hub.docker.com/r/oitc/modbus-server

[38] E. Project. (2022) Eclipse mosquitto dockerhub. Eclipse Project. [Online]. Available: https://hub.docker.com/_/eclipse-mosquitto

[39] plgd. (2022) Coap-gateway. plgd. [Online]. Available: https://hub.docker.com/r/plgd/coap-gateway

[40] SHODAN, "Shodan," 2021. [Online]. Available: https://www.shodan.io/

[41] Censys, "Censys search," 2021. [Online]. Available: https://censys.io/

[42] A. Wick and community. (2022) Arkime. Arkime. [Online]. Available: https://arkime.com/index#home/

[43] Virustotal, "Virustotal," 2022. [Online]. Available: https://www.virustotal.com

[44] V. Valeros and S. Garcia, "Hornet 40: Network dataset of geographically placed honeypots," *Data in Brief*, vol. 40, p. 107795, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352340922000075

[45] D. Ocean, "Digital ocean droplet monitoring," 2022. [Online]. Available: https://docs.digitalocean.com/products/monitoring/

[46] R. Trapkickin, "Who is scanning the internet?" 2015.

[47] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the mirai botnet," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1093–1110. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis

[48] Microsoft. (2021) The mozi botnet. Microsoft. [Online]. Available: https://www.microsoft.com/security/blog/2021/08/19/how-to-proactively-defend-against-mozi-iot-botnet/

[49] T. T. Project. (2022) Exonerator. The Tor Project. [Online]. Available: https://metrics.torproject.org/exonerator.html

[50] iphub. (2022) iphub. iphub. [Online]. Available: https://iphub.info/

[51] SHODAN, "Honeypot or not?" 2022. [Online]. Available: https://honeyscore.shodan.io

[52] GreyNoise, "Greynoise," 2022. [Online]. Available: https://viz.greynoise.io/

Paper F

Deceptive directories and "vulnerable" logs: a honeypot study of the LDAP and log4j attack landscape

Shreyas Srinivasa, Jens Myrup Pedersen, Emmanouil Vasilomanolakis

The layout has been revised.

Abstract

The Lightweight Directory Access Protocol (LDAP) has been widely used to query directory services. It is mainly utilized for reading, writing, and searching directory services like the Active Directory. The vast adoption of LDAP for authentication has entailed several attack attempts like injection attacks and unauthorized access due to third-party key storage. Furthermore, recent vulnerabilities discovered in libraries like the Log4j can lead adversaries to obtain unauthorized information from the directory services through pivoting attacks. Moreover, the LDAP can be configured to operate on UDP, motivating adversaries to exploit it for Distributed Reflection Denial of Service attacks (DRDoS). This paper presents a study of attacks on the LDAP by deploying honeypots that simulate multiple profiles that support the LDAP service and correlating the attack datasets obtained from honeypots deployed by the Honeynet Project community. We observe a total of 39,388 malicious events targeting the honeypots and discover 273 unique attack sources performing pivot attacks in a period of one month.

1 Introduction

The Lightweight Directory Access Protocol (LDAP) has been used for querying and searching the directory services over many years. As the name suggests, LDAP is a lightweight implementation and the Internet variant of the Directory Assistance Service (DAS) from the X.500 protocol (aka. Directory Access Protocol) [1, 2]. Due to its light implementation, many applications support LDAP for synchronizing and managing directory services (e.g., the Active Directory Server from Microsoft). LDAP allows cross-platform clients to query the directory services that contain attribute-value pairs of users, applications, computers, and devices in the network through an LDAP client [3]. Enterprise applications use LDAP for authentication in applications that include email clients, SSH, server, and workstation access.

However, over the years, there have been many vulnerabilities in LDAP that enable injection attacks, unauthorized access, and remote code execution capabilities [4–6]. As many enterprise applications use LDAP for authentication, attackers are highly motivated to exploit the protocol to gain unauthorized access into the targeted infrastructure. According to the ENISA Threat Landscape Report 2021, there were several DDoS campaigns that leveraged UDP-based LDAP services in 2020. It was observed that a wave of DDoS attacks that targeted several Internet Service Providers in France, Belgium and Netherlands leveraged DNS and LDAP services for amplification attacks [7]. Furthermore, Internet scanning data from *Project Sonar* [8], shows up to three million LDAP services on the Internet with open TCP port 389 that accept unencrypted requests, implying that misconfigured LDAP services can lead to attacks of significant impact.

Honeypots are deception systems that simulate target systems or services. They

work as decoys to attract attacks and store all the attack traffic. Traditionally, honeypots have been used to gather attacks from bots and as an effective source for threat intelligence data. There are several open-source honeypot projects, some maintained by the *Honeynet Project*, that are focusing either on specific protocols or vulnerabilities [9]. The simulation ranges across diverse application protocols used in IT, OT (Operational Technology), and IoT environments. Honeypots have been an obvious choice to study attack trends and, more recently, about attacker behavior psychology [10].

In this paper, we aim to extend and deploy a honeypot that simulates open-source implementations of directory services to gather attack trends in LDAP. Moreover, we add a *Log4j* component to our honeypots to allow an analysis of pivoting attacks towards LDAP. Furthermore, we enhance our findings by correlating them with attack data gathered from honeypots deployed by the Honeynet Project. We summarize our contributions as follows:

- We extend an open-source honeypot to simulate three different LDAP profile services.

- We deploy LDAP honeypots and perform an analysis of the attacks received on the honeypots.

- We correlate the attacks received in our honeypots with attack data from the Honeynet Project.

2   Related Work

In this section, we discuss related work in the areas of LDAP attack types and LDAP honeypots.

2.1   LDAP attacks

Several vulnerabilities have been reported on the LDAP over the years. These include Denial of Service attacks, remote code execution and privilege escalation on different independent LDAP implementations [11]. Furthermore, more recently, the LDAP has been exploited as a part of APTs that exploit other vulnerabilities (for example, CVE-2021-44228 of the Apache Log4j vulnerability) [12]. Early research from Alonso et al. show injection techniques possible through the LDAP [5]. The authors present injection techniques by manipulating the filters used for searching the directory services. Obimbo et al. present the risks of using LDAP as an authentication protocol by executing a DoS attack exploiting the TCP three-way handshake required for connection initialization with an LDAP server [4]. More recently, Jeitner et al. presented techniques to inject malicious payloads to launch injection attacks on protocols like DNS,

LDAP, and Eduroam [6]. As LDAP is extensively used in enterprise infrastructure as an authentication service, any potential attack vector towards LDAP is of high risk.

2.2   LDAP honeypots

Early work on LDAP Honeypots was proposed from Grimes [13]. The author provides an overview of honeypots in general and Windows-based honeypots that administrators can deploy to detect potential zero-day attacks. Furthermore, the author provides an overview for modeling honeypots for windows-based environments and protocols by using scripts from the *HoneyD* honeypot framework [14–16]. The *HoneyD* honeypot framework acts as a daemon that can create virtual hosts on a network that can be configured to run arbitrary services. The daemon can run on multiple addresses and provide scripts to emulate an entire device or a specific protocol. Moreover, there is active research that proposes using Honeytokens, a subset of honeypots that emulate a digital entity like user accounts, files, and folders to detect malicious activity or infections. For instance, Lukas et al. propose the creation of fake user accounts as honeytokens on Active Directory Server to capture malicious access attempts [17].

The T-Pot project [18] is a collection of 25 different honeypots that includes the Log4Pot honeypot [19]. Log4Pot simulates a vulnerable Log4j environment and can be configured to listen on multiple ports. The honeypot further provides a log analysis tool that extracts the attack payloads, decodes them and builds a timeline of attacks. The GreedyBear Project [20] aggregates the attack data from the honeypots of the T-Pot project, specifically from the Log4Pot and Cowrie honeypots, and converts them into actionable feeds to facilitate threat intelligence. The GreedyBear project is currently maintained by the Honeynet Project [9] and provides public access to feeds aggregated by the GreedyBear project. Nevertheless, there is no work on honeypots that aims at capturing attacks specific to LDAP. We address this gap by extending an open-source honeypot to simulate directory services with LDAP and capture the attacks [21].

3   Methodology

This section presents the methodology for the LDAP honeypot implementation, the experimental setup and the analysis of attack data from the *Honeynet Project* community.

3.1   LDAP honeypot

To simulate LDAP service, we extend RIoTPot, an open-source honeypot that is modular and capable of operating in hybrid-interaction levels [21]. RIoTPot provides high-interaction capability by running services on dedicated, ephemeral containers with capturing the traffic as *pcap* files and in an attack database. Leveraging the modular feature

of RIoTPot, which facilitates easy integration of protocols and services into the simulation portfolio, we integrate three profiles: Apache Directory Server [22], OpenLDAP [23] and OpenDJ [24]; that support the LDAP service and run them in containerized mode. We set up individual containers of the three profiles and utilize RIoTPot's orchestration and logging features to capture the attack traffic. Furthermore, we simulate a webservice with the *Log4J* vulnerability [12] that refer to the directory services simulated by the profiles in containers. In total, we deploy three webservices that connect to individual directory services. We describe the simulated profiles in detail below.

Apache Directory Service

The Apache Directory Server (ADS) [22] is an open-source, extendable implementation of Directory services. The service is implemented using the Java programming language and can be embedded as a module in a server application. ADS supports the communication through LDAP and is compliant with the LDAP v3. In addition to the LDAP, ADS supports Kerberos 5 and the Change Password protocols. Furthermore, ADS uses an adaptation of the X.500 basic access control scheme with subentries to control access and attributes within the Directory Information Tree (DIT). The directory service can be configured through an LDIF file, a known format to define the properties of DIT, directory objects, and attributes. The Apache community actively maintains the ADS open-source repository.

OpenLDAP

OpenLDAP is an open-source implementation of LDAP [23]. The package includes a stand-alone LDAP load-balancing daemon (*lloadd*) , a standalone LDAP service daemon (*slapd*) and libraries that implement LDAP with additional utilities. The *lloadd* listens for LDAP connections on a specified number of ports and forwards the LDAP operations received over these connections to be processed by the backend, while the *slapd* listens to incoming LDAP requests and responds to the LDAP queries received over the connections. In addition, the *slapd* offers operation in *tool* mode which provides multiple profiles for the daemon.

OpenDJ

OpenDJ is an opensource LDAPv3 compliant implementation of the directory service, developed using Java [24]. The implementation features scalability for large domains, monitoring tools, and replication between multiple instances. In addition to LDAP v3, OpenDJ supports the Directory Service Markup Language (DSMLv2). The OpenIdentity Platform actively maintains the OpenDJ project.

HTTP Service with Log4j vulnerability

Log4j is an open-source logging Java library that provides multiple logging levels for debugging applications. The library is extensively used by applications developed in Java. Recently, a bug in the Log4j library was disclosed in which an attacker can perform remote code execution on the victim using the library for debug-logging [12]. This vulnerability allows unauthorized users to run arbitrary code on the target machine when a configuration uses a JDBC Appender with a JNDI LDAP data source URI [25]. Attackers can spawn malicious LDAP servers to carry out the Log4j attacks on the victims. To understand if there are any potential pivot attacks, that may target the LDAP services through the Log4j exploit, we enhance our honeypot instances (see also experimental setup below) with an HTTP service that showcases the Log4j vulnerability and configure them to connect to individual directory services. The websites simulate a login dashboard with a welcome header, fields for user login, and a login button. The login button performs a standard procedure of verifying the username and password from the directory service configured. The websites are each hosted on the same instance as the directory simulations, and a search user is configured with the websites to be able to search the directories, which enables the examination of LDAP injection attacks.

3.2   Experimental setup

To capture attacks on individual profiles, we deploy RIoTPot on three hosts, with each RIoTPot instance simulating a directory service and an HTTP service. Figure F.1 shows the experimental setup of the honeypots in our lab environment. Each host is assigned a public IP address and has ports 389 (LDAP) and 80 (HTTP) open to the Internet. The traffic from each host is captured as a pcap file and stored in a remote file repository. Furthermore, all traffic received on ports 80 and 389 are logged in an attack database. The file repository and the attack database are set up on a remote host to avoid disruption in logging in case of a crash. The directory service is configured with basic authentication and is set with an admin username with a non-complex password. We configure all the directory services with the same domain name (LDAP.xxx.xx) and are initialized with five organization units and 120 users to look similar to a production service.

3.3   Honeynet Project dataset

To get a holistic view of attacks, we analyze the data from the honeypots deployed by the Honeynet Project community. In particular, we request the feed from the GreedyBear [20] project that aggregates attacks towards the Log4j vulnerability. We correlate these logs to the findings of our own honeypots. Upon analysis of the Honeynet Project data, we identify JNDI calls in the payloads and find similar attacks in our honeypots. We describe our findings in Section 5.
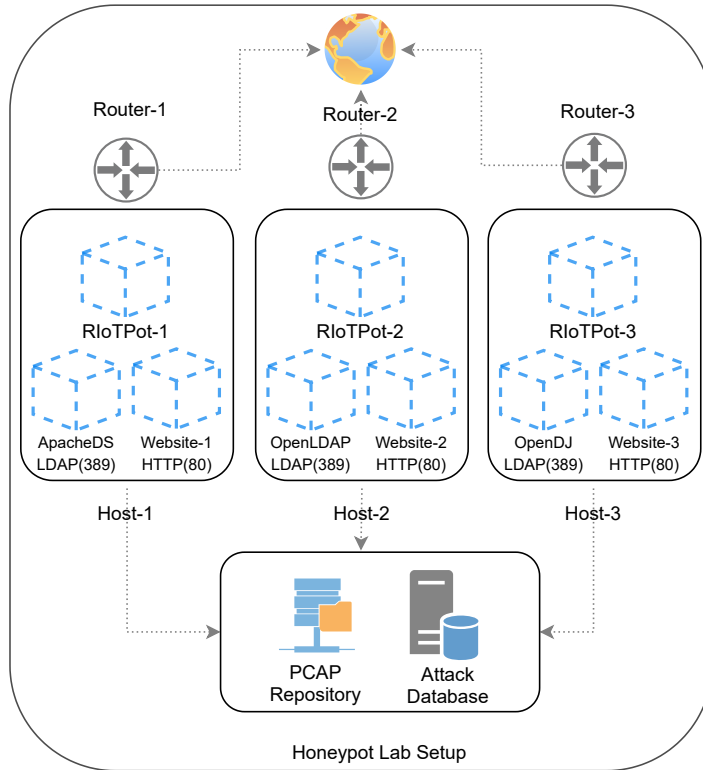
Fig. F.1: Overview of our experimental setup

4   Results

This section lists our findings on the attacks gathered from our honeypots.

4.1   Attack traffic count

We deploy three profiles of open-source Directory Services that support LDAP and add three vulnerable websites with Log4j vulnerability associated with each profile. We classify suspicious traffic as an LDAP attack when an injection pattern or an irregular search is observed in the traffic [5]. Similarly, on the HTTP, we classify the traffic as an attack when brute-force attempts and remote code execution patterns are detected. Figure F.2 summarizes the number of attacks received on each directory service profile on ports 389 and 80 for 30 days. At a glance, we received at total of $39,388$ attacks. The

OpenLDAP directory service received the highest number of attacks on LDAP (2613) in comparison to ApacheDS (2414) and OpenDJ (2341). We observe that the attacks increased after the first 14 days of the deployment on all three profiles. We suspect this could be because of possible listing on the Internet-wide scanning services. Note that the attacks shown are exclusive of probing traffic from known Internet-scanning services. In particular, the HTTP service received a total of 22,673 events and the LDAP received 8,100 events from known scanning services. The traffic from these benign scanning services was identified using the noise-filter module of RIoTPot [21].

Fig. F.2: Attacks received over 30 days - LDAP and HTTP
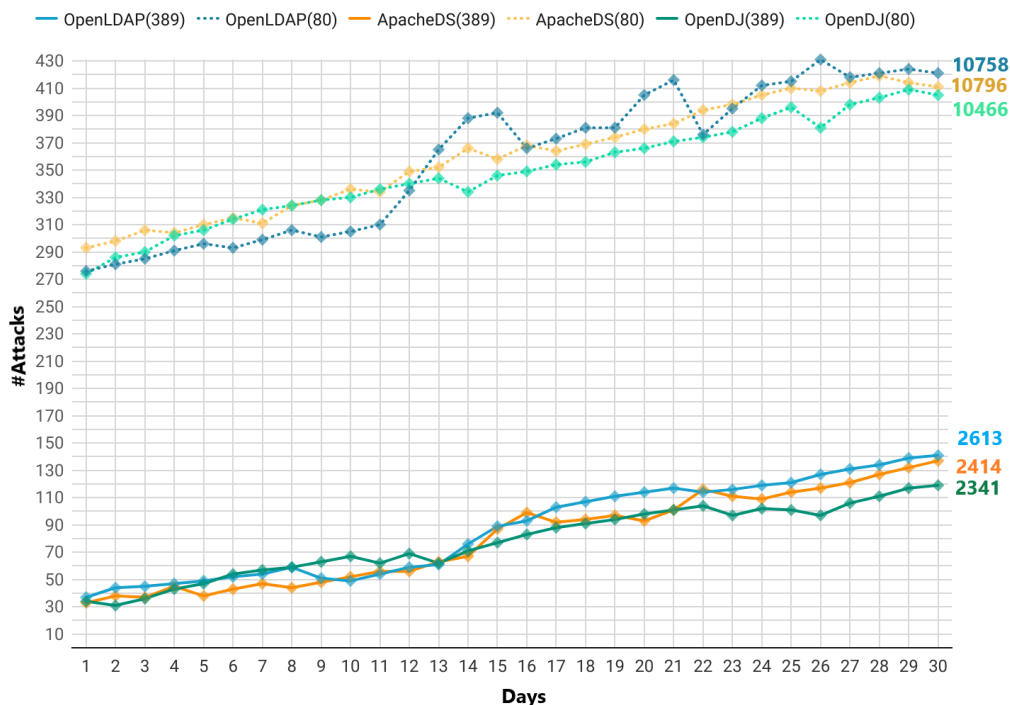
4.2   Attack sources

As a result of exposing our honeypots to the Internet, we receive high traffic volume, primarily benign, from Internet-wide scanning services. Figure F.3 shows the distribution of traffic from scanning services (benign) and attack traffic with malicious intent. RIoTPot filters the traffic received on the honeypots by identifying the probing traffic

from 19 Internet-wide scanning services [21]. Filtering of benign scanning traffic reduces the noise in the gathered data, thereby concentrating on the remaining suspicious traffic. All traffic towards the honeypot instances can be considered suspicious as there is no productive value in interaction with a honeypot. We label suspicious traffic to be an attack upon observing malicious intent in the requests. We observe that the OpenLDAP profile received the highest number of malicious requests compared to the other profiles. The honeypots received traffic from 273 unique attack sources.
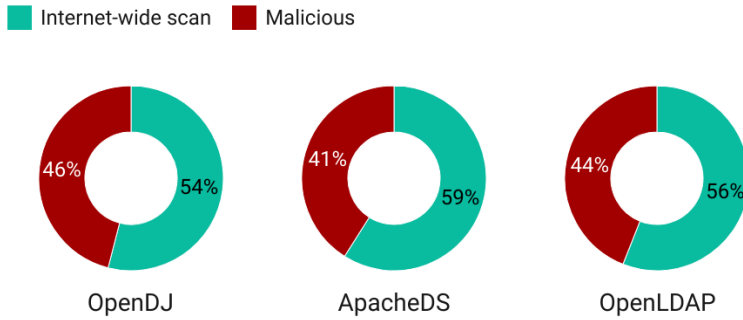
Fig. F.3: Traffic classification on honeypots

4.3   Attack types

We observe multiple attack types in our honeypots, including many LDAP injection attacks, suspicious search, remote code execution, and brute-force attempts. Figure F.4 shows the percentage of different attack types received on each simulated directory service.
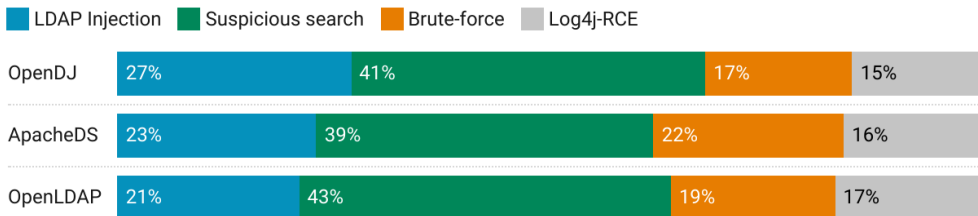
Fig. F.4: Attack types received on honeypots

The OpenDJ profile received the most LDAP Injection attacks in comparison to the other profiles. The attacks aimed at bypassing the authentication by using blind ex-

ploitation techniques to fetch the *userPassword* attribute. The profiles further received random suspicious search queries with logical-operators on the LDAP filters. Moreover, we identified many brute-force attempts on the HTTP webservice. In addition to the brute-force attacks, the websites received attacks that exploited the Log4j vulnerability. We observe fewer attacks towards Log4j in comparison to the other attack types and this could be because of the time elapsed since the disclosure of the vulnerability.

5 Discussion

In this section, we discuss our findings from the analysis of the attack data received from the *Honeynet* community and additional findings from the attack data received on our honeypots.

5.1 Correlating data from the Honeynet Project

The data obtained from the Honeynet Project is an aggregated feed from GreedyBear [20]. The project aggregates data from 30 Log4j honeypot instances. First, we correlate the attack sources observed on both datasets. Over a period of 30 days, the GreedyBear feed had an average of 3,269 events per day and 693 unique source IPs. Figure F.5 shows the correlation of the number of unique IPs that have been observed on Honeynet data and our honeypots over the same period of 30 days. The number of same actors denote the total attack sources observed on both honeypot datasets and the different actors denote the attack sources that were observed exclusively on our honeypots. Upon further analysis, we find that the different actors observed on our honeypots targeted also the LDAP service. The different actors observed on our honeypots may be the result of running both LDAP and Log4j simulations. The attack sources shown in the figure include the attacks received only on the Log4j simulation in our honeypots. Furthermore, we find recurring probes from attack sources that are not from known Internet-wide scanning services and appear to be performing pivot attacks. In addition, we examined the code that was called through RMI to find patterns. Upon analysis we find similarities in the code that aimed at performing LDAP injections from many sources.

5.2 Attack samples

We list sample attacks in appendix Table F.1 for each attack type categorized in Figure F.4. The table further lists different LDAP injection attack types and samples observed on our honeypots. The *Authentication Bypass* attacks aimed at injecting filtered LDAP queries with sequences to bypass authentication. The privilege escalation attacks aim at listing unauthorized directory contents bypassing a search sequence with a low-security level. We observe blind injection attacks that request a Boolean operation to check if an
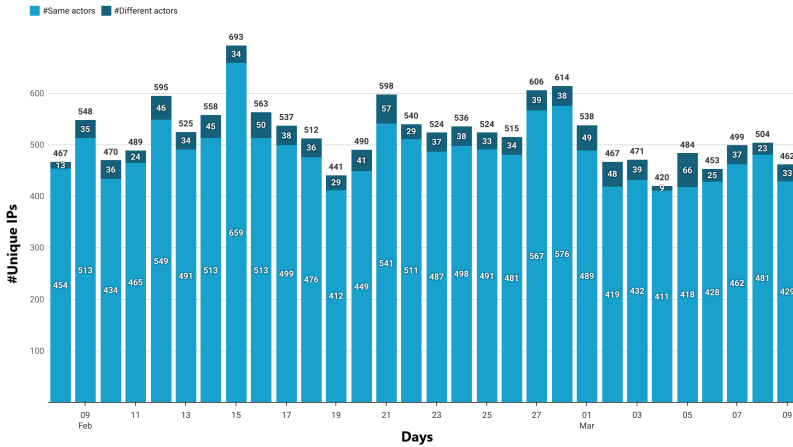
Fig. F.5: Correlation of attack sources from the Honeynet Project and our honeypots

admin class exists that belongs to a *domain* type. In addition, the honeypot instances received many suspicious search query requests. For instance, the sample listed in Table F.1 requested a sequence from the LDAP service on the same host. This search entails that the adversary previously performed reconnaissance to discover open LDAP ports on the host. Many brute force attacks were identified in which adversaries tried to log in via a list of passwords. We further determine, by checking the word list order, that the passwords used were part of the NMap default password list [26]. Lastly, there were Log4j attacks observed that performed RMI calls. We list some sample Log4j exploits received in Table F.1.

5.3 Pivot attacks

Pivoting attacks can be described as attacker movement from one compromised system to more systems within the same or remote infrastructure. We observe some attacks that try to pivot into the directory services by leveraging the Log4j vulnerability through LDAP injection techniques. Upon examining the code from RMI calls specified through JNDI, we find LDAP filters that aim to list all organizational units and enumerate domain users and domain admins groups. We observe such attacks on all three simulation profiles of our experiment. Figure F.6 depicts the number of pivoting attacks observed on each simulation profile. The attacks begin with targeting the Log4j vulnerability, and sequentially move on to target the simulated directory services through LDAP. We observe that out of 429 unique attack sources (observed exclusively on Log4j), 273 of them attempted pivot attacks on the directory services.

Fig. F.6: Pivot attacks overview

5.4   Limitations

We acknowledge the following limitations in our approach. First, we exclusively consider open-source implementations of directory services and LDAP. This limits our scope as most enterprises use Microsoft Active Directory as their directory service [27]. Second, our work is further limited in the simulation of LDAP operational modes, such as LDAPS and CLDAP. The simulation of CLDAP would provide an overview of the reflection-based attacks. Third, though we simulate a high-interaction profile for the directory services and LDAP, we limit the experiment in terms of the domain simulation by using an unregistered domain. Hence, using a registered domain in our experiment may enhance the deception layer and appear more attractive for adversaries. Lastly, the total attack events observed on each profile are the result of a month study only; an extended study is needed for a more holistic understanding of the field.

5.5   Ethical considerations

As honeypots are systems that simulate vulnerable environments, they can be leveraged by adversaries to cause attacks on the Internet. To prevent such attacks, we limit the egress traffic from our honeypots. Furthermore, the containers spawned from our

honeypots for simulation are ephemeral, such that new instances are created periodically to avoid spread of infections. In regards to the dataset from the Honeynet project, we take care in not disclosing the IP addresses of honeypots deployed by the community.

6   Conclusion

This paper conducts a honeypot study of the attacks on LDAP by deploying three open-source directory service profiles with the webservers simulating the Log4j vulnerability. We observe many attack types, including LDAP injection attacks and suspicious search queries. Lastly, we summarize the attack types and correlate our findings with the data from the Honeynet community. As future work, we aim to perform a longitudinal study of LDAP honeypots with extended profiles that include the Active Directory.

A   Appendix

A.1   Samples of attack types

Table F.1 lists the sample attacks received on our honeypots like LDAP injection, suspicious search queries, brute-force attacks and the Log4j RMI attacks. The table further lists the different types of LDAP injection attacks in particular the authentication bypass technique which aims to gain unauthorized access by injection of a filter that ignores the password attribute in the LDAP query, the privilege escalation attacks which aims at fetching unauthorized information and blind injection attacks that aims at fetching boolean information about specific objects in the directory.

Attack-type	Received Attack Sample
LDAP-Injection Authentication Bypass	&(USER=admin)(&))(PASSWORD=Pwd)
LDAP -Injection Privilege elevation	"www)(security_level=*))(&(directory=html"
LDAP -Injection Blind LDAP Injections	(&(objectClass=admin*)(type=domain*))
Suspicious search	GET /?x=$jndi:ldap://127.0.0.1
Brute-force	#cn=root,cn=users,dc=resilient,dc=dk password
Log4j-RCE	GET /$%7Bjndi:$%7Blower:l%7D$%7Blower:d%7Da$%7Blower:p%7D://************.*.psc****

Table F.1: Samples of attacks received on honeypots

References

[1] M. Rose, "Directory assistance service," in *RFC 1202, Performance Systems International, Inc.* Citeseer, 1991.

[2] B. Smetaniuk, "Distributed operation of the x. 500 directory," *Computer Networks and ISDN Systems*, vol. 21, no. 1, pp. 17–40, 1991.

[3] M. Wahl, T. Howes, and S. Kille, "Rfc2251: Lightweight directory access protocol (v3)," 1997.

[4] C. Obimbo, B. Ferriman *et al.*, "Vulnerabilities of ldap as an authentication service." *J. Information Security*, vol. 2, no. 4, pp. 151–157, 2011.

[5] J. M. Alonso, R. Bordon, M. Beltran, and A. Guzmán, "Ldap injection techniques," in *11th IEEE Singapore International Conference on Communication Systems.* IEEE, 2008, pp. 980–986.

[6] P. Jeitner and H. Shulman, "Injection attacks reloaded: Tunnelling malicious payloads over dns," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 3165–3182.

[7] ENISA. (2020) Enisa threat landscape 2020 - malware. ENISA. [Online]. Available: https://www.enisa.europa.eu/publications/malware

[8] R. Research, "Project sonar," 2021. [Online]. Available: https://www.rapid7.com/research/project-sonar/

[9] T. H. Project, "The honeynet project."

[10] K. J. Ferguson-Walter, M. M. Major, C. K. Johnson, and D. H. Muhleman, "Examining the efficacy of decoy-based and psychological cyber deception," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1127–1144.

[11] MITRE. (2021) Ldap vulnerabilities and disclosures. [Online]. Available: https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=ldap

[12] A. S. Foundation. (2021) Cve-2021-44228. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-23302

[13] Springer, Ed., *Windows Honeypot Modeling.* Berkeley, CA: Apress, 2005, pp. 63–88. [Online]. Available: https://doi.org/10.1007/978-1-4302-0007-9_3

[14] N. Provos, "Honeyd-a virtual honeypot daemon," in *10th DFN-CERT Workshop, Hamburg, Germany*, vol. 2, 2003, p. 4.

[15] N. Provos and T. Holz, *Virtual honeypots: from botnet tracking to intrusion detection.*   Pearson Education, 2007.

[16] N. Provos *et al.*, "A virtual honeypot framework." in *USENIX Security Symposium*, vol. 173, no. 2004, 2004, pp. 1–14.

[17] O. Lukas and S. Garcia, "Deep generative models to extend active directory graphs with honeypot users," *arXiv preprint arXiv:2109.06180*, 2021.

[18] T. Security, "T-pot - the all in one honeypot platform," 2022. [Online]. Available: https://github.com/telekom-security/tpotce

[19] P. Thomas, "A honeypot for the log4shell vulnerability (cve-2021-44228)," 2022. [Online]. Available: https://github.com/thomaspatzke/Log4Pot

[20] (2022) Greedybear honeypot feed. Honeynet Project. [Online]. Available: https://github.com/honeynet/GreedyBear

[21] S. Srinivasa, J. M. Pedersen, and E. Vasilomanolakis, "Riotpot: a modular hybrid-interaction iot/ot honeypot," in *26th European Symposium on Research in Computer Security (ESORICS) 2021*, Springer.   Darmstadt, Germany: Springer, 2021.

[22] Apache. (2021) Apache directory. [Online]. Available: https://directory.apache.org/apacheds/

[23] O. Kuzník. (2021) Openldap. [Online]. Available: https://www.openldap.org/

[24] OpenIdentityPlatform. (2021) Opendj. [Online]. Available: https://www.openidentityplatform.org/opendj

[25] (2021) Cve-2021-44832. Apache Software Foundation. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-44832

[26] G. Lyon, "Nmap network mapper," 2021. [Online]. Available: https://nmap.org/

[27] S. Reimer and M. Mulcare, *Active Directory® for Microsoft® Windows® Server 2003 Technical Reference.*   O'Reilly Media, Inc, 2009.

Internet Security Measurements

Paper G

Open for hire: attack trends and misconfiguration pitfalls of IoT devices

Shreyas Srinivasa, Jens Myrup Pedersen, Emmanouil Vasilomanolakis

The layout has been revised.

Abstract

Mirai and its variants have demonstrated the ease and devastating effects of exploiting vulnerable Internet of Things (IoT) devices. In many cases, the exploitation vector is not sophisticated; rather, adversaries exploit misconfigured devices (e.g. unauthenticated protocol settings or weak/default passwords). Our work aims at unveiling the state of IoT devices along with an exploration of the current attack landscape. In this paper, we perform an Internet-level IPv4 scan to unveil 1.8 million misconfigured IoT devices that may be exploited to perform large-scale attacks. These results are filtered to exclude a total of 8,192 devices that we identify as honeypots during our scan. To study current attack trends, we deploy six state-of-art IoT honeypots for a period of 1 month. We gather a total of 200,209 attacks and investigate how adversaries leverage misconfigured IoT devices. In particular, we study different attack types, including denial of service, multistage attacks and attacks from infected online hosts. Furthermore, we analyze data from a /8 network telescope covering a total of 81 billion requests towards IoT protocols (e.g. CoAP, UPnP). Combining knowledge from the aforementioned experiments, we identify 11,118 IP addresses (that are part of the detected misconfigured IoT devices) that attacked our honeypot setup and the network telescope.

1 Introduction

With the adoption of IoT, there is an increase of misconfigured devices on the Internet. Some are incorrectly configured or left with default configuration, thereby making them vulnerable [1]. Misconfigured IoT devices are exploited on a large scale by malware like Mirai that infect vulnerable devices with bots [2]. A device is considered to be *misconfigured* if its incorrect configuration leads to vulnerabilities. NIST defines misconfiguration as *"An incorrect or suboptimal configuration of an information system or system component that may lead to vulnerabilities"* [3]. Moreover, attacks like denial-of-service, ransomware, or data leaks can be purchased and facilitated through botnets. For instance, many variants of the Mirai botnet and newer IoT malware like GitPaste-12 [4], Kaiji [5], RHOMBUS [6] continue to look for vulnerable devices on the Internet [2]. Furthermore, recent research shows the possibilities of DoS attacks through messaging protocols like MQTT [7, 8] and CoAP [9].

According to the ENISA Threat Landscape Report 2020, malware attacks are the leading and emerging threats worldwide [10]. While it is known that botmasters look for vulnerable devices with misconfigured protocols of Telnet and SSH, research suggests that bot deployments are now possible with IoT-based protocols like MQTT, AMQP, and UPnP [11–14]. With the increasing adoption of IoT in diverse sectors like Industry 4.0, healthcare, and critical infrastructure, we argue that this poses a significant threat.

Heretofore, there has been research on the underlying IoT vulnerabilities and propos-

ing honeypots to analyze the threat actors for specific protocols [15–18]. However, to the best of our knowledge, no work combines an active search for misconfigured devices with an analysis of the attack trends in IoT by deploying multiple honeypots and studying the traffic flow received on a network telescope. In this paper, we unveil the vulnerable aspects of misconfigured services on IoT devices and emphasize the importance of authentication and authorization in IoT protocols and devices.

Our contributions are summarized as follows:

- We perform Internet-wide scans on six protocols: Telnet, MQTT, CoAP, AMQP, XMPP, and UPnP. As a result, we unveil 1.8 million misconfigured IoT devices that can either be infected with bots or be leveraged for a (D)DoS amplification attack. In addition, we use open datasets to complement our findings. Furthermore, our scan takes into account the existence of honeypots. To deal with the lack of ground truth knowledge for deployed honeypots on the Internet, we analyze the response banners from our scan and the static banners returned by open-source honeypots. Hence, we filter out from the results 8, 192 systems that we classify as honeypots.

- We deploy six SOTA *IoT honeypots*, to capture and analyze the attack vectors on the protocols scanned. Moreover, we analyze data from a /8 *network-telescope* with 16 million IP addresses to better understand Internet scanning trends in IoT protocols.

- Combing knowledge from the IPv4 scan, the honeypot deployment and the network telescope traffic analysis, we discover 11, 118 (out of the 1.8 million) misconfigured IoT devices that attacked our honeypot setup and the network telescope.

The rest of the paper is organized as follows. Section 2 introduces the related work in detecting vulnerable IoT devices on the Internet and IoT honeypots. In Section 3 we describe our methodology of finding misconfigured devices on the Internet, detection of honeypots and deploying state-of-the-art honeypots in our lab environment to learn the attack vectors and analysis of FlowTuple data from a network telescope. Section 4 shows the results obtained from our methodology. In Section 5 we discuss the attack trends and findings of our research. Section 6 concludes the paper and discusses potential future work.

2 Related Work

This section discusses the related work in the area of Internet-wide scanning for finding vulnerable IoT devices, IoT honeypots, and IoT honeypot fingerprinting.

2.1 Internet-wide scanning for vulnerable IoT devices

The widespread increase of IoT devices on the Internet has called upon various kinds of research, focusing on their security and trust [19]. The majority of the research in this area includes fingerprinting IoT devices to facilitate exploitation based on their type. However, there is less research that follows the approach of scanning the Internet to find vulnerable devices. Markowsky et al. [20] demonstrate how to scan the Internet for vulnerable IoT devices using the Shodan scan engine [21] and scanning tools like Masscan [22], NMap [23], and PFT [24]. The authors describe multiple ways of finding vulnerable devices on the Internet using banners of known services. The scan finds more than 1.6 million vulnerable devices on the Internet. Although we make use of a similar methodology, i.e, we utilize ZMap and Shodan in our scanning approach, we leverage open datasets and run the scans with custom probes for both TCP and UDP protocols. Furthermore, unlike Markowsky et al. we do not try to connect to the devices after the scanning process. We also use the banners and the initial response received from the hosts from our scans. In addition to results from Shodan, we combine datasets from open projects that do not index the scan results based on banners or responses.

Neshenko et al. [25] make an exhaustive survey of IoT vulnerabilities by an empirical study of the published research work on IoT. Their analysis proposes a taxonomy of IoT vulnerabilities, including their technical details and consequences. The authors also evaluate IoT exploits through analysis of a passive network dataset obtained by a network telescope. The evaluation provides good insights into the number of vulnerable IoT devices by country, infected devices, and malicious IoT traffic. To sum up, there is significant research on fingerprinting of IoT devices using passive data sets. However, there is scarce work on scanning the Internet with custom probes to discover misconfigured IoT devices.

The work of Springall et al. [26] is the closest to ours. The authors attempt to find FTP servers on the Internet that accept anonymous logins and investigate real-world attacks by deploying FTP honeypots. Springall et al. detect more than $20,000$ public FTP servers that allowed write access. The authors focus mainly on the FTP protocol and the anonymous login misconfiguration that allows remote users to authenticate without any access information.

2.2 IoT-Honeypots

The use of honeypots and network telescopes to monitor attacks is not new. Honeypots are deception-based entities that simulate the services of a target system. All connection attempts to a honeypot can be considered malicious as there is no real reason for accessing a honeypot system. Over the years, many honeypots have been proposed, both open-source and research-based, to understand the threats to IoT protocols. The Honeynet Project [27] offers a number of open-source honeypots such as: Conpot [28], Dionaea [29] and HosTaGe [30] that simulate IoT protocols (e.g. Telnet, MQTT, CoAP

Honeypot	Telnet	MQTT	CoAP	AMQP	XMPP	UPnP	Open-source
IoTPot (2016)	Yes	-	-	-	-	-	-
ThingPot (2018)	-	-	-	-	Yes	-	Yes
U-Pot (2018)	-	-	-	-	-	Yes	Yes
IoTCandyJar (2017)	-	Yes	Yes	-	Yes	-	Yes
HosTaGe (2020)	Yes	Yes	Yes	Yes	-	-	Yes
Conpot (2020)	Yes	-	-	-	-	-	Yes
Cowrie (2020)	Yes	-	-	-	-	-	Yes
Dionaea (2020)	-	Yes	-	-	-	-	Yes
MQTT and CoAP Honeypots (2019)	-	Yes	Yes	-	-	-	-
Anglerfish	Yes	Yes	-	-	-	Yes	-

Table G.1: IoT Honeypots

and AMQP). Other honeypots include ThingPot [17], IoTPot [15], UPot [18] and IoT-CandyJar [16].

Table G.1 lists IoT-honeypots and the protocols the simulate. IoTPOT [15] proposes a honeypot and a sandbox environment for capturing Telnet-based attacks. Through IoTPOT, the authors were able to identify four distinct DDoS malware families targeting Telnet-enabled IoT devices based on the attacks gathered. Wang et al. propose ThingPot [17] that emulates the XMPP protocol. The authors also implemented the Philips Hue smart home lighting system profile into ThingPot that emulates the Hue devices like the bridge and the smart lamps. During the evaluation of ThingPot, the authors discovered attacks that tried to gain control of the system and some fuzzing attempts. Hakim et al. propose U-Pot [18], a UPnP-based honeypot framework for capturing attacks on IoT devices that use Universal Plug and Play (UPnP) protocol. The authors claim that U-Pot offers high-interaction capabilities and is agnostic of device type. The authors deploy the profile of the Belkin Wemo smart switch [31] into U-Pot and evaluate its performance by trying to measure the response times from the honeypot. The results are observed to have near similar response times to real devices.

Luo et al. propose IoTCandyJar [16], a machine learning-based honeypot that learns the behavioral knowledge of IoT devices by continuous Internet-wide probing. The honeypot sends Internet-wide probes as seed requests to get response information from devices with specific open ports. The honeypot responds to the attacker queries, using the saved responses and the requests in its training database. HosTaGe [30, 32] is a low-interaction mobile honeypot that emulates many protocols, including IoT protocols like MQTT, CoAP, and AMQP. Further, the honeypot offers device profiles like Arduino, a smoke-sensor, and a temperature sensor for simulation. Shimada et al. implemented MQTT, and CoAP honeypots [33] to observe the possible attack vectors on the IoT messaging protocols. The authors observed a large number of MQTT requests on the honeypot and requests from unknown protocols. Lastly, we discover the Anglerfish honeypot from the results of our honeypot detection methodology which is described

in Section 3. The honeypot is managed by Netlab 360 [34], a commercial security organization.

2.3 Network Telescopes

Data from network telescopes has been utilized in some research to study the scanning trends. Durumeric et al. [35] use the data from an extensive network telescope to gain insights in scanning traffic, behavior, and patterns. The authors reveal many attacks detected from Darknet IP sources and derive many statistical patterns from the scanning data. Similarly, Heo et al. [36] analyze the connection-level log data of a large-scale campus network to study the trends in scanning. The log data used for analysis is acquired from the firewalls deployed in the campus network. The authors provide an in-depth analysis and classification of the scan traffic.

Jonker et al. [37] use four independent datasets that include honeypots and a network telescope to perform a comprehensive analysis of the gathered attacks and introduce a new framework to enable a macroscopic characterization of attacks, attack targets, and DDoS Protection Services. The authors present significant results regarding the global problems caused by DoS attacks and the most targeted types of servers.

Lastly, Richter et al. analyze the unsolicited traffic at firewalls from $89,000$ hosts across $1,300$ networks of a significant Content Distribution Network [38]. Their findings indicate that localized scanning campaigns likely target narrow regions in the IP address space. Their characteristics vary compared to the Internet-wide scanning services in terms of target selection. The authors further compare the suspicious traffic received on the firewalls to the UCSD Darknet Network Telescope [39] and provide a comprehensive analysis of the scanning services.

2.4 IoT-Honeypot Fingerprinting

Honeypot fingerprinting is the process of detecting if a target system is indeed a honeypot. The fingerprinting process may involve either active, passive, or both fingerprinting techniques. Some examples include banner-based, static-response, the use of low-interaction libraries, and response times. Honeypot fingerprinting can help adversaries in avoiding any interaction with a honeypot either directly or through malware propagation. Research on honeypot fingerprinting has increased over time. Early works on honeypot fingerprinting started in 2005 by Holz et al. [40] who queried the target system for known static responses from honeypots. More recent works include Vetterl et al. [41] who systematically detected known open-source honeypots by analyzing the deviation in response from that of honeypots. The authors considered open-source honeypots that emulate Telnet, SSH, and HTTP protocols.

A first approach towards the detection of IoT honeypots was proposed by Surnin et al. [42]. The authors detect honeypots that emulate SSH and Telnet protocols by performing multiple checks through tests that determine if the target is a honeypot.

Based on the results of each test, the authors assign a probability for the target. In this paper, we also use static banners sent by known IoT honeypots to detect and filter them from our scan results. For this, we extend our previous work on honeypot fingerprinting [43].

3   Methodology

This section describes the methodology for unveiling vulnerable devices and the attack trends.

3.1   Detection of misconfigured IoT-devices

We follow two approaches for the detection of misconfigured IoT devices that are exposed to the Internet. *First*, we perform Internet-wide scans for six protocols. In particular, *MQTT, CoAP, AMQP, XMPP* and *UPnP* are chosen on the basis of their adoption and usage in IoT [44]. In addition, *Telnet* is selected as it has been significantly targeted by malware in the past [45–47]. We subsequently examine the received banners for known vulnerabilities and misconfigurations, e.g. accepting the authentication in plain text. *Second*, we use the available and open network datasets to search for vulnerable devices.

Internet-wide scanning:

In this approach, we scan the Internet for six protocols (Telnet, MQTT, CoAP, AMQP, XMPP, and UPnP). We utilize ZMap [48] along with ZGrab [49] to capture the banners of the responding hosts for further analysis. We use one of the servers running Ubuntu 20.04-LTS OS with a fixed static IP address in our lab as the scanning host. For the scan of UDP protocols like CoAP and UPnP, we used custom scripts that requested a response from the target host. For example, the UDP scan for CoAP protocol included the query "/.well-known/core" in the scan request. Note that CoAP responds to all requests if there is no authentication configured. Similarly, for UPnP, we send an "ssdp:discover" request. The scans for all the six protocols were completed in a week between March 1-5 2021 (see Table G.9 in the Appendix for the specific scan dates for each protocol). The information retrieved from the scans, such as IP address, port, response, banner, were stored in a database for further analysis to identify the vulnerable hosts. The scans followed the default blocklist provided by ZMap [50] and the European blocklist from the FireHOL Project [51]. We discuss the ethical aspects of scanning in Appendix Section A.3.

Open datasets:

Open datasets of Internet-wide scans are provided by projects like Project Sonar from Rapid7 [52] and Shodan [21]. These datasets contain essential information like IP address, port, protocol, headers, and banner information of the host with the open ports identified through the scan. We utilize the datasets from Project Sonar and Shodan to search for misconfigured IoT devices in Telnet, MQTT, CoAP, AMQP, XMPP, and UPnP. The information from the datasets assists us in verifying the results obtained from our scans. The aforementioned datasets vary by scan frequency, and hence we correlate the results identified in all the datasets.

Identifying misconfigured hosts:

The protocols considered in our work involve both TCP and UDP protocols. We consider vulnerabilities associated with the misconfiguration of protocols in IoT devices. We focus on devices that prominently lack any authentication, authorization, and encryption configurations. Furthermore, we derive that many devices with default configurations also use default parameters for authentication. To identify vulnerable hosts from the scan data obtained from the above approach, we classify our methodology into two: *Banner-based* and *Response-Based.*

Banner-based (TCP): This approach involves the analysis of the banners received on a successful connection with the target host. Banner grabbing is a technique that is used to retrieve more information from the target host. The information in the banners may help know the type, version, username, and even the session-related metadata. Based on the scanned protocol, the banners vary in the information sent. We use the ZGrab tool in our scan to fetch the banner information from the connected target. This approach is followed for the Telnet, MQTT, AMQP, and XMPP protocols. In Table G.2 we list sample banners that indicate misconfiguration of the protocol on the target device and are explained below.

- **Telnet:** We examine the banners received from the Telnet scan. The scan tries to establish a session with the target host to discover an open Telnet port, either 23 or 2323. Upon connecting, the target host sends a banner to our scanning host with basic server information. While the Telnet protocol itself can be exploited for active banner grabbing, we instead use our ZMap Telnet scan probe to get essential information on the target host. The banners received from the hosts provide us with information like the protocol, server, version, and some headers. We examine the banners received for established connections with unauthenticated console access. In case of finding certain characters like "$", "root@xxx:∼$" and "admin@xxx:∼$" in the response banners, we infer that the target hosts accept unauthenticated connections.

Protocol	Banner Response Indicator	Misconfiguration
Telnet	$	No auth, console access
Telnet	root@xxx:∼$	No auth, root console access
Telnet	admin@xxx:∼$	No auth, root console access
MQTT	MQTT Connection Code:0	Connection Accepted with no auth
AMQP	Version: 2.7.1	No auth
AMQP	Version: 2.8.4	No auth
XMPP	MECHANISM<PLAIN>	No encryption
XMPP	MECHANISM<ANONYMOUS>	No auth

Table G.2: Misconfiguration indicators: TCP protocols

- **MQTT:** The MQTT (Message Queuing Telemetry Transport) protocol scan investigates the possibility of connecting to port 1883 without any authentication. The banner received upon connection establishment with a target host provides information about supported authentication methods or connects to the target directly. After a successful connection, all the topics and channels on the target host are listed. We examine the received banners for "MQTT Connection Code:0" which specifies unauthenticated access to MQTT servers.

- **AMQP:** The AMQP (Advanced Message Queuing Protocol) scan involves scanning the Internet for port 5672. The probe retrieves metadata from the target host like version, product, and the supported authentication mechanisms on connection. The AMQP protocol has many open-source implementations like RabbitMQ [53], Apache Qpid [54] and Apache ActiveMQ [55]. We refer to the CVE [56] and NIST NVD [57] database to search for known vulnerable versions of the protocol used in the devices detected from our scan. The findings are listed in Section 4.

- **XMPP:** The XMPP protocol (Extensible Messaging and Presence Protocol) is widely used in IoT devices for message passing and communication. The XMPP protocol is scanned for both client (5222) and server ports (5269). We primarily scan for devices that support non-TLS connections on these ports. Then, we examine the banners received from the hosts for known vulnerabilities and misconfigurations, like accepting the authentication in plain text. Furthermore, as XMPP supports anonymous logins, it is possible to establish connections with the servers without any authentication. The banner provides information like version, features and supported authentication-mechanisms. The information received from the banners is used to determine the potential vulnerabilities on the device.

Response-based (UDP): The protocols using UDP as the transport layer do not respond with banners and therefore have to be explicitly queried for any information on

the service. We target two UDP-based protocols, namely CoAP and UPnP, employed in
IoT devices on the Internet to search for any misconfigurations and known vulnerabilities. We use the ZMap tool to scan for open CoAP and UPnP ports. The methodology
followed for each of the protocols is described below.

Protocol	Response	Misconfiguration
CoAP	x1C	Full Access
CoAP	220	Connected Session
CoAP	220-Admin	Admin access connection
CoAP	CoAP Resources	Resource Disclosure
UPnP	upnp:rootdevice USN: uuid:5a34308c-1a2c-4546 -ac5d-7663dd01dca1::upnp:rootdevice EXT: SERVER: Ubuntu/lucid UPnP/1.0 MiniUPnPd/1.4 LOCATION: http://192.168.0.1:16537/rootDesc.xml	Resource Disclosure

Table G.3: Misconfiguration indicators: UDP protocols

- **CoAP:** The CoAP (Constrained Application Protocol) is a web-based transfer
  protocol used in constrained environments like IoT devices for machine-to-machine
  communication. CoAP supports multicast and uses UDP as the transport layer.
  We scan the Internet for CoAP port 5683 and query the end systems for "/.well-known/core". The query triggers a response from the host, based on the configuration set by the administrators. Since CoAP can easily translate to HTTP, it
  responds with responses like "x1C" that indicate full access to the target system.
  Table G.3 summarizes some of the responses received from misconfigured devices
  and their misconfiguration details. The sample responses listed in the table show
  the indicators in the response that denote a specific misconfiguration. However,
  having systems with CoAP exposed to the Internet itself is a vulnerability and
  can be recruited for DoS amplification attacks [58].

- **UPnP and SSDP:** The UPnP (Universal Plug and Play) protocol enables device
  discovery and control in a network. Internet providers use UPnP forwarding on
  routers to deploy network configuration. The UPnP protocol uses SSDP (Simple
  Service Discovery Protocol) for the advertisement and discovery of devices on
  a network. SSDP has been used extensively in smart-home and industrial IoT
  environments for automation and control of IoT devices. We scan the Internet for
  devices with SSDP enabled on port 1900 and trigger a response to a query. Table
  G.3 shows a sample response obtained from a device exposed to the Internet and
  SSDP enabled. The devices exposed to the Internet could be recruited by malware
  or botmasters or adversaries for DDoS attacks [59].

The banners and the responses received from active scanning and querying are stored in a database to perform further analysis. Furthermore, we analyze the responses for known high-severity vulnerabilities from the CVE database. The results are correlated with the open datasets analyzed from Subsection 3.1. We find a total of $1,832,893$ unique, vulnerable hosts exposed to the Internet and present our findings and analysis in the results section.

3.2   IoT-Honeypot Fingerprinting

From our Internet-scanning methodology, we expect that some of the misconfigured devices may be honeypots and can poison our result dataset. Thus, we perform honeypot fingerprinting to identify honeypots in our dataset and filter them. Honeypots are widely used deception-based network monitoring systems that proactively detect attacks. They work by simulating protocols and services on the target system and classified based on their simulation levels into low, medium, and high interaction. We filter honeypots from our scan results by following banner-based honeypot fingerprinting. This technique is adapted from existing research methodology proposed by Morishita et al. and Vetterl et al. [41, 60] and is extended to detect IoT-based honeypots.

Honeypot fingerprinting is the technique used to determine if a vulnerable target system is a honeypot [41, 42, 60]. This may assist honeypot developers improve the simulation capabilities, or help adversaries evade honeypots. The techniques are based on banners, response-deviation, static content, lack of simulation, and interaction capabilities. We leverage our previous work on multistage honeypot fingerprinting that is based on banners and responses received from the honeypots [43]. The framework performs sequential checks based on the services discovered on the target host and the response received is analyzed to determine if the target is a honeypot. We deploy open-source and widely used honeypots in our lab to determine the unique characteristics that differentiate them from existing systems. These characteristics can be static banners, response, or content. For the purposes of this paper we only attempt fingerprinting for honeypots emulating Telnet. These include the HoneyPy [61], Cowrie [62], MT-Pot [63], Telnet IoT honeypot [64], Conpot [28], Kippo [65], Kako [66], Hontel [67] and Anglerfish [34] honeypots.

3.3   IoT Honeypot Deployment

The scans from our methodology reveal a large number of misconfigured devices. To determine the potential attack vectors and to study the attack trends, one of the obvious ways is to deploy honeypots. Honeypots have been a valuable resource for studying the attack trends. We choose open-source honeypots and deploy them in our lab setup, where they are configured to face the Internet without any firewall (see Appendix Section A.3 for details about how we ensured that our honeypots were not used for malicious purposes). The network traffic gathered on all these honeypots is analyzed to understand

the attack trends. We describe the IoT honeypots and their deployment in the following subsections.

IoT Honeypots

We choose Cowrie [62], HosTaGe [30], Dionaea [29], ThingPot [17], U-Pot [18], and Conpot [28] honeypots in our methodology as we find these honeypots relevant to our study based on emulated protocols and because they are open source and widely used [41, 60]. Furthermore, these honeypots are capable of simulating IoT-based device profiles. For example, the HosTaGe honeypot can simulate a CoAP-based smoke sensor or, an Arduino board running IoT protocols. The protocols emulated by these honeypots are listed in Table G.1.

Deployment Setup

The honeypots are deployed in our lab environment with an unfiltered network. Moreover, the honeypots are grouped based on the emulated protocols as shown in Figure G.1. By grouping them in this way, we ensure no overlap of the protocols emulated by the honeypots. Each group is assigned a public IP address with port-forwarding enabled on the routers. This way, the honeypots are independent of their network and are exposed to the Internet. All the honeypots, except HosTaGe, run as containers on a system with Ubuntu 18.04 LTS Server. The HosTaGe honeypot is deployed on a rooted Samsung S10 Galaxy device to emulate services on ports below 1024. All the attacks gathered on the honeypots are exported daily and imported into the database. We record the attacks on all the honeypots for one month in April 2021 on a day to day basis. The findings are summarized in the Section 4.

3.4 Network-Telescope Analysis

The honeypots deployed in our lab environment provide us with traffic on a limited IP address space. To address this limitation and get a more holistic view of the attack landscape, we analyze the FlowTuple data from a network telescope. A network telescope is a portion of routed IP address space in which no legitimate traffic exists [68]. Telescopes contain massive data that is captured across large number of routed IP address space. This data helps us to understand the attack landscape across the large network, in addition to the traffic we receive on our honeypots. An analysis of the traffic received on the telescope provides information about the remote network events such as flooding DoS attacks, infection of hosts by Internet worms, and network scanning [39]. Studying these networking events assists us in further understanding the latest scanning and attack trends employed by adversaries. In addition to the data from the honeypots, we analyze the data from the CAIDA UCSD Network-Telescope scanners dataset [68].
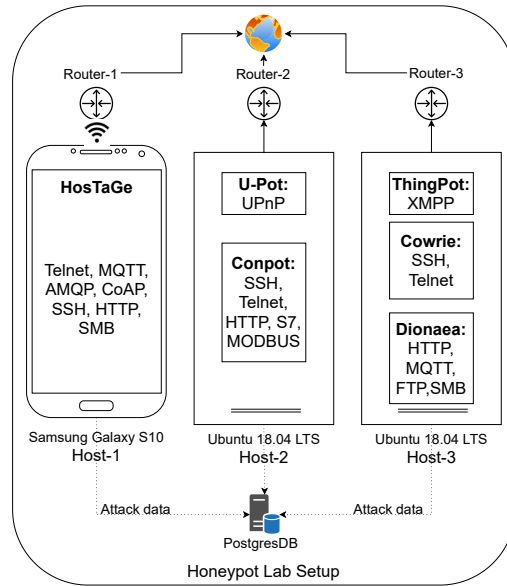
Fig. G.1: Honeypot experimental setup

The UCSD network telescope consists of a globally routed /8 network that carries almost no legitimate traffic. The captured data provides us with a snapshot of anomalous 'background' traffic to 1/256th of all public IPv4 destination addresses on the Internet. Unlike honeypots, telescopes do not simulate any protocols and hence do not respond to any requests. A significant part of the addresses are unused, and any traffic on this network is potentially suspicious.

The traffic to CAIDA UCSD Network Telescope is captured and offered in three forms; *FlowTuple* data, *Raw pcap* data, and *Aggregated Daily RSDoS Attack Metadata.* The FlowTuple data is captured hourly and consists of elementary information about the suspicious traffic. The information includes source and destination IP address, ports, timestamp, protocol, TTL, TCP flags, IP packet length, TCP-SYN packet length, TCP-SYN window length, packet count, country code, and ASN information [69]. Furthermore, additional metadata like *is_spoofed* and *is_masscan* provide information if the source IP address may be spoofed and if the Masscan tool [22] is used for the scan. The files are stored on a minute basis, and hence there are 1, 440 files generated per day. We use the FlowTuple data provided by CAIDA and parse the records for April 2021 and requests targeting the Telnet, AMQP, MQTT, XMPP, CoAP, and UPnP protocols. Furthermore, we analyze and classify the suspicious sources into scanning and malicious traffic based on the results we obtain from our honeypot deployment and the ground truth from threat intelligence repositories GreyNoise [70], and Virustotal [71].

4 Results

This section presents our findings primarily on misconfigured devices on the Internet and the attack trends observed through our honeypots. The section is divided into the results obtained through the Internet-wide scan, honeypot detection and the observations from the deployed honeypots.

4.1 Results from Internet-wide scanning

Exposed devices

Upon scanning the Internet with ZMap [48] for six protocols namely Telnet, MQTT, AMQP, XMPP, UPnP and CoAP, we find a total of 14 million hosts with open ports. We compare our scan results with the Project Sonar [52] Internet-wide scan dataset and Shodan [21]. The total number of unique hosts exposed to the Internet by the protocol identified through our scan is listed in Table G.4. The Project Sonar does not provide datasets for AMQP and XMPP protocols.

Protocol	ZMap Scan	Project Sonar	Shodan
AMQP	34,542	NA	18,701
XMPP	423,867	NA	315,861
CoAP	618,650	438,098	590,740
UPnP	1,381,940	395,331	433,571
MQTT	4,842,465	3,921,585	162,216
Telnet	7,096,465	6,004,956	188,291
Total	14,397,929 (14M)	10,759,970 (10M)	1,709,380 (1M)

Table G.4: #Exposed systems on the Internet by protocol and source

The number of hosts listed from Project Sonar and Shodan was from the same period when our scans were performed. The total number of exposed hosts detected by our scan is higher than the Project Sonar dataset and Shodan. We argue that this could be because of possible allow-listing performed by these scanning services. Another reason could be that our methodology involves scanning the Internet for multiple ports for one protocol. For example, we perform scans with both ports 23 and 2323 for the Telnet protocol, while Project Sonar performs the scans only with port 23. This leads to having a higher number of detected hosts.

Exposed Device Types

From Table G.4, we observe that the number of devices exposing Telnet (7M) is higher than the other protocols. Telnet is highly targeted by botnets to infect with malware.
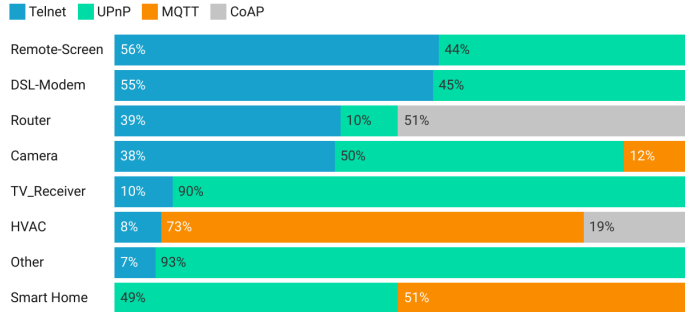
Fig. G.2: Top IoT device types by protocol (%)

From the banners and the responses received, we attempt to detect the device type. The device types are identified by matching specific text from the banners and the response. For example, the HiKVision Network Camera responds with a banner *"192.0.0.64 login:"* for Telnet connections. The IP address is assigned to the camera as a default configuration and hence responds with this banner [72]. We discover many device types upon performing a similar approach to find consistent banner and response patterns across the scan results. We use the results obtained from the scanning of the protocols to identify device types. We list the major device types and the protocols on which they were detected in Figure G.2. We observe that most of the device types are identified through the Telnet and the UPnP responses. The IoT devices were identified with responses from the Telnet, UPnP, MQTT and CoAP protocols. The response received from XMPP and AMQP services were not sufficient to label the target as an IoT device. The basis on which the device types are identified is listed in Appendix-Table G.11 for every protocol. Furthermore, other device types like NAS, micro 3D printers and so on are also listed. To facilitate automated detection, we leverage ZTag [73], a tool for annotation of raw data with additional metadata that facilitates tagging and automation of the data from our scans. The banners and static responses are used as metadata for tagging the device types.

Misconfigured Devices

We consider the misconfigurations for the protocols listed on Tables G.2 and G.3 for identifying the vulnerable devices. A misconfigured device is a device with no authentication, no encryption, or no authorization configured. We analyze the response received from the scans of all the protocols and find a total of 1, 832, 893 misconfigured devices that satisfy at least one of the conditions. The number of misconfigured devices identified by the protocol are listed in Table G.5. The table shows the vulnerability identified in each of the protocols scanned and analyzed by us. In TCP protocols, we see that there

Protocol	Vulnerability	#Devices found
CoAP	No auth, admin access	427
AMQP	No auth	2,731
Telnet	No auth	4,013
XMPP	No encryption	5,421
CoAP	No auth	9,067
Telnet	No auth, root access	22,887
MQTT	No auth	102,891
XMPP	Anonymous login	143,986
CoAP	Reflection-attack resource	543,341
UPnP	Reflection-attack resource	998,129
Total		**1,832,893**

Table G.5: Total misconfigured devices per protocol

are devices exposed with no authentication configured. This means that with a simple connection request, the adversary could connect to the device. There is also a lack of authorization configured in devices that allow the end systems to respond to queries from unknown hosts. Furthermore, we detect many UDP-based devices that respond to discovery queries and can be leveraged in denial of service attacks. We further discuss this type of attack in Section 5. Table G.10 in the appendix lists the number of misconfigured devices distributed by country on the six protocols. The source location of the attacks are determined by using the ipgeolocation database [74]. We observe a large number of countries including USA (27%), China (13%), Russia (9.1%), Taiwan (8.9%), Germany (7.8%), Philippines(6.2%), UK(5.8%), Brazil (3.3%), India (3.2%), Thailand (2.7%) , Hong Kong (2.7%), South Korea (2.5%), Israel (2.1%), Canada (1.9%), Bangladesh (1.1%), France (0.9%), Japan (0.7%), and other (1.3%).

4.2 Honeypot Detection

The misconfigured devices identified from our methodology could contain honeypots that can lead to poisoned results. We use the honeypot detection approach, described in Subsection 3.2, to filter out the honeypots from our results. To fingerprint honeypots, we initially perform a search for open-source and research-based IoT-based honeypots. We deploy these honeypots in our lab and capture the banners obtained through a Telnet session from the ZMap client. Then, we systematically search the responses received from our scanning process to filter the honeypot instances. Table G.6 lists the honeypots detected using the Telnet banners and the response identified from honeypots *. Overall, with this approach we were able to detect a total of 8,192 honeypots. The

---

*The Anglerfish honeypot is not open-source, but was detected retrospectively as a result of large number of suspicious static banners observed in the scan results.

results are validated on the basis of our previous work on honeypot fingerprinting [43].

Honeypot	Telnet Banner	#Detected Instances
HoneyPy	Debian GNU/Linux 7\r\r\nLogin:	27
Cowrie	\xff\xfd\x1flogin:	3,228
MTPot	\xff\xfb\x03\xff\xfb\x01\xff\xfd\x1f\ xff\xfd\x18\r\nlogin:	194
Telnet IoT Honeypot	\xff\xfd\x01Login: Password: \r\nWelcome to EmbyLinux 3\.13\.0-24-generic\r\n #	211
Conpot	Connected to [00:13:EA:00:00:0]	216
Kippo	SSH-2.0-OpenSSH_5.1p1 Debian-5	47
Kako	BusyBox v1.19.3 (2013-11-01 10:10:26 CST)	16
Hontel	BusyBox v1.18.4 (2012-04-17 18:58:31 CST)	12
Anglerfish	[root@LocalHost tmp]$	4,241
Total		8,192

Table G.6: Detected honeypots through Telnet banner signatures

4.3 Attack trends from honeypots and network telescope
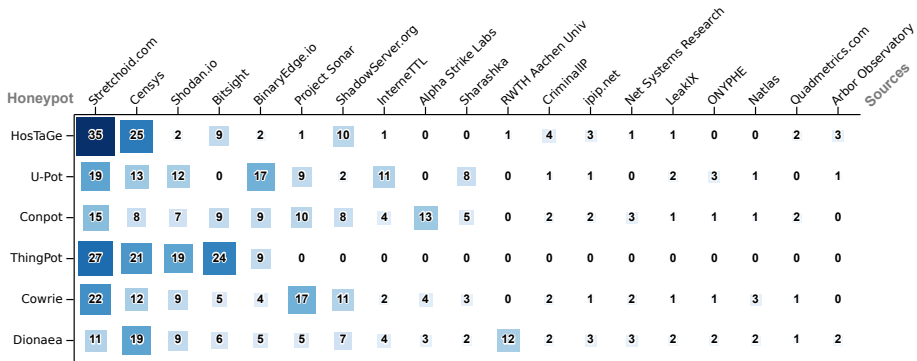
Fig. G.3: Scanning-service traffic on honeypots (%)

Honeypots

We deploy six honeypots as depicted in Figure G.1 at our lab environment. The total number of attack events detected by each honeypot by protocol over one month is listed in Table G.7. We observe a total of 200, 209 attack events from all the honeypots. Even

though any interaction with honeypots is considered an attack, we argue that recurring scans from known sources (e.g. Shodan [21]) can be considered benign traffic. The attack events consist of both benign and malicious traffic. Scanning-service traffic involves internet-wide scanning events from known sources like Shodan [21], Censys [75], Project Sonar [52], BinaryEdge [76], ZoomEye [77], Fofa [78] and educational organizations like RWTH Aachen University [79]. Malicious traffic involves attacks from unknown scanning sources or attacks with malicious payloads. The packets include both scanning probes and malicious payloads.
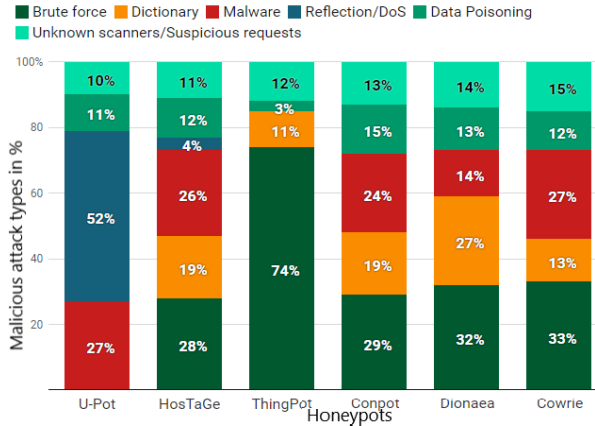
Fig. G.4: Attack types in different honeypots (%)

Scanning service traffic We perform a reverse lookup of the source IP addresses of the suspicious traffic received on the honeypots. We identify a total of 10,696 unique IP addresses that are registered to known scanning services shown in Figure G.3. Table G.7 lists the total unique IP addresses registered to scanning services, detected per honeypot. Figure G.3 shows the scanning-services received on each honeypot. It lists the percentage of total scanning traffic distributed between the identified services. The suspicious traffic that does not resolve to the scanning-services is classified as unknown and is not included as a scanning service. Furthermore, we observe that the IPs from the scanning services scan the Internet periodically and thus are recurring, unlike suspicious one-time scans. The prominent scanning services identified are Stretchoid.com [80], Censys, Shodan, Bitsight [81], BinaryEdge [76], Project Sonar [52], Shadow Server [82], Interne TTL [83], Alpha Strike Labs [84], Sharashka [85], RWTH Aachen University [79], CriminalIP [86], ipip.net [87], Net Systems Research [88], LeakIX [89], ONYPHE [90], Natlas [91], Quadmetrics.com [92] and Arbor Observatory [93].

Honeypot	Simulated Device Profile	Protocol	#Attack events	Scanning service*	Malicious*	Unknown/ Suspicious*
HosTaGe	Arduino Board with IoT Protocols	Telnet MQTT AMQP CoAP SSH HTTP SMB	19,733 2,511 2,780 11,543 19,174 16,192 1,830	2,866	21,189	2,347
U-Pot	Belkin Wemo smart switch	UPnP	17,101	1,121	7,814	1,786
Conpot	Siemens S7 PLC	SSH Telnet S7 HTTP	12,837 12,377 7,113 11,313	1,678	11,765	1,876
ThingPot	Philips Hue Bridge	XMPP	11,344	967	2,172	963
Cowrie	SSH Server with IoT banner	SSH Telnet	15,459 14,963	2,111	12,874	1,113
Dionaea	Arduino IoT device with frontend	HTTP MQTT FTP SMB	11,974 1,557 3,565 6,873	1,953	13,876	1,694
Total			**200,209**	**10,696**	**69,690**	**9,779**

Table G.7: Total attack events by type and protocol on honeypots (* unique source IPs)

Malicious traffic   Since honeypots have no production value, all traffic that is not coming from a known scanning service is considered malicious. These interactions include brute-force attempts, dictionary attacks, malware droppers. Besides, the traffic that does not match the scanning attributes of known scanning tools is malicious. The malware classification is based on the received payloads. The requests are examined for port scans from recognized scanning tools like ZMap. Furthermore, we classify the source as malicious upon receiving recurring requests with malicious payloads. Figure G.4 shows the malicious requests received per honeypot and type. We also observe reflection attack attempts on the CoAP and UPnP protocols. The malware attacks listed in Table G.7 were classified based on the requested content. The requests included URLs used for downloading the malware and messages with the malicious payload. We also observed data poisoning attacks on the honeypots. For example, there were CoAP requests that changed the data by publishing messages. The malware are identified by analysis of the pcap files stored on the honeypots for unusual content. Upon finding any unusual content, for example a file or script in the payload, we check the file with VirusTotal. Regarding poisoning attacks, we observe if the data has been modified or deleted from the services simulated by the honeypots. For example, we check for any modifications attempted on the data in the MQTT queues. We further discuss some interesting cases in Section 5. The honeypots further encountered non-recurring scanning traffic from unknown sources and suspicious requests that were not identical to

any known attack types. Such type of suspicious traffic is grouped under the unknown
scanners or suspicious requests.

Network-Telescope:

The UCSD CAIDA network telescope consists of 16 million IP addresses. Upon parsing
the FlowTuple dataset captured from the telescope, we observe an average of 78 billion
requests per day. An average of 2.7 billion is targeted towards the Telnet, MQTT,
AMQP, CoAP, XMPP, and UPnP protocols. Table G.8 shows the average number of
suspicious requests received on each protocol daily and the number of IPs that belonged
to scanning-services and unknown scanners. We observe that the Telnet protocol dom-
inates the number of suspicious traffic in comparison to the other protocols. This could
be because of the presence of many systems infected with malware like Mirai that con-
stantly scan for vulnerable systems on the Internet. For deeper analysis into the attack
sources, we check the source IPs to known scanning services and classify them into
known and suspicious sources. Table G.8 lists the number of known scanning-services
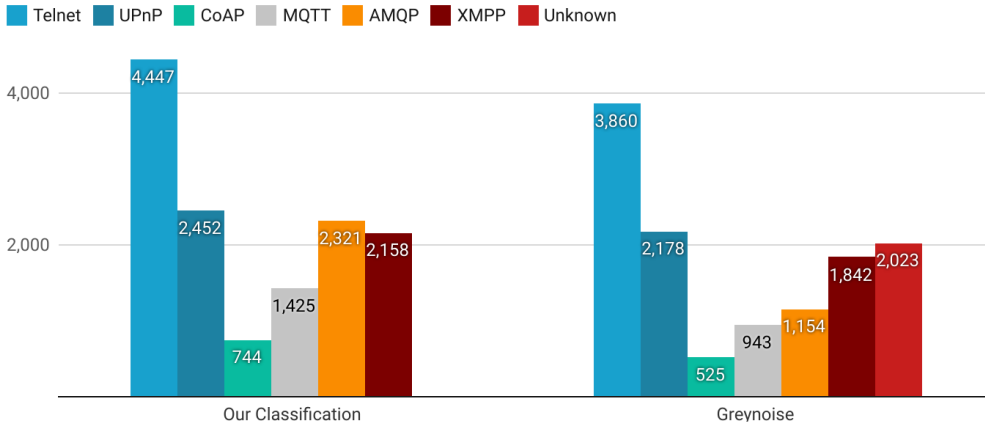and the unknown suspicious scans.

Protocol	Daily Avg. Count	Unique IP	Scanning-service	Unknown/Suspicious
Telnet	2,554,585,920	85,615,200	4,142	85,611,058
UPnP	131,794,560	1,8633	2,279	16,354
CoAP	68,353,920	2,342	627	1,715
MQTT	17,072,640	5,572	1,248	4,324
AMQP	13,907,520	7,132	2,256	4,876
XMPP	6,429,600	4,255	1,973	2,282
Total	**2.7 Bil.**	**85.6 Mil.**	**12525**	**85.6 Mil.**

Table G.8: Telescope suspicious traffic classification

Suspicious traffic classification

We validate our findings on classification of attack sources i.e. scanning services and
malicious with [70], and Virustotal [71] databases. Greynoise offers a classification of
the attack sources observed on its honeypots into benign, malicious and unknown. The
unique source IP addresses of the traffic received on the honeypots and the telescope are
searched and corroborated with the classification from Greynoise database. Figure G.5
shows the comparison between the total number of attack sources classified as scanning
service by our classification and Greynoise. We find that a majority of the sources were
identified to be from scanning services by both our method and Greynoise, however,
there were 2,023 IP addresses that were not identified by Greynoise. We also observe
that the number of scanning services detected by our method is higher for the AMQP,
Telnet and MQTT protocols, which is because we received traffic from multiple cyber-
security risk rating platforms. We suspect that these scans were limited to the European

continent or were country-specific.

Fig. G.5: Classification of scanning-services

The source IP addresses are further examined with the VirusTotal threat database. We perform a search of the IP addresses from unknown suspicious requests received on the honeypots and the telescope. Upon performing a search for an IP address, Virus-Total provides a positive score attribute that indicates the number of security vendors that have flagged them as malicious. Note that we consider the IP to be a malicious actor if there is at least one security vendor to label them as malicious (VirusTotal has other labels like phishing). The results are summarized in Figure G.6 that lists the percentage of IPs indicated as malicious by protocol as classified by Virustotal. The protocols from the honeypot are indicated by (H) and the telescope as (T). The details about specific malware detected in the traffic are elaborated in Section 5. We observe that the attack sources of the SMB from the honeypots have the highest classification of malicious actors. This is because many well known malware propagate via SMB and hence the detected numbers are higher.

5   Discussion

This section summarizes the attack trends observed from analyzing the attacks on honeypots and the suspicious traffic from the network telescope. We then discuss the impact of listing vulnerable honeypot hosts by scanning services like Shodan. Finally, we investigate the attacks observed from infected hosts and the multistage attacks on honeypots.
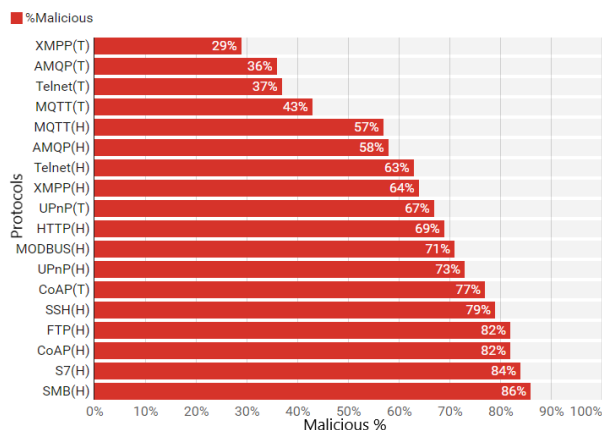
Fig. G.6: Malware classification by Virustotal (%)

5.1 Attack trends by protocol

In the following, we provide an overview of the attack trends on the protocols simulated by the honeypots. In addition to the logs, the network traffic is captured with *tcpdump* on the hosts where the honeypots are deployed and the pcap files are further analyzed to determine the attack vectors. Moreover, we discuss the findings from the analysis of the pcap files from the honeypots by protocol.

Telnet and SSH Attacks

The Telnet protocol (simulated by HosTaGe, Conpot, and Cowrie honeypots) received the highest number of attacks, with a total of $47,073$ attacks, of which $12,709$ were the result of known scanning services. The remaining suspicious traffic received can be further categorized into scans from unknown scanning actors and malware. We examine the pcap files with the Virustotal database for signs of malware signatures and discover 113 Mirai variants. The hashes of the malware identified are listed in Appendix Table G.13. Upon tracing the sources of the malware, we discovered that one of the sources had a valid domain registration as a website for a restaurant in the UK. Beyond Mirai variants, we identified *BrickerBot.2, BrickerBot.1, Hehbot* and *Luabot* malware that brute-force into a target with default credentials. The Appendix Table G.12 lists the default most used credentials that were recorded for Telnet and SSH. Moreover, we observe a large number of brute-force attacks with default passwords targeting routers and modems.

The SSH protocol was simulated by HosTaGe, Conpot, and Cowrie honeypots. We observe a high number of brute-force and dictionary attacks on all honeypots. The honeypots received many recent crypto-mining malware like LemonDuck and FritzFrog,

among other prominent malware variants. The hash of the malware samples is listed in Appendix Table G.13.

MQTT, AMQP and XMPP Attacks

The MQTT protocol was simulated by the HosTaGe and Dionaea honeypots. The attacks mainly aimed at accessing and changing data in the topics. A majority of the attacks tried to access the *'$SYS'* topics. Some attacks tried to poison the data in the topics available while others subscribed to receive messages from specific topics.

The AMQP protocol, simulated by HosTaGe, received similar attacks to that of the MQTT protocol. The adversaries aimed at poisoning the data in the queue through publishing data and subscribing to receive new messages. We also observed a large number of messages published by the adversaries, causing a flood leading to a Denial Of Service.

The XMPP protocol, simulated by the ThingPot honeypot, received brute-force attacks where the adversaries tried to log in to the Philips Hue Bridge system. In addition, we detected some dictionary attacks on the protocol. Lastly, we recorded attempts from malware trying to log in as anonymous users to change the configured state of the lights on the device. We speculate that the malware was trying to examine their write privileges.

CoAP and UPnP attacks

The primary attacks on the CoAP protocol, simulated by HosTaGe, involved discovery requests. However, after the reconnaissance, we observed returning threat actors, especially after being listed on scanning engines like Shodan and Binary Edge (see also Section 5.2). The number of attacks increased, followed by poisoning attacks. Moreover, we detected flooding attacks from unknown malicious actors which resulted in a DoS attack against the honeypot. We observed that the flooding attacks originated from two different sources at the same time. A reverse lookup of the IP addresses showed the existence of duplicate DNS entries for both the IP addresses, which leads to the possibility of reflection or amplification attacks. The webpages of the IPs pointed to an Apache2 Ubuntu Default Page. Other sources of the DoS attacks appeared to originate from Italy, Taiwan, and Brazil.

The U-Pot honeypot received a large number of discovery requests. Following the discovery, there were many DoS attempts recorded on the honeypot. Similar to the attacks on the CoAP protocol, the adversaries performed UDP flood attacks on the honeypot. More than 80% of the total attacks received were a part of the DoS attacks. Two of the adversaries were first observed scanning for the protocol three days before the attack with the same source IP addresses. The source was traced to have a valid domain registration and addressed to a construction service provider in Taiwan.

Modbus and S7 attacks

The Modbus and the S7 protocol, simulated by Conpot, received a large number of poisoning attacks where adversaries tried to access and change the values stored in the registers. The attacks targeted three of the nineteen available function codes for reading device identification, the holding register, and the reporting server. Only 10% of the Modbus traffic used valid function codes to access the register data. Furthermore, we observed DoS attacks from attackers that possibly targeted the ICSA-16-299-01 vulnerability for the Siemens S7 protocol [94]. The DoS was performed by flooding the requests with PDU type 1, that results in spawning of a job request in the device.

FTP and SMB attacks

The FTP protocol, simulated by Dionaea, received brute-force and dictionary attacks. In addition, a few attacks deployed malware upon successful authentication to the FTP server. We examined the binary files deployed on the FTP server with Virustotal and found positive results for malware. We discovered multiple deployments of the Mozi and the Lokibot malware. The hash of the malware from Virustotal is listed in Appendix Table G.13.

The SMB protocol, simulated by HosTaGe and Dionaea, was largely targeted with the EternalBlue, EternalRomance, and the EternalChampion exploits that attack Microsoft's implementation of the SMB protocol. Among the malware deployed, we find the WannaCry and its variants the most common on the honeypots. The hash of the malware identified via Virustotal is listed on Appendix Table G.13.

HTTP attacks

HTTP was simulated by HosTaGe, Conpot, and Dionaea. The honeypots responded with static content and a login page for the simulated device profiles. The protocol was targeted with a large number of web-scraping requests, brute-force, and dictionary attacks. In addition, we observed DoS attacks with HTTP flood packets causing the honeypots to crash. The majority of the DoS attacks came from China, Russia, Israel, USA, and Italy. The attackers also tried to exploit the HTTP protocol by injecting crypto-mining malware. Upon performing a reverse lookup of the attack sources with the Exonerator service [95] we determine a total of 151 unique IPs originating from Tor relays. Furthermore, we observe a daily recurring pattern of scans from these sources and an increasing trend over the month.

Summary

We summarize the attack trends for each protocol emulated by the honeypots for April 2021 in Figure G.7. We observe that UDP protocols (CoAP and UPnP) received higher traffic related to Denial of Service in comparison to TCP protocols. Furthermore, the
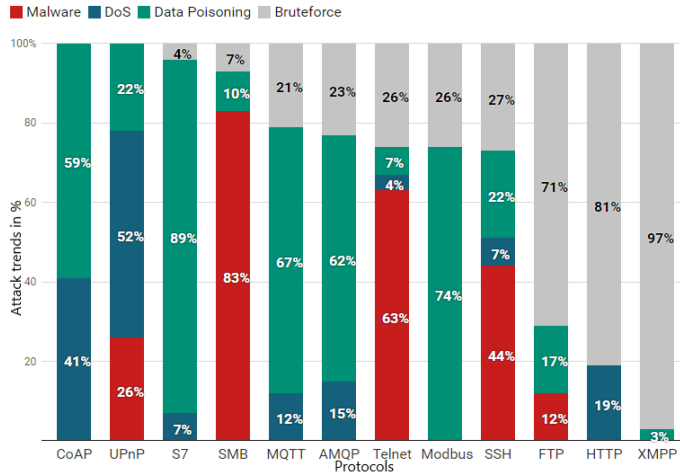
**Fig. G.7:** Attack trends by type (%) and protocol

TCP protocols have seen an increase in malware deployment and data poisoning. Our simulated IoT environment suggests that there is an increasing number of attacks concentrating on misusing misconfigured IoT devices.

5.2   Impact of listing by scanning-services

The honeypots received many requests from known scanning-services as listed in Figure G.3. We observed an increase in the number of attacks on the honeypots after their listing on scanning-services like Shodan, BinaryEdge and ZoomEye. Figure G.8 shows the total number of attacks on the honeypots by day. The attacks include all the requests from scanning-services and other malicious sources. The attacks are distinct by the connection sessions established from the source. The dates at which scanning-services listed the honeypots are also marked in the figure. Furthermore, the figure shows the days on which some major DoS attacks occurred (Day 24, 26). We observe an upward trend in the number of attacks after being listed by scanning-services.

5.3   Attacks from infected hosts

From the results of the honeypots and the network telescope, we observe that there is a large number of attacks originating from unknown sources. Furthermore, from the attack trends, we observe many attempts of malware injections from unknown sources. To determine attack sources originating from infected IoT devices, we search how many of the identified misconfigured devices (see Table G.5) are present as attack sources
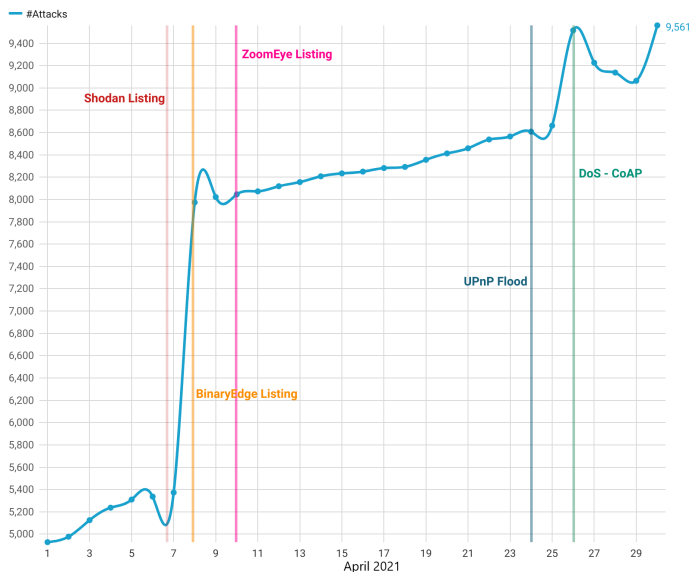
Fig. G.8: Total attacks by day. We highlight known scanning listings and interesting attack events

against our honeypots and the telescope. We identify a total of $11,118^{\dagger}$ unique IP addresses that originate from misconfigured IoT devices. Furthermore, all of the aforesaid IP addresses were flagged as malicious by at least one scanning vendor in Virustotal.

We extend the detection of infected IoT devices by searching the remaining source IP addresses in the Censys database [96]. The Censys database has a labelled dataset of IoT devices and returns an "*iot*" tag if the IP address was identified as an IoT device from its periodic Internet-wide scans. We identify an additional $1,671^{\ddagger}$ IoT devices from the Censys database. A further analysis to determine the type of these IoT devices reveals that the majority of the attacks originate from cameras, routers and IP phones.

Lastly, we extend the search for attacks from infected hosts from non-IoT devices. Upon performing a simple reverse lookup of all the source IP addresses, we discover a total of 797 registered domains of which 427 have a webpage. The domains were looked up to see if they served additional on additional IP addresses than the one discovered from our analysis. We found the domains registered with /30 and /29 subnets with some unused IP addresses. From this analysis, we also infer that some of the Telnet malware injections originated from an infected URL serving HTML. Upon searching

---

†In more details, $1,147$ attacked only the honeypots, $1,274$ attacked only the telescope, and $8,697$ both of them.

‡In particular, 439 attacked only the honeypots, 564 attacked only the telescope, and 668 both of them.

Virustotal for these URLs, we find 346 of them tagged as malicious. The webpages were found serving default wordpress sites, Ubuntu Apache test pages, static ad pages and fake online shopping portals.
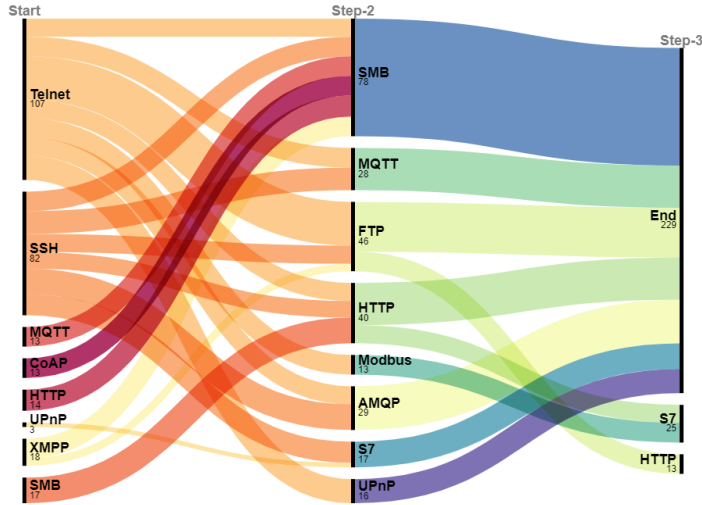

Fig. G.9: Multistage attacks detected on honeypots

5.4 Multistage attacks in honeypots

We define *multistage attacks* as attacks in which there is a pattern of multiple protocols that are being sequentially attacked by the same adversary. Attackers may employ the multistage attack strategy to amplify an attack or to find alternate sources for malware injection. Although such types of traffic may be observed from scanning services, we filter such sources by checking if they are registered to a domain affiliated to a scanning service. The HosTaGe honeypot offers the detection of multistage attacks as a service. For the other honeypots, we group the attacks from distinct source IP addresses and check if multiple protocols are targeted. We note that the attacks are grouped based on the source IP addresses and the time interval between attacks is not taken into account. This entails that a follow up attack from the same adversary may have occurred anytime in the one month experiment period.

We list the protocols targeted by attackers in the identified multistage attacks across the honeypots in Figure G.9. The figure depicts the protocols targeted step-wise. The numbers below the protocol indicate the total number of attacks received on that protocol at that stage and the thickness of the bars indicate the amplitude of the attacks. A total of 267 multistage attacks were detected and we observe that the majority of them initiated with Telnet and SSH. Furthermore, the SMB is noticed to be receiving most

of the attacks at the second step and the S7 protocol in step three.

6   Conclusion

With this work, we combine the search for misconfigured IoT devices on the Internet with an analysis of attack trends in the IoT. To the best of our knowledge, our work is the first to combine the results of a complete IPv4 scan with knowledge gained by honeypot deployment and network telescope data. Beyond the large number of attacks that we received and analyzed, we show that many of the misconfigured devices take themselves the role of the attacker as part of malware propagation campaigns.

In particular, our scans reveal that there is a large number of misconfigured IoT devices that can be leveraged to perform diverse type of attacks on the Internet. Furthermore, the attacks received on the honeypots suggest a trend in attackers searching for vulnerable IoT devices. This is supported by the network telescope data that suggest a global trend. The attacks received from infected IoT hosts show that high magnitude attacks are possible, specifically with devices running CoAP and UPnP. Through this work, we aim at creating awareness about the implications of the misconfigurations of IoT devices by exploring such devices that are making us of six popular protocols. In fact, it is worth noting that by intersecting all of our experiments (IPv4 scanning, network telescope and honeypots) we are able to identify 11,118 misconfigured IoT systems that are actively attacking the Internet; simultaneously 1.8 million devices are potentially waiting to be exploited by adversaries.

In comparison to previous work on Internet-wide scanning [20, 26], we use custom probes that scan for specific IoT protocols and further use open datasets to verify our findings from the scan. We identify a large of number of misconfigured IoT devices based on specific banner-based and response-based indicators. While Markowsky et al. [20] demonstrate how to scan and find vulnerable devices using Shodan and Masscan, they do not specifically search for misconfigured IoT devices. Our results confirm the methodology of [26], which combines scanning the Internet and deploying honeypots to study the attack trends on the FTP protocol. We instead focus on 6 protocols that are used in IoT. We enhance our methodology by using the data from a network telescope as with Neshenko st al. [25], who use the data to support their proposed taxonomy of IoT vulnerabilities. Furthermore, our work highlights the need for sanitization of Internet-scan data from honeypots. In this context, we identify 8,192 honeypots that would otherwise be classified as misconfigured IoT systems. While individual work on honeypot fingerprinting has shed light into this field [41, 43], no previous work on the Internet measurements has taken honeypots into account.

To summarize our contributions, we scan the Internet, specifically to find misconfigured IoT devices by the use of custom probes on 6 protocols (TCP and UDP). We verify the results from our scan by validating them with open datasets on Internet-scanning. We filter out potential honeypots from our scanning results by using our multistage

honeypot fingerprinting techniques [43] to avoid poisoning of the results. Lastly, we deploy 6 IoT honeypots that emulate misconfigurations observed from IoT devices in our scan. Furthermore, the analysis of the data from the telescope compliments our observations on the attacks received on honeypots.

With regard to future work, we plan to extend the scanning scope of protocols to include TR069, SMB, and industrial IoT protocols like DDS and OPC UA. The analysis from the network telescope also motivates us to perform a deeper analysis on raw packet data to uncover new threat actors on Industrial IoT devices and protocols. Lastly, based on the recent work of Wan et al. [97] we see the need for combining geographically distributed scanners, especially for certain protocols (e.g. SSH).

A   Appendix

A.1   Scanning dates by protocol

The Internet-wide scans on all the 6 protocols were performed in a span of one week. Table G.9 lists the dates on which the scans were started for the corresponding protocols.

Protocol	Scan Date
CoAP	1 March 2021
UPnP	2 March 2021
Telnet	2 March 2021
MQTT	4 March 2021
AMQP	4 March 2021
XMPP	5 March 2021

Table G.9: Scan dates per protocol

A.2   Misconfigured IoT devices by country

Based on our scan, we detect a total of $1,832,893$ misconfigured devices over the response received from six protocols. Table G.10 lists the distribution of misconfigured devices by country.

Country	Count
USA	494,881 (27%)
China	238,276 (13%)
Russia	166,793 (9.1%)
Taiwan	163,127 (8.9%)
Germany	142,966 (7.8%)
Philippines	113,639 (6.2%)
UK	106,308 (5.8%)
Brazil	60,485 (3.3%)
India	58,653 (3.2%)
Thailand	49,488 (2.7%)
Hong Kong	45,822 (2.5%)
South Korea	45,822 (2.5%)
Israel	38,491 (2.1%)
Canada	34,825 (1.9%)
Other countries	23,828 (1.3%)
Bangladesh	20,162 (1.1%)
France	16,496 (0.9%)
Japan	12,830 (0.7%)
Total	**1,832,893**

Table G.10: Misconfigured devices by country

A.3 Ethical Considerations

The Internet-wide scans were performed through a set of dedicated IP address provided in the University network. The motivation to perform our own scans with ZMap is because some networks blocklist Shodan, Censys and other scanning services. However, we wanted to include datasets from scanning-services to cover the networks that may have blocked our scanning IP. Furthermore, because of the CoAP and the UPnP protocols in our scanning portfolio, we ran custom scripts to fetch specific response from the hosts that helps us in identification of misconfigured devices. Moreover a recent study shows the impact of location on Internet-wide scans, which presents certain limitations of scanning services [98]. We were motivated by this study to perform our own scans.

Information regarding the misconfigured devices, like the source IP addresses are not shared or disclosed. The data will be stored for a period of three months from the date of collection, followed by anonymization of IP addresses to follow the local privacy regulations. Furthermore, a webpage stating the purpose of the scan and research was setup to ensure transparency and indicate intent of the scanning process. The scans included a default blocklist from the ZMap repository [50] and the European blocklisted provided by the FireHOL project [51].

The samples identified by Virustotal as malware will be shared on online threat repositories like Malpedia [99] and Malware Bazaar [100] to facilitate research from the open source community. Lastly, the destination IP addresses in the UCSD CAIDA network telescope have not been disclosed or shared and are anonymized in our database to facilitate the purposes of the telescope.

Honeypot sandboxing

We want to emphasize that our setting focused only on collecting attacks from the Internet and in principle did not allow for honeypots to attack back a system or entity. Furthermore, we use state of the art honeypots (HosTaGe, Conpot, Cowrie, Dionaea, ThingPot and U-Pot) for which, to the best of our knowledge, there is no scientific publication suggesting the possibility of an adversary being able to hack their way out of them and attack systems on the Internet through them. Moreover, we want to highlight that we are only utilizing low/medium interaction honeypots. In contrast to high interaction honeypots, which are real systems and thus may be compromised, low/medium interaction honeypots only partially emulate protocols. Adding to this, each honeypot (except HosTaGe that runs on a mobile device) was deployed as a container for better managing and as an additional security layer. HosTaGe is safeguarded by the device's firmware (Samsung's Linux Container (LXC) sandboxing). Furthermore, HosTaGe's implementation of the various protocols does not allow the attacker for a lot of interaction (the reason for this, as with most low interaction honeypots, is the utilization of protocol emulation libraries that are incomplete it terms of capabilities). In regards to measures against reflection attacks (i.e., on CoAP and UPnP) we would like to note the following. The CoAP implementation of HosTaGe is implemented using the *JAVA mbed-coap* library and only responds to service discovery requests with static information. Hence, it does not allow for an attacker to attack other devices. Similarly, for U-Pot we utilized a low interaction image of the IoT device that responds to only service discovery requests (by using a limited UPnP library, i.e., GUPNP).

In addition, for all the honeypots, we performed continuous monitoring on a daily basis. That is, we examined what kind of attacks and communication was taking place and whether anything looked overly suspicious. Moreover, note that all containers had egress rules to limit any traffic attempting to leave the network. As we write on the paper, the majority of the observed attacks come as the result of automated attacks (e.g. via malware). Lastly, the IP space used for the honeypots is part of our monitored university network; we can confirm that we have not received any complaints with regard to the IP addresses of the honeypots neither from our NOC nor our ISP.

A.4 Most common Device-type identifiers with banners/response

Device	Protocol	Device-Type	Banner/Response
HiKVision Camera	Telnet	Camera	"192.168.0.64 login:"
Polycom HDX	Telnet	Camera	"Welcome to ViewStation"
D-Link DCS-6620	Telnet	Camera	"Welcome to DCS-6620"
D-Link DCS-5220	Telnet	Camera	"Network-Camera login:"
Avtech AVN801	UPnP	Camera	"Server: Linux/2.x UPnP/1.0 Avtech/1.0"
Panasonic BB-HCM581	UPnP	Camera	"Friendly Name: Network Camera BB-HCM581"
Anbash NC336FG	UPnP	Camera	"Model Name: NC336FG"
Beward N100	UPnP	Camera	"Friendly Name: N100 H.264 IP Camera - 004B1000E3E2"
Io Data TS-WLC2	UPnP	Camera	"Model Name: TS-WLC2"
Io Data TS-WPTCAM	UPnP	Camera	"Model Name: TS-WPTCAM"
Io Data TS-WLCAM	UPnP	Camera	"Model Name: TS-WLCAM"
Io Data TS-WLCE	UPnP	Camera	"Model Name: TS-WLCE"
G-Cam EFD-4430	UPnP	Camera	Friendly Name: G-Cam/EFD-4430
Seyeon Tech FW7511-TVM	UPnP	Camera	"Model Name: FW7511-TVM"
ZyXEL PK5001Z	Telnet	DSL Modem	"PK5001Z login"
ZTE ZXHN H108N	Telnet	DSL Modem	"Welcome to the world of CLI"
Technicolor modem	Telnet	DSL Modem	"TG234 login:"
ZTE ZXV10	Telnet	DSL Modem	"F670L Login"
Datacom DM991	Telnet	DSL Modem	"DM991CR - G.SHDSL Modem Router"
TP-Link TD-W8960N	Telnet	DSL Modem	"TD-W8960N 6.0 DSL Modem"
Cisco C11-4P	Telnet	DSL Modem	"MODEM : C111-4P"
TP-Link TD-W8968	Telnet	DSL Modem	"TD-W8968 4.0 DSL Modem Router"
BelAir 100N	Telnet	Router	"BelAir100N - BelAir Backhaul and Access Wireless Router"
Tenda Wireless Router	UPnP	Router	"Manufacturer: Tenda"
Totolink N150	UPnP	Router	"Friendly Name: TOTOLINK N150RA"
ZTE H108N	UPnP	Router	"Model Name: H108N"
OBSERVA BHS_RTA 1.0.0	UPnP	Router	"Model Name: BHS_RTA"
DASAN H660GM	UPnP	Router	"Model Name: H660GM"
Huawei HG532e	UPnP	Router	"Model Name: HG532e"
ASUSTeK RT-AC53	UPnP	Router	"Friendly Name: RT-AC53"
NDM	CoAP	Router	"/ndm/login"
QLink	CoAP	Router	title: Qlink-ACK Resource
Signify Philips hue bridge	UPnP	Smart home	"Model Name: Philips hue bridge 2015"
EQ3 HomeMatic	UPnP	Smart Home	"Model Name: HomeMatic Central"
Hyperion 2.0.0	UPnP	Smart Home	"Model Description: Hyperion Open Source Ambient Light"
Home Assistant	Telnet	Smart Home	"Home Assistant: Installation Type: Home Assistant OS"
Home Assistant	MQTT	Smart Home	"homeassistant/light/"
Emby	UPnP	TV Receiver	"Friendly Name: Emby - DS720plus"
Dedicated Micros Digital Sprite 2	Telnet	TV Receiver	"Welcome to the DS2 command line processor"
Roku	UPnP	TV Receiver	"Server: Roku UPnP/1.0 MiniUPnPd/1.4"
Realtek RTL8671	UPnP	Access Point	"Model Name: RTL8671"
Synology DS918+	UPnP	NAS	"Friendly Name: DiskStation (DS918+)"
Sonos ZP100	UPnP	Smart Speaker	"Model Number: ZP120"
Octoprint	MQTT	3D Printer	"octoPrint/temperature/bed"
Gozmart	MQTT	HVAC	"gozmart/sonoff/CC50E3C943CC110511/app"
Advantech	MQTT	HVAC	"Advantech/"
Emerson	Telnet	Remote Display Unit	"Emerson Network Power Co., Ltd."
Trimble SPS855	UPnP	Remote Display Unit	"Friendly Name: SPS855, 6013R31531: Trimble"

Table G.11: Most common device-types with identifiers in banners/response

A.5   Top Telnet and SSH credentials used by count

Protocol	Credentials	Count
Telnet	admin,admin	9,772
Telnet	root,root	1,721
Telnet	root,admin	1,254
Telnet	telnet,telnet	689
Telnet	root,xc3511	556
Telnet	admin,admin123	467
Telnet	root,12345	456
Telnet	user,user	321
Telnet	admin,12345	267
Telnet	admin,polycom	217
Telnet	admin,(blank)	198
SSH	admin, admin	11,543
SSH	root, root	3,432
SSH	root, admin	1,943
SSH	zyfwp, PrOw!aN_fXp	1538
SSH	cisco, cisco	629
SSH	cisco, cisco	629
SSH	admin, ssh1234	254

Table G.12: Top Telnet and SSH credentials used by adversaries

A.6   SHA256 Hash of Malware variants

SlNo	SHA256 Hash	Malware Variant
1	27870ada242e0f7fd5b1e7fc799f503004b3fd2c0f971784208cae31880b9950	Mirai
2	f05b1018a6fb23154885f55e27a7d20c36c186df5f4d08bd061a5666fdb05be9	Mirai
3	ad9d20dd5159975e4c192a335a41eabcb0bc10e3110d894416a025ac9955f7e7	Mirai
4	dd86acf2bd99afd9da305bb9a4c3da320df617e36f53f206fcf161c04152eca4	Mirai
5	c0571eee3ef8830218dd7bbfd7b915cf5516ba91691e1019b2699191ab3a332c	Mirai
6	88511349498f79eaccfab8c9dd39a8d37560a016d00796c70699023fc76938fc	Mirai
7	5552ee40fdb037c9b64be8e43c19bcee05b92578ce52a6998a90c2f1fca5c5b2	Mirai
8	5657f3003c50b602c15054d9fa7dfb2519a43413885c40ab1a617fc19275f913	Mirai
9	f489758839fb6afb5431ca7dff377b6c86168d251300328d0e6a135105233b3f	Mirai
10	5b9d2c6415873feb6b98ca963bd4b61059056087d5010eb096ce00a2726c983f	Mirai
11	5bb032bda8cc48150744fc08684fcf2c898abf0816f1479cfac02fe729cfa637	Mirai
12	dbebd8e8c11f9e06c1a1ab3019015157f1c82ccdda44f0f0707c69ae721c6890	Mirai

Continuation of Table G.13		
SlNo	**SHA256 Hash**	**Malware Variant**
13	72455f499bb407cd090fd079616eb7055824f321d90cbb86bb2f53a757f02c6e	Mirai
14	378df341cea00d8c7838744959fab950d15ae443d14b770cfa2998ae7daf5190	Mirai
15	ae75c29f5f7d3bc602d9cfd355ab6dbcd466c96282fa8ae93a187470ddd34c50	Mirai
16	b8c05074193134695fb975549124835b8f3d1a1ccd24865a2531ad8a90059c7f	Mirai
17	51167f36c3355359a873b19b1aa038fd0772e87b192c8f69b20336d48f980eb6	Mirai
18	bfdd172a08860b7fbfd278e6757f9219d90c25ff47cdf94b57bd3037e81470a1	Mirai
19	652589c71720af72f3566c978fa314408ab12a1286b798f2bec2a4f8525e629d	Mirai
20	e4fafd804c7c9cf29326d4203a74333b211799798cb49d87adb45b9c52938bec	Mirai
21	030b477706540babbfd5997d6afffe47a5cfd3f846521f03873a391a839853c5	Mirai
22	ededadd2a14910547f7dc3d63505b9c03cbf93cecebd302de2e10a75259b13d6	Mirai
23	9b8b0ad1b6f3fa068eec2ddfcb711739b131f4ea5199697a025821729d24ea5b	Mirai
24	4f12ad1c5faa5e43bf17d1906e928e3c7291daa097f9011043582827340604cf	Mirai
25	08fcac8bd754b5b38bad7cb2d17f4347462bc3711a1d82f88da010524ba83f5b	Mirai
26	32b22639b5562d8ef9aa20057053c824ab767cc750a9b17b386f97f829dcdcb3	Mirai
27	94db041c5f1a70c755db90d54c72fb3dfa842729b2d158fb284b3dd90a47491d	Mirai
28	a73ffc17dce716dceb0da272f73d3c6781100aed40565fc601909ef76e908dba	Mirai
29	cda2b6de339a145e6bae502ce3aa71c26de3da7f59547a5764707afdc98fd24e	Mirai
30	acae3ef96626d6b674ca9879419b2fcdc2875bbcc6483f9b4c6057f6374eacde	Mirai
31	e60f7b11d9e26c4a105ca434a2b60bbbd77d69cb13a38b3d2d8aaff0794c9502	Mirai
32	6332c9baecf13d4d9aed26e8d0f14915e0052f34e2cbd84392a3648a0e61fb23	Mirai
33	79d78b3b1aab8e36228f1570659f08c7efc862abc8293291346c837306b3244c	Mirai
34	ec62a759455911c621efb7d6c6aac0b781deabb42931967b712de23ced214589	Mirai
35	3a1063f0af803f8ec5a51076fd5758e1ff784d4eb75645bb81e86cd6fd2504ad	Mirai
36	1925f7a2b715b4af5ff66221447cc5ed135d1b9f9aff2dee8ea1acb62d0dc0a0	Mirai
37	a897bfcd40d42e6d9d8d0b490310a4d21afe4da83bf107f9adc680b52bb09ad9	Mirai
38	f2c7a185f63f76b49c06479b754431b3c897b1e8b47073b0b6e87a49da6db056	Mirai
39	1947ab53faace7d095341791cd2583bcef5419c09b6de6b9052277a3b77e0a14	Mirai
40	bd59588546fe611472c611f46c1a94fd563d59673fa286b7e1d30344bd6cd64b	Mirai
41	0c49abe389cb5f3e59d9f0950468714a68f15c4d1eb1a1c65c9b346ec30471b6	Mirai
42	f064edd2cbb8ab8e0abcfc54406d076390d454b156a6bb71988ebe57b3a3af55	Mirai
43	ce1de869640398a0e51f0f8ad798db97ecfac0b62a3095e823b4ad16f1ef5440	Mirai
44	4cd74e1b5d0441e3b44f4f22c85d41a38dc15ee7de45c6a88b3cadca3c144ef9	Mirai
45	7ba175cd5650ed0d9220003340aae62ee7dec51fea10bc3bf2204dc0899a3873	Mirai
46	d3c865bda24fff7a86d6f70c6909527561097ab7f83db9118dbdd8244dded9b5	Mirai
47	78b6d223f22ed8bf2b628b308eed80a641d415c8a73fdb31994607f3e5e1b570	Mirai
48	b89e37012f39d5abfedf07221cbb1e47e77229210362ad06185f042748118ede	Mirai
49	c06f048b5facaf690ca6bb29f7de30f8cb25803fdeb98e41dc700b1e114b367c	Mirai
50	82080712e408cbeba704ebb29cfd4d1f85cf1f07086008c451331287aa902a16	Mirai
51	5acad83b6314ff5800b5131902a3790d32d9bae5c8a642a23e2936509197072d	Mirai
52	222a737ed1ea068fbc48b3df47627ab9b1f9b06dbe0f0303d38d2546f0afef65	Mirai
53	181a7eca48bef9e356287680dd4a8dd1657662722d26f21305e7939e0a4d96ad	Mirai

Continuation of Table G.13

SlNo	SHA256 Hash	Malware Variant
54	3c725081c68aed61e1ce646f665865f7b171b379ec3241a0f8a0ed4ab717d728	Mirai
55	88a5b54b9281a7c4b421786af35ff2b7e1107712a027f8f07ad3c28224bacc29	Mirai
56	2e687dc6895cd29e515fe81cecf0fad92530d0d2f18a47b7fb92090b7234e0e0	Mirai
57	5d756eb57c9cff97407a699c96219423061a39ff33b36ac3ff2b4563e4a506f9	Mirai
58	9fc6591bbdd807413dac29d5589ce6b8a1d59c7591fb14affba44a5b91add167	Mirai
59	0d2a1e914747bd6ca919180a491839506c90f2c86b1b1fab543569493389accb	Mirai
60	354ea3fc68c4c745d67417554099d0fa523cd6028ce6d9bac66e67c9739a4325	Mirai
61	179933aa4c9b520b636f1aa49f05b922f7d80b7ec252cb485764508704fc7321	Mirai
62	afae979dc58e9b601a75cfc5af9d2764bbb88d9042e984f2b89c417978ab3a4f	Mirai
63	321cb08441c3a780a1247760c348b5a142e66013be3b3e194a2471d51f7f5891	Mirai
64	ee8ae18792c45b4e1ccede856e30fb141ad000142e90eea7c0a80f4ea9da0322	Mirai
65	4fafd982fd204e1549acdb7653cd4532acaada0fe3475f498387649b5211a852	Mirai
66	5ba6803107fc5d942c158ecfb2eedf7d1b620620574789a8244aca3a58608b66	Mirai
67	87cd6daa315466b7260b1e023da2b6dff926c6418592cfdcb6dc10f2bf323901	Mirai
68	4678be773edfec69238f6352033ef27ce0c78c63828434c06ed69d6128a57d73	Mirai
69	3324e5ccd6b28bbb18cf2d7f0e19b48c1603c29a8b562c12d40137b08f7b8725	Mirai
70	1aec808dc691fee0bf8de862cb088f97f3ab637fb7746668f04fe25798955c8a	Mirai
71	9bb73bc9981ee9bdd3f0f628b0e727b6bf8ac06240e56608517487667a2e9f51	Mirai
72	0434c27a45ee62accfc00ca5fabe07d1d730575cca91df1efef17201a90fde29	Mirai
73	6aacfe5ffbc9808d585bfc623d1fec14ae22b9d8eca8e535583c76ef119fa071	Mirai
74	6f7199d4c55b4006c9f451e48ddcd1f80535660927d0aebe1374ad7598929218	Mirai
75	8f144c1cc3a37120a00abafb28091b3e399b4f65b9b798cbea5a123867eeffb2	Mirai
76	f3c2d7da375ed1afc88c1bc79787675603bed1bb82a67d360300bc7e77b5b6b4	Mirai
77	d4a6e144b49a5e16befa2974384a59bcf68da14ef394948bdf1780bbc589ba67	Mirai
78	7c7c7b54beb1bd503ebdc472b08ed35b0c4291fb465bcad34c26a80b92cb682f	Mirai
79	224f2df0563584885aa637f71077ccce8bb4dab9d7e82dcb12dce92d4e0d704c	Mirai
80	0c45b6faed996600eb05585c532fe7e9d34dc85526afffc08b2fe0fda204f0e9	Mirai
81	c91712f66f9522b6219808b0721baf5f309f627be6025b148f8688a89150cdbf	Mirai
82	9e55a50e619d7f3724e0750449097c387601c255839a7e80676f0c25b4217efc	Mirai
83	a99d6d088071bd216de1fb7dc104bc9fa0b5447debf63958ac4ebf904ac8da45	Mirai
84	1b3bb39a3d1eea8923ceb86528c8c38ecf9398da1bdf8b154e6b4d0d8798be49	Mirai
85	faccd187812a48c7911fb1b643bd346c74f4bc7ddcda2c84e97033e0385ff458	Mirai
86	71adda1a01f2a779796673ec08b1155aa55ffb3f40bdd8752b5a3955684d272e	Mirai
87	82df7a015470179794acc9dc60868ea11221525090f5beecb1c98cdba8510389	Mirai
88	57744761595c2dccdf76560c4e0fe7ea33ea85be281d1b7a9c9b4e9e9dbb0221	Mirai
89	9de1b56f76a47fb1aadc6a78f20e0906bdd9dfcf5379f28fa2927fbdf15bd73b	Mirai
90	2ca71e114a5388aa4d17bb0727bb668dca590b81a063670a44d2dab3adc05af0	Mirai
91	c56fafd9207e18c83d2bfb26550aa00fdba64e05bf5b2aea61629d4108c86517	Mirai
92	3321535dc19687c1d2fd5705012d2653fd6a828733302e4b5932780e7637c084	Mirai
93	8e3fb9f382c1a3136da6e83361464e694d77502b483907eb3f9c55890372e66c	Mirai
94	c052c89438af51f7b8af26b3c5864650d0f2c2199653edc76671d62258f234cf	Mirai

SlNo	SHA256 Hash	Malware Variant
	Continuation of Table G.13	
95	1f56bc65381ff6e095c5aa0c84de5d368c08f3a8ee12a0e84c67fcd80626b4fa	Mirai
96	b4623f517f49a825f2f53e4497f944fe10fe9368b3c0db1d30b3ebc63c120962	Mirai
97	b279115318cd447823c8410ee3a318a8c531733404394ec8336184102854c554	Mirai
98	46823451734e47b845bff6d3440ffbd096f8742d9bd8962b1a7c18bf0144d9b4	Mirai
99	8976173ee948c64e89657f734eaea431c5e7a49d5ab7528c676a8d50f1306157	Mirai
100	b79f967aece83b7eda2db4feb08c2eec352eff9d5802f6ea9214064b128987d9	Mirai
101	a05a6affb61f1c84dc30cc0578ddf5aa32b833fc5a772f3abac613293ff89b06	Mirai
102	865c8d9c31a120a92524cf24a8961bc16fe521bea6e72702afc8ac1a0ea9b4ad	Mirai
103	3ef73e98076dc49d83f733eceab93dcffeeefdbdf6f0a36bef756d3448b5d9dc	Mirai
104	5ce6a9bb4daca8f2d6624c654a445d68b2c2f28440c648adb16d0546b9299ecd	Mirai
105	46d8fb0b1e46ff8ee0d65697080af8f7ee11d0a741ae0ca662aedad63a716ebd	Mirai
106	6c5679eb7bd905b3ee86ea5770dbfd8fb50be013c6e93ad1df8fd75a6689d523	Mirai
107	057b9b5d11a4500bc0f46b9c2317ef8f82beaa6d95d5babe0194d4a7379d4f6f	Mirai
108	439dc5e3183a9f4316472691791ca6c33c9e56bce480d88ed5a82c28481f6bc2	Mirai
109	de82fb3927bb9357cdba7f8c595bb87940e7502acb8b605ea7eb0e876ac2808b	Mirai
110	7965a27369b329db4004b871429432beb5c0301c03a48bf64d0961e951e712cf	Mirai
111	b936597d0d868607e45478b9be01a9365078d33bdda2a8c053500c729a8cbaf6	Mirai
112	0c1472a800bdaaad840110f93f3c4b248509f7505fc2a1330af2cdc7c2eccfc4	Mirai
113	bfee2d1d34214a93024447bc054c1eea2b05111a74508bd74997eea6a7c4ef65	Mirai
114	f060058462bfaef0bd9382de38d238b96ff4f886967b70020406dab38190bff6	LuaBot
115	0206efba7fc13700efd59354e9c6ca4d1ffe34f6689bd195798181824d46b83d	LuaBot
116	e1f6d967db61ee131dc32b817a9285f5da3ebe3e1f9a4281c8fac9339e2b4521	LuaBot
117	9866b4f2f533de7d742251915802dab355a59f10a51a8bf7d146fd4cb015cd5a	Brickerbot
118	4f9b895a2785f9788fcae8743ab04a24b62e0962b1f8a28dc1206c52327b7916	HEHbot
119	8a2a28d164a6d4011e83ae3f930de8bf1e01ba2e013bee43460f2f58bdaf4109	Photominer
120	01d8e2bcf22422e9c995d43c403c63477389fc9f4a141ef3bbd31c8f5c6ef7e6	Mozi
121	01d8e2bcf22422e9c995d43c403c63477389fc9f4a141ef3bbd31c8f5c6ef7e6	Mozi
122	c9038e31f798119d9e93e7eafbdd3e0f215e24ee2200fcd2a3ba460d549894ab	Lokibot
122	b47e281bfbeeb0758f8c625bed5c5a0d27ee8e0065ceeadd76b0010d226206f0	WannaCry
123	0a73291ab5607aef7db23863cf8e72f55bcb3c273bb47f00edf011515aeb5894	WannaCry
124	d7d0f18071899c81ee90a7f8b266bd2cf22e988da7d0e991213f5fb4c8864e77	LemonDuck
125	d1e82d4a37959a9e6b661e31b8c8c6d2813c93ac92508a2771b2491b04ea2485	FritzFrog
	End of Table	

Table G.13: Malware hashes detected by our honeypots as found in Virustotal

References

[1] J. Frahim, C. Pignataro, J. Apcar, and M. Morrow. (2015) Securing the internet of things: A proposed framework.

[2] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "Ddos in the iot: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[3] NIST, "misconfiguration," 2021. [Online]. Available: https://csrc.nist.gov/glossary/term/misconfiguration

[4] A. Burt, "Gitpaste-12: a new worming botnet with reverse shell capability spreading via github and pastebin," Juniper Threat Labs, 2020. [Online]. Available: https://blogs.juniper.net/en-us/threat-research/gitpaste-12

[5] R. I. Augusto, N. C. Patrick, and I. T. Karen, "Xord-dos, kaiji variants target exposed docker servers," Trend Micro, 2020. [Online]. Available: https://www.trendmicro.com/en_us/research/20/f/xorddos-kaiji-botnet-malware-variants-target-exposed-docker-servers.html

[6] Malwaremustdie, "Rhombus - linux ddos botnet aims vps & iot, w/persistence & dropper," Alienvault, 2020. [Online]. Available: https://otx.alienvault.com/pulse/5e6aacfe61b118f3fc41026a

[7] I. Vaccari, M. Aiello, and E. Cambiaso, "Slowite, a novel denial of service attack affecting mqtt," *Sensors*, vol. 20, no. 10, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/10/2932

[8] ——, "Slowtt: A slow denial of service against iot networks," *Information*, vol. 11, no. 9, 2020. [Online]. Available: https://www.mdpi.com/2078-2489/11/9/452

[9] A. T. Vasques and J. J. C. Gondim, "Amplified reflection ddos attacks over iot reflector running coap," in *2020 15th Iberian Conference on Information Systems and Technologies (CISTI)*. Seville, Spain: IEEE, 2020, pp. 1–6.

[10] ENISA. (2020) Enisa threat landscape 2020 - malware. ENISA. [Online]. Available: https://www.enisa.europa.eu/publications/malware

[11] S. Andy, B. Rahardjo, and B. Hanindhito, "Attack scenarios and security analysis of mqtt communication protocol in iot system," in *2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*. Yogyakarta, Indonesia: IEEE, 2017, pp. 1–6.

[12] I. Mcateer, M. I. Malik, Z. Baig, and P. Hannay, "Security vulnerabilities and cyber threat analysis of the amqp protocol for the internet of things," in *Australian*

Information Security Management Conference. Perth, W.A.: Edith Cowan University, 2017, p. 11.

[13] M. H. Syed, E. B. Fernandez, and J. Moreno, "A misuse pattern for ddos in the iot," in *Proceedings of the 23rd European Conference on Pattern Languages of Programs*, ser. EuroPLoP '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3282308.3282343

[14] hackingump. (2020) Upnp – messing up security since years. [Online]. Available: https://malwareandstuff.com/upnp-messing-up-security-since-years/

[15] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "Iotpot: Analysing the rise of iot compromises," in *9th USENIX Workshop on Offensive Technologies (WOOT 15)*. Washington, D.C.: USENIX Association, Aug. 2015, p. 9. [Online]. Available: https://www.usenix.org/conference/woot15/workshop-program/presentation/pa

[16] T. Luo, Z. Xu, X. Jin, Y. Jia, and X. Ouyang. (2017) Iotcandyjar : Towards an intelligent-interaction honeypot for iot devices.

[17] M. Wang, J. Santillan, and F. Kuipers, "Thingpot: an interactive internet-of-things honeypot," 2018.

[18] M. A. Hakim, H. Aksu, A. S. Uluagac, and K. Akkaya, "U-pot: A honeypot framework for upnp-based iot devices," in *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*. Orlando, FL, USA: IEEE, 2018, pp. 1–8.

[19] J. Wang, M. K. Lim, C. Wang, and M.-L. Tseng, "The evolution of the internet of things (iot) over the past 20 years," *Computers & Industrial Engineering*, vol. 155, p. 107174, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360835221000784

[20] L. Markowsky and G. Markowsky, "Scanning for vulnerable devices in the internet of things," in *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, vol. 1. Warsaw, Poland: IEEE, 2015, pp. 463–467.

[21] SHODAN, "Shodan," 2021. [Online]. Available: https://www.shodan.io/

[22] R. D. Graham, "Masscan: Mass ip port scanner," 2014.

[23] G. Lyon, "Nmap network mapper," 2021. [Online]. Available: https://nmap.org/

[24] G. H. Tools. (2021) Pft printer exploration. [Online]. Available: http://www.phenoelit.org/fr/tools.html

[25] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, "Demystifying iot security: An exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale iot exploitations," *IEEE Communications Surveys Tutorials*, vol. 21, no. 3, pp. 2702–2733, 2019.

[26] D. Springall, Z. Durumeric, and J. A. Halderman, "Ftp: The forgotten cloud," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, IEEE.   Toulouse, France: IEEE, 2016, pp. 503–513.

[27] T. H. Project, "The honeynet project."

[28] L. Rist, J. Vestergaard, D. Haslinger, A. Pasquale, and J. Smith, "Conpot ics/scada honeypot," *Honeynet Project (conpot. org)*, 2013.

[29] D. Tools, "Web honeypot," 2010. [Online]. Available: https://github.com/DinoTools/dionaea/

[30] E. Vasilomanolakis, S. Srinivasa, and E. Lygerou, "Hostage: mobile honeypots for rapid deployment," in *Black Hat Europe 2020*, 2020.

[31] Belkin. (2021) Belkin wemo. [Online]. Available: https://www.belkin.com/us/

[32] E. Vasilomanolakis, S. Karuppayah, M. Fischer, M. Mühlhäuser, M. Plasoianu, L. Pandikow, and W. Pfeiffer, "This network is infected: Hostage-a low-interaction honeypot for mobile devices," in *Proceedings of the Third ACM workshop on Security and privacy in smartphones & mobile devices*, 2013, pp. 43–48.

[33] H. Shimada, K. Ito, H. Hasegawa, and Y. Yamaguchi, "Implementation of mqtt/coap honeypots and analysis of observed data," *SECURWARE 2019, The Thirteenth International Conference on Emerging Security Information, Systems and Technologies*, vol. 10, pp. 35–40, 2019.

[34] N. 360, "Anglerfish honeypot," 2021. [Online]. Available: https://blog.netlab.360.com/tag/anglerfish-honeypot/

[35] Z. Durumeric, M. Bailey, and J. A. Halderman, "An internet-wide view of internet-wide scanning," in *23rd USENIX Security Symposium (USENIX Security 14)*.   San Diego, CA: USENIX Association, Aug. 2014, pp. 65–78. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/durumeric

[36] H. Heo and S. Shin, "Who is knocking on the telnet port: A large-scale empirical study of network scanning," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ser. ASIACCS '18.   New York, NY, USA: Association for Computing Machinery, 2018, p. 625–636. [Online]. Available: https://doi.org/10.1145/3196494.3196537

[37] M. Jonker, A. King, J. Krupp, C. Rossow, A. Sperotto, and A. Dainotti, "Millions of targets under attack: A macroscopic characterization of the dos ecosystem," in *Proceedings of the 2017 Internet Measurement Conference*, ser. IMC '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 100–113. [Online]. Available: https://doi.org/10.1145/3131365.3131383

[38] P. Richter and A. Berger, "Scanning the scanners: Sensing the internet from a massively distributed network telescope," in *Proceedings of the Internet Measurement Conference*. Amsterdam, Netherlands: Association for Computing Machinery, New York,NY,United States, 2019, pp. 144–157.

[39] D. Moore, "Network telescopes: Observing small or distant security events," in *11th USENIX Security Symposium (USENIX Security 02)*. San Francisco, CA: USENIX Association, Aug. 2002, p. 9. [Online]. Available: https://www.usenix.org/conference/11th-usenix-security-symposium/network-telescopes-observing-small-or-distant-security

[40] T. Holz and F. Raynal, "Detecting honeypots and other suspicious environments," in *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*. West Point, NY, USA: IEEE, June 2005, pp. 29–36.

[41] A. Vetterl and R. Clayton, "Bitter harvest: Systematically fingerprinting low- and medium-interaction honeypots at internet scale," in *12th USENIX Workshop on Offensive Technologies (WOOT 18)*. Baltimore, MD: USENIX Association, Aug. 2018, p. 9. [Online]. Available: https://www.usenix.org/conference/woot18/presentation/vetterl

[42] O. Surnin, F. Hussain, R. Hussain, S. Ostrovskaya, A. Polovinkin, J. Lee, and X. Fernando, "Probabilistic estimation of honeypot detection in internet of things environment," in *2019 International Conference on Computing, Networking and Communications (ICNC)*. Honolulu, HI, USA: IEEE, 2019, pp. 191–196.

[43] S. Srinivasa, J. M. Pedersen, and E. Vasilomanolakis, "Gotta catch 'em all: a multistage framework for honeypot fingerprinting," 2021.

[44] L. Babun, K. Denney, Z. B. Celik, P. McDaniel, and A. S. Uluagac, "A survey on iot platforms: Communication, security, and privacy perspectives," *Computer Networks*, vol. 192, p. 108040, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128621001444

[45] B. Vignau, R. Khoury, S. Hallé, and A. Hamou-Lhadj, "The evolution of iot malwares, from 2008 to 2019: Survey, taxonomy, process simulator and perspectives," *Journal of Systems Architecture*, vol. 116, p. 102143, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762121001053

[46] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the mirai botnet," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1093–1110. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis

[47] ——, "Understanding the mirai botnet," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1093–1110. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis

[48] Z. Durumeric, E. Wustrow, and J. A. Halderman, "Zmap: Fast internet-wide scanning and its security applications," in *22nd USENIX Security Symposium (USENIX Security 13)*. Washington, D.C.: USENIX Association, Aug. 2013, pp. 605–620. [Online]. Available: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/durumeric

[49] Z. Durmeric, "zgrab2," 2018. [Online]. Available: https://github.com/zmap/zgrab2

[50] ZMap, "Zmap block and allow lists," 2020. [Online]. Available: https://github.com/zmap/zmap/wiki/Block-and-Allow-Lists

[51] fireHOL, "Europe blocklist," 2021. [Online]. Available: https://github.com/firehol/blocklist-ipsets/blob/master/ip2location_country/ip2location_continent_eu.netset

[52] R. Research, "Project sonar," 2021. [Online]. Available: https://www.rapid7.com/research/project-sonar/

[53] VMWare, "Rabbitmq," 2021. [Online]. Available: https://www.rabbitmq.com/

[54] A. Foundation, "Apache qpid," 2021. [Online]. Available: https://qpid.apache.org/

[55] ——, "Apache activemq," 2021. [Online]. Available: https://activemq.apache.org/

[56] N. National Vulnerability Database, MITRE, "Common vulnerabilities and exposures."

[57] NIST. (2021) National vulnerability database. [Online]. Available: https://nvd.nist.gov/

[58] S. Arvind and V. A. Narayanan, "An overview of security in coap: Attack and analysis," in *2019 5th International Conference on Advanced Computing Communication Systems (ICACCS)*. Coimbatore, India: IEEE, 2019, pp. 655–660.

[59] Cloudflare, "Ssdp ddos attack," 2021. [Online]. Available: https://www.cloudflare.com/learning/ddos/ssdp-ddos-attack/

[60] S. Morishita, T. Hoizumi, W. Ueno, R. Tanabe, C. Gañán, M. J. van Eeten, K. Yoshioka, and T. Matsumoto, "Detect me if you... oh wait. an internet-wide view of self-revealing honeypots," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, IEEE. Washington DC, USA: IEEE, 2019, pp. 134–143.

[61] P. Maddux. (2019) Honeypy honeypot. [Online]. Available: https://github.com/foospidy/HoneyPy

[62] M. Oosterhof, "Cowrie ssh/telnet honeypot," 2016. [Online]. Available: :https://github.com/micheloosterhof/cowrie

[63] Cymmetria, "Mtpot," 2016. [Online]. Available: https://github.com/Cymmetria/MTPot

[64] P. Jeitner. (2018) Telnet iot honeypot. [Online]. Available: https://github.com/Phype/telnet-iot-honeypot

[65] Decester, "An ssh honeypot," 2000. [Online]. Available: https://github.com/desaster/kippo

[66] P. Adkins. (2017) Kako honeypot. [Online]. Available: https://github.com/darkarnium/kako

[67] M. Stampar. (2017) Hontel honeypot. [Online]. Available: https://github.com/stamparm/hontel

[68] CAIDA. (2021) The caida ucsd network telescope "darknet scanners" dataset - april-may2021. [Online]. Available: https://www.caida.org/data/passive/telescope-darknet-scanners_dataset.xml

[69] C. STARDUST, "Flow level traffic (flowtuple)," 2021. [Online]. Available: https://stardust-dev.caida.org/docs/data/flowtuple/

[70] GreyNoise, "Greynoise," 2022. [Online]. Available: https://viz.greynoise.io/

[71] Virustotal, "Virustotal," 2022. [Online]. Available: https://www.virustotal.com

[72] HKVision, "Hkvision network camera - user manual," 2021. [Online]. Available: https://www.hikvision.com/UploadFile/image/EN-user%20manual%20of%20%20network%20camera%20v3.0.0.pdf

[73] Z. Durumeric, "Ztag," 2017. [Online]. Available: https://github.com/zmap/ztag

[74] ipgeolocation, "ipgeolocation.io," 2021. [Online]. Available: ipgeolocation

[75] S. University, "Censys universal ipv4 internet dataset," 2021. [Online]. Available: https://scans.io/

[76] C. Inc., "Binaryedge," 2021. [Online]. Available: https://www.binaryedge.io/

[77] Z. Org., "Zoomeye," 2021. [Online]. Available: https://www.zoomeye.org/

[78] F. C. Surveying and Mapping, "Fofa," 2021. [Online]. Available: https://fofa.so/

[79] C. . D. S. R. A. University. (2021) Rwth aachen scan. [Online]. Available: http://researchscan.comsys.rwth-aachen.de/

[80] Stretchoid.com, "Stretchoid.com," 2021. [Online]. Available: http://stretchoid.com/

[81] Bitsight.com, "Bitsight.com," 2021. [Online]. Available: https://www.bitsight.com/

[82] ShadowServer.org, "Shadowserver.org," 2021. [Online]. Available: https://www.shadowserver.org/

[83] InterneTTL, "Internettl," 2021. [Online]. Available: http://www.internettl.org/

[84] A. Strike, "Alpha strike," 2021. [Online]. Available: https://www.alphastrike.io

[85] Sharashka. (2021) Sharashka. [Online]. Available: https://sharashka.io/data-feeds

[86] CriminalIP. (2021) Criminalip. [Online]. Available: https://security.criminalip.com/

[87] ipip.net. (2021) ipip.net. [Online]. Available: https://en.ipip.net/

[88] N. S. Research, "Net systems research," 2021. [Online]. Available: https://www.netsystemsresearch.com/

[89] LeakIX, "Leakix," 2021. [Online]. Available: https://leakix.net/

[90] Onyphe, "Onyphe," 2021. [Online]. Available: https://www.onyphe.io/

[91] Natlas, "Natlas," 2021. [Online]. Available: https://github.com/natlas/natlas

[92] Quadmetrics, "Quadmetrics," 2021. [Online]. Available: https://www.quadmetrics.com/

[93] Arbor-Bbservatory, "arbor-observatory," 2021. [Online]. Available: https://www.arbor-observatory.com/

[94] ICSA, "Cisa-icsa-16-299-01," 2016. [Online]. Available: https://us-cert.cisa.gov/ics/advisories/ICSA-16-299-01

[95] T. T. Project. (2022) Exonerator. The Tor Project. [Online]. Available: https://metrics.torproject.org/exonerator.html

[96] Censys, "Censys search," 2021. [Online]. Available: https://censys.io/

[97] G. Wan, L. Izhikevich, D. Adrian, K. Yoshioka, R. Holz, C. Rossow, and Z. Durumeric, "On the origin of scanning: The impact of location on internet-wide scans," in *Proceedings of the ACM Internet Measurement Conference*, ser. IMC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 662–679. [Online]. Available: https://doi.org/10.1145/3419394.3424214

[98] ——, "On the origin of scanning: The impact of location on internet-wide scans," in *Proceedings of the ACM Internet Measurement Conference*, ser. IMC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 662–679. [Online]. Available: https://doi.org/10.1145/3419394.3424214

[99] F. IKE, "Malpedia," 2021. [Online]. Available: https://malpedia.caad.fkie.fraunhofer.de/

[100] abuse.ch, "Malwarebazaar," 2021. [Online]. Available: https://bazaar.abuse.ch/

Paper H

A Bad IDEa: Weaponizing uncontrolled online-IDEs in availability attacks

Shreyas Srinivasa, Dimitrios Georgoulias, Jens Myrup Pedersen,
Emmanouil Vasilomanolakis

The layout has been revised.

Abstract

Botnets are an ongoing threat to the cyber world and can be utilized to carry out DDoS attacks of high magnitude. From the botmaster's perspective, there is a constant need for deploying more effective botnets and discovering new ways to bolster their bot ranks. Integrated Development Environments (IDEs) have been essential for software developers to write and compile source code. The increasing need for remote work and collaborative workspaces have led to the IDE-as-a-service paradigm that offers online code editing and compilation with multiple language support. In this paper, we show that a multitude of online IDEs do not run control checks on the user code and can be therefore leveraged by a botnet. We examine the concept of uncontrolled execution environments and present a proof of concept to show how uncontrolled online-IDEs can be weaponized to perform large-scale attacks by a botnet. Overall, we detect a total of 719 online-IDEs with uncontrolled execution environments and limited sandboxing. Lastly, as ethical disclosure, we inform the IDE developers and service providers of the vulnerabilities and propose countermeasures.

1   Introduction

Botnets malicious networks of infected systems responsible for high-impact cyber attacks like Distributed Denial of Service (DDoS). They operate by leveraging vulnerable devices connected to the Internet to execute attacks and are managed through a command and control system (C&C Server). Mirai-like malware has caused high-impact attacks in the past, and infected vulnerable IoT devices for performing DoS-like attacks [1]. Moreover, research studies indicate that browser-based bots are more effective and economical than malware-based bots [2]. The ENISA Threat Landscape Report 2021 states a rise in newer malware used for Denial of Service attacks, ransomware injection, and crypto mining. The report further states that the number of attacks due to malware is decreasing from previous years; however, the focus of newer malware is reduced on quantity but possesses increased quality [3]. This entails that bot developers are exploring newer delivery methods that are more discreet and effective [4].

Programmers have traditionally used IDEs (Integrated Development Environment) for software development. IDEs facilitate the compilation, debugging, and execution of language-specific source code. The leap in cloud computing has aided the idea of online-IDE, and REPL environments (read, eval, print, loop), that are offered as a service on the Internet [5]. Compared to traditional local-IDEs, online-IDEs have no prerequisites like installation or setup. Like local-IDEs, online-IDEs provide features like project creation, sharing, and version control that further facilitate collaboration, remote work, training, and interviewing possibilities. Moreover, many online-IDEs offer multi-language support that includes diverse programming and scripting styles. Online-

IDEs are now popular for many online learning platforms, collaborative development services, and cloud-ready deployment providers.

However, upon careful analysis of online-IDEs and REPL platforms on the Internet, we observe that many do not perform checks on the user code and can be therefore leveraged to execute arbitrary code with malicious intent. Furthermore, recent research shows deceptive source-code attacks that appear different to the compiler and human eye, can be used to deceive the compilers into performing malicious operations [6]. In this work, we aim at checking the uncontrolled behavior of IDEs by executing code that leads to flooding requests on a target hosted in our lab infrastructure. Furthermore, we implement a bot that can perform Denial of Services attacks by exploiting several such online-IDE environments. To the best of our knowledge, there is no previous work which looks at leveraging uncontrolled online IDE environments to perform attacks on the Internet. Our contributions are summarized as follows:

- We examine the concept and criteria for uncontrolled execution environments and find vulnerable online-IDEs by searching the Internet.

- As a proof of concept, we implement a bot that exploits the uncontrolled IDEs and performs a flooding attack against a web server hosted at our lab.

- We estimate the magnitude of the attacks possible from uncontrolled online IDEs by performing multiple attack types.

The rest of the paper is structured as follows. In Section 2 we discuss the background and related work. Section 3 gives an overview of online-IDEs and uncontrolled execution environments. We discuss our methodology in section 4 and section 5 provides an evaluation of our approach. In section 6 we discuss the attack types and the limitations. Section 7 describes the ethical considerations followed in our methodology and disclosure. We discuss the future work and conclude in Section 8.

2 Background & Related Work

2.1 Online IDEs

Modern IDEs provide features that help in accelerating development with the use of Artificial Intelligence, collaborative development, cloud deployments [7], and additional features that include build automation tools, class browsers, object browsers, and version control. Online IDEs provide most features of a local IDE with the advantage of no installation required on the user system and with the possibility of remote access. However, there are some issues specific to online-IDEs, like running static and dynamic program analysis on the user program [8]. The rise in demand for online platforms and cloud deployments, leads to an increase in online IDE services offered for training, assessments, and development environments. With this increase, there is also a

risk of potential exploitation of these platforms, wherein an adversary could use them to spoof the attack source. Adversaries and botnet campaigns can utilize uncontrolled online-IDEs to cause varied attacks such as availability attacks, crypto mining, and malware injection [9]. For example, *PyCryptoMiner*, a Linux crypto-miner botnet, spreads through a compromised SSH service and deploys a base64-encoded Python script that connects to the command and control (C&C) server to fetch and execute the crypto-mining Python code [10]. The bot mines the Monero [11] crypto-currency, which is the preferred mode of payment in the Darkweb [12]. Furthermore, botnets like Mirai have caused availability attacks of high magnitude by infecting vulnerable systems [1].

2.2   Uncontrolled execution environment

Although there is research towards enhancing the capabilities of IDEs through the inclusion of static and dynamic testing plugins, there is little work on control of execution in online-IDEs. Wu et al. summarize the problems in online-IDEs in regards to uncontrolled execution into (i) wrong file operations, (ii) banned method calls, and (iii) excessive resource consumption that can lead to arbitrary code execution and resource depletion [13]. The authors indicate that IDEs must offer partial file-based operations; for example, deletion of a file on the platform must not be permitted. Similarly, the authors specify the need for banning specific methods and packages that facilitate a compromise of the IDE infrastructure or remote systems. The authors emphasize the need for timeouts that limit resource consumption for the user program and present techniques that can be used to handle the three risks using a program behavior analysis and control model. The model includes static and dynamic analysis techniques to analyze the program behavior and control the code execution.

Arbitrary code execution (ACE) is an adversarial technique where the attacker can execute malicious code on the target system [14]. The attackers leverage the vulnerabilities in the target system to gain access to an execution environment. ACE is not uncommon in online-IDEs as they offer users an open code execution environment. While the main objective of these IDEs is to provide an online execution environment, there is little focus on controlling the environment for any malicious code (for example, HTTP Flood requests). Kiransky et al. propose program shepherding, a method for monitoring control flow transfers during program execution to enforce a security policy [14]. The authors provide three techniques from program shepherding that act as building blocks for security policies, including restricting execution privileges, restricting control transfers, and sandboxing checks. The authors further present a detailed approach to security policies regarding program shepherding that ensures malicious code detection through multiple techniques and prevents execution. Program shepherding includes IDE sandboxing, ensuring that malicious code execution does not impact external systems.

The approach closest to our work is from Pellegrino et al. [2], where the authors explore the idea of using browser-based DDoS botnets and review ways attackers can

weaponize them. The authors present three ways of using browser-based JavaScript to initiate thousands of HTTP requests per second and evaluate the costs compared to a traditional botnet. In our work, we specifically search the Internet to find many online-IDEs that do not have controlled execution environments and have limited sandboxing capability. We demonstrate the impact through various attack types originating from these vulnerable IDE instances. To the best of our knowledge, our work is the first to explore the area of uncontrolled execution in online-IDEs and assess the impact of potential attacks through measurement and estimation.

3 Uncontrolled execution environments

This section describes the generic architecture of online-IDEs and the criteria for classifying the online-IDEs into uncontrolled execution environments.

3.1 Generic architecture of online-IDEs

The architecture of online-IDEs can be broken down into three components (i) frontend, (ii) backend, and (iii) messaging service. Figure H.1 shows the generic architecture of online-IDEs. The frontend component provides a GUI as a web application for the user to input the source code, execute button, and a window to view the output of the executed code, similar to that of local IDE. The backend component contains the compiler and the file system that compiles the user code. This component can either share the host of the frontend or on a remote host for scalability purposes. The messaging service is responsible for transporting the user input (source code) from the frontend to the backend and the output from the backend to the frontend. In addition to the frontend, backend, and message transport, some online-IDEs offer extensions that provide collaboration, version control, build tools, and other features. There exist opensource online-IDE frameworks such as Eclipse Theia [5], ICEcoder [15], Microsoft VSCode [16], Code-server [17] and AtheosIDE [18] that follow similar architecture. However, we observe from our reconnaissance that most online-IDEs have their stack similar to the generic architecture.

3.2 Uncontrolled online-IDE environments

In order to classify an online-IDE as having an uncontrolled execution environment that can be exploited and weaponized to carry out flooding attacks, we adopt and extend the criteria defined by Wu et al. [13].
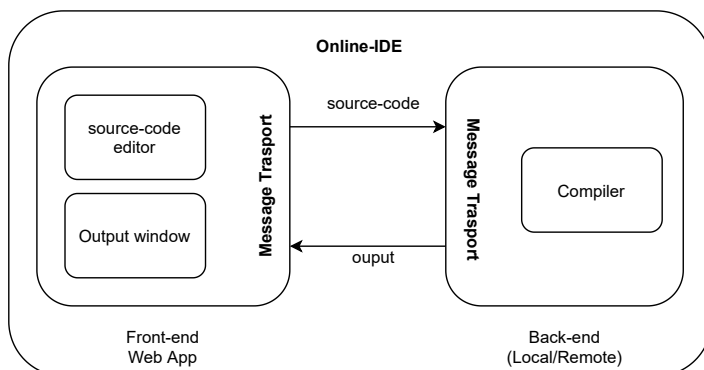
Fig. H.1: Generic architecture of online IDEs

Unrestricted file operations

File operations such as read, write and modify are facilitated through the default packages in many programming languages. The file operations facilitate data import from files and export of output. Most online-IDEs support the import of source-code through files, which we observe are not validated for malware. Furthermore, it is crucial to restrict access to the file system on the hosting system. Unrestricted file operations on the online-IDEs can lead to malware injection, installation of malicious libraries, and corruption of the file system that can compromise the availability of the service.

Unrestricted package/module import

Programming languages depend on modules or packages for specific operations like creating HTTP requests or file handling. Developers import/include these packages into their source code to support such operations. We observe that an adversary can leverage unrestricted package imports for malicious purposes like creating HTTP floods or malware injection.

Unbounded resource consumption

It is crucial to prevent or limit the execution of a program that consumes resources as this may entail resource depletion and eventually cause the online-IDE to crash. However, many online-IDEs run on scalable cloud platforms. Bots can leverage such online-IDEs to carry out high magnitude attacks or inject crypto-mining malware. An adversary can further leverage the elastic resources offered by the IDE to run malicious code that exploits remote systems.

Non-sandboxed environments

Sandboxing is a mechanism in which an instance is isolated to prevent any spread of vulnerabilities or infections to other machines in the network. Furthermore, sandboxed environments offer controlled use of underlying resources and are ideal for executing untested code. Non-sandboxed environments are risky, allow for access to networked systems, and can be used to exploit remote systems. Non-sandboxed online-IDEs in particular are ideal for malware spreading. Botnets can inject malware or execute malicious code to cause a flood on remote systems and further allow communication between the infected system and the control server.

Stateless runtime sessions

Web servers maintain sessions to maintain the current user's data for a period. Online-IDEs can use sessions to track the use and the requests from the current user to limit them to a specific period. Moreover, online-IDEs can use sessions to track the user's state and stop the program execution when the user state is idle or disconnected. In stateless sessions, the online-IDEs do not keep track of the user sessions, which can be exploited by a user running arbitrary code and terminating the session while the online-IDE is still executing the user code. The user can create multiple sessions, run arbitrary code and close the sessions while saving system resources. Furthermore, an online IDE must also restrict the number of sessions per user for controlled resource usage.

4   Methodology

This section presents our methodology for finding online IDEs, checking for uncontrolled execution, and leveraging them in our botnet for performing a Denial of Service attack.

4.1   Reconnaissance

We use Google Dorking to find IDEs on the Internet using. Google Dorking is the process of finding specific web pages on the Internet by using search parameters with keywords [19]. For example, *intext:"online IDE"* returns a list of online IDEs. The search parameters can be narrowed to find language-specific IDEs, like *intext:"online python compiler"*. The keyword search in the dork process of our approach has specific keywords, for example, "Python Online IDE" that provide language-specific results. However, some online-IDEs support execution of multiple languages. In this case, we manually determine the languages supported by the IDEs to check for multi-language support. This work limits the proof of concept to include online-IDEs that support Python language execution. Moreover, we leverage datasets from Internet-wide scanning services like Censys [20] and Shodan [21] for searching for online-IDE instances

using keywords like "online IDE", "online REPL", and further filtering the results using common labels contained in the HTML of code-editors like the Ace editor [22] (e.g. JavaScript editor syntax like *ace.edit*).
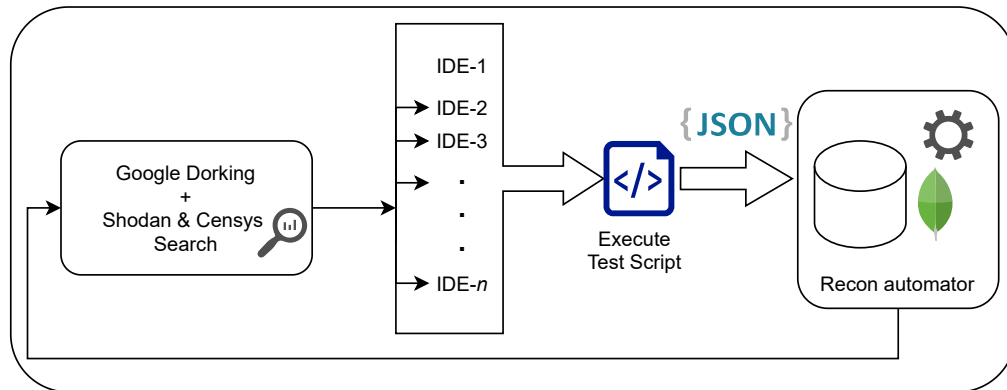
Fig. H.2: Reconnaissance-phase automation

The overall reconnaissance process is illustrated in Figure H.2. After the Google Dorking process, we use language-specific scripts to check if the IDEs from the search results support uncontrolled code execution and group them based on language in our database. The IDEs are further checked for uncontrolled execution parameters by execution of custom language-specific scripts. The output of the scripts is posted to a remote server repository as a *json* document embedded in an HTTP post request. The server repository contains a list of all the IDEs with an uncontrolled execution environment, the language supported by the IDE, and additional metadata about rate limiting. The reconnaissance process is performed weekly to find new IDE environments and check if the existing IDEs in the list are still unpatched.

To check for the uncontrolled execution parameters defined in Section 3.2 we use the following approaches in our proof of concept script:

- Unrestricted-file operations: To check for restricted file read and write operations, we try accessing the environment variables from the host. The access to environment variables further helps determine if the IDE is operating in container mode. We use the checks os.environget() to read the HOME variable and os.environ['FOO']='1' to create a new variable in our proof of concept script. To check for delete operations, we remove the environment variable created during the test write operation. We set flags to determine the successful operation. In addition to the above checks, we check for access to some critical file system paths.

- Unrestricted package/module import: We import packages from python default libraries such as sockets, threading, and os that can be used for our proof of

concept. The successful import of the packages is determined by implementing checks in the proof of concept script.

- Unbounded resource consumption: By importing the threading package from the previous check, we implement a function that can create a large number of HTTP requests over time. In this way, we check for both unbounded resource consumption and network rate limiting.

- Non-sandboxed environments: To check if the python code is executing under a container mode, we read the "/proc/self/cgroup" and check if the field "docker" exists. In addition to the container check, we check for Internet access through the IDE host by executing a simple HTTP request through the sockets package.

- Stateless runtime-sessions - To check if an online-IDE environment supports state-less runtime sessions, we execute multiple HTTP requests to our test webserver for a specific period (i.e., 5 seconds) and close the IDE session. We measure the number of requests received and the period to check the total execution time from the first request received.

4.2   Botnet architecture

In order to explore the magnitude of the discovered vulnerability, we decided to develop an application that would, to some extent, simulate the operation of a real-world bot, part of a botnet architecture. This translates into the botmaster having control over the bots and utilizing them to carry out attacks at will. Additionally, in this particular case, the botmaster does not have to worry about propagating the bot malware to infect new hosts and increase the size of their network, and also the availability of the bots/IDE instances is very high since they are running on active websites.

Overall, there are only two requirements for the botnet to be functional, namely discovering the IDEs (see Section 4.1) and then including them in the *botmaster application.*

Botmaster application

The basis for the vulnerability's exploitation lies within the arbitrary execution of code on the IDEs. In our implementation, which was developed in *Java*, for this task we use the *Selenium* web browser automation software and a *Chrome WebDriver*. Locating the *XPaths* of the elements of interest in the HTML of each website allowed for the interaction through Selenium, which runs locally in our code editor. The most vital elements in common in all IDEs were the code editor text-area, the programming language option, and the run/execute button. These XPaths were hardcoded into the application, and since they differed in each platform, the entire process had to be repeated uniquely for each IDE. Lastly, to provide ease and simplicity to our experiments, as part of the

botmaster application, we included an interface that can be used to navigate through the list of available IDEs and to orchestrate attacks by "tailoring" the *Bot Attack Code* (see Section 4.2) which can be deployed on the IDE, through a *Start Attack* button.

Bot attack code

The bot attack code is an HTTP flood attack; it was found publicly available on a *GitHub* repository * and is a part of the botmaster application. It is written in *Python*, and it was slightly altered in order to match the requirements of our experiments. The attack revolves around 4 discrete variables, the *target*, the *attack duration*, the *number of utilized IDEs*, and the *sessions per IDE*. After inserting the values of these variables in the botmaster interface, the attack code runs on the chosen IDEs. At this point, we noticed that a large number of IDEs would have an issue providing the expected indentation in the code using the carriage return $\backslash r$ and new line $\backslash n$ whitespace characters. Hence, the solution was to create two separate attack classes, one that would use the whitespace characters and one that would use Selenium keyboard commands such as ENTER, HOME, and BACKSPACE, to achieve the desired outcome.

4.3 Experimental setup

We leverage 18 online-IDEs supporting the Python program execution discovered during the reconnaissance process. To ethically involve these IDEs in our experiment, we ask for consent from the environments that provide contact information about the owner. The experimental setup is described in Figure H.3. We deploy an *Nginx* web server in our lab to target the attacks from the IDEs. We set up the *Zeek-IDS* on the host of the webserver to log all the ingress traffic on the webserver.

To visualize the logs from the Zeek-IDS and monitor the resources on the host, we use the *Kibana* visualization dashboard. All ingress traffic towards the server is logged. The web server and Zeek are set up on a host with a quad-core Intel Xeon processor and a memory of 32 gigabytes. The Nginx web server listens on the HTTP port 4444 and is deployed with the default configuration. The Elastic search database and the Kibana dashboard run on a remote host, and the Zeek logs are shipped using an Elastic agent. The bot developed for the proof of concept runs on a remote client in our lab. The bot client has a quad-core Intel Xeon processor of 2.4 GHz and 32 gigabytes of memory. The web server is publicly accessible via the Internet through an unfiltered network in the lab environment.

4.4 Exploitation

Before we use the uncontrolled online-IDEs in our experiment, we ethically disclose the vulnerabilities to the service owners and developers. Furthermore, we ask for consent

---

*https://anonymous.4open.science/r/Bad-IDEa-1078
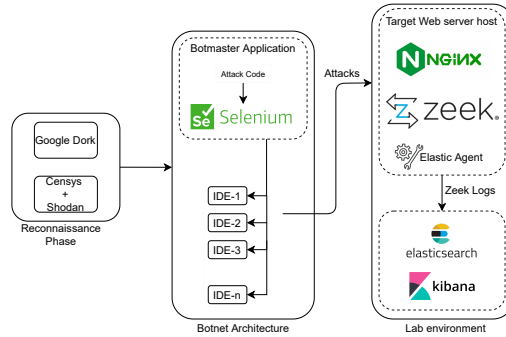
Fig. H.3: Methodology overview

from the owners to use the IDEs in our experiment and test our bot. We request consent for testing from 50, but end up having consent from 18 IDE owners and test the exploit by performing an HTTP-flood for five seconds from each online IDE. Furthermore, we create multiple instances of the online-IDEs to increase the magnitude of requests. All traffic to the webserver is logged and monitored with Zeek IDS. We limit the exploitation to IDEs that support the Python language. The IDEs are listed in the bot as described in section 4.2 and the HTTP flood code that targets our web server is executed. All incoming traffic is measured per IDE and time. Evaluating the maximum traffic capability of these individual IDEs is ethically challenging without compromising the availability of the underlying host and the network. Therefore, we perform a controlled execution of the experiments to ensure that the host's availability and the network are not compromised. Furthermore, we evaluate our approach and estimate the impact of the attacks.

5 Evaluation

5.1 Reconnaissance

This section summarizes our findings from the search for uncontrolled online-IDEs.

IDEs found

We search the Internet through our reconnaissance approach specified in Section 4.1 to find a total of 2269 online-IDEs of which 719 had uncontrolled execution. Most of the IDEs from the results supported more than one programming language. Figure H.4 shows the total number of IDEs classified based on their language support. As mentioned in the methodology, the IDEs were found through the reconnaissance process. We also

observed that most of the IDEs used multiple hosts for their backend based on the language chosen by the user, and some of them did not have any login or authentication from the user before program execution.
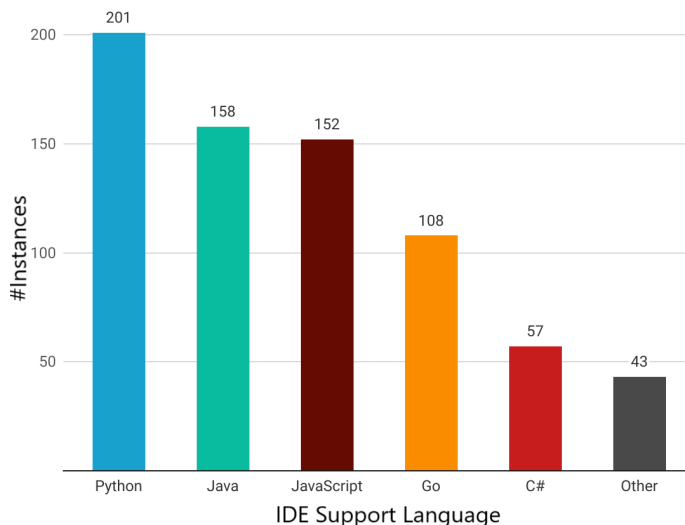
Fig. H.4: Classification by language support

Classification by use-type

Most of the online IDEs are development purpose-driven, where the user can set up a collaborative development workspace. We further classify the uncontrolled online-IDEs that we find in our reconnaissance based on their user type into an interview (24%), skill-training (22%), practice (23%), and collaborative development (31%) environments. We also find online-IDEs used as notebooks, where the user has an interactive environment with the possibility of importing datasets. Lastly, we find IDEs used by educational and training platforms that offer programming courses as a service.

Classification by uncontrolled-criteria

We define criteria for uncontrolled online-IDE in section 3.2 and classify the IDEs found during the reconnaissance phase. Figure H.5 shows the percentage of IDEs classified based on the criteria of uncontrolled execution of online-IDEs. We observe that most of the IDEs run on non-sandboxed environments, followed by unrestricted file operations and package imports. Furthermore, we find that 719 online-IDEs from the total of 2269 from our reconnaissance process satisfy all the criteria for uncontrolled execution. We

consider this a base for evaluating our experiment further on uncontrolled online-IDEs in availability attacks.
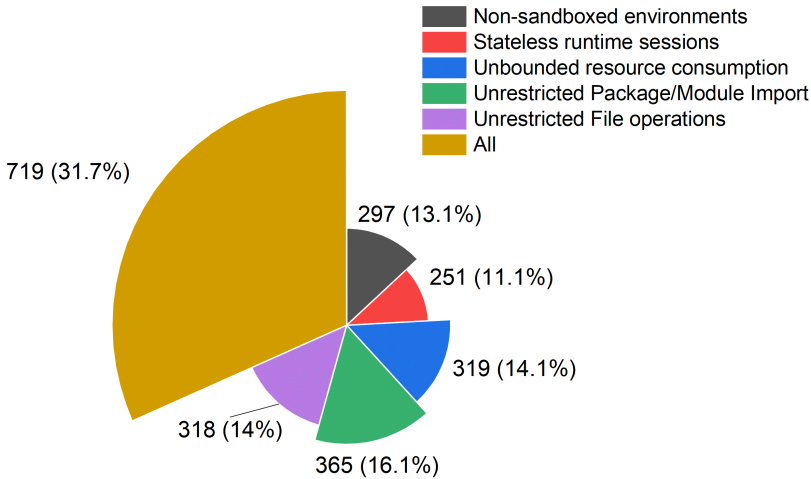
Fig. H.5: Classification by criteria for uncontrolled environment

5.2 Attacks & impact

We evaluate the possibility of leveraging the uncontrolled environments by performing controlled flood requests through the online-IDEs to target the webserver hosted at our lab facility. We used 18 of the total uncontrolled IDEs detected from our reconnaissance and performed code injection through the bot explained in section 4.2. The online-IDE environments that were used in our experiment did not have any user authentication or registration. Although we wanted to use as many IDEs identified in our search, we limited the number based on the consent we received for experimentation and did not cause any compromise in the availability of intermediary networking systems. Furthermore, we identified a range of 17 instances of uncontrolled online-IDEs from a reputed database provider during the reconnaissance process. As the number of instances was high, we immediately contacted the service owners about the potential misuse. Similarly, we disclosed the vulnerability to many critical operators so that the systems could be patched as quickly as possible and could not be used in our evaluation. In the following sections, we summarize the estimation, attacks, and impact of the requests sent from the uncontrolled IDEs.

Estimation

Performing DDoS attacks ethically over the Internet is challenging. To address this challenge, we follow an estimation-based approach to determine the impact of the attacks sourced from the IDEs. While there is existing work on mathematical modeling of DDoS attacks to predict the probability of resource depletion and bandwidth, to estimate the impact of the attacks from the IDEs, we refer to the method proposed by Balarezo et al. [23] for traffic-based models and specifically the *Queuing Model*. The *Queuing Model* uses a multidimensional approach that provides the probabilities for bandwidth, CPU, and memory exhaustion based on how networking elements process traffic. The ingress traffic measurements are carried out at periodic intervals of five seconds, and the values for the bandwidth, CPU, and memory are noted.

With the aim of developing a formula able to estimate the average attack magnitude of the architecture described in Section 4.2, we performed controlled HTTP-flood requests from the IDEs to our web server with a duration of 5 seconds, to avoid any potential disruption of the service. The experiment resulted in an average of 103 requests per IDE session, throughout all of the 32 IDEs, which proved vital in the formulation process. Figure H.6 shows the average number of requests from a single session of an IDE over time up to 60 seconds, calculated using Formula H.1. The figure also represents the estimated average number of requests possible from two *(n=2)* instances of an IDE running in parallel.

The variables taken into account when estimating the average total number of requests that can be achieved over a specific time interval are the number of *IDEs* used in the attack (I), the number of *sessions* per IDEs (S), the total *duration* of the attack in seconds (D), and the number of average *requests per second* for each IDE session (r). Combining all of these variables, we developed Formula H.1:

$$R_{Avg} = I * S * D * r \qquad \text{(H.1)}$$

Attack requests received on multiple IDE instances over time

We further estimate the number of requests possible from multiple IDEs with multiple instances running in parallel. Figure H.7 depicts the estimated average number of requests from multiple IDEs denoted by I and the number of instances of each IDE denoted by S. The experiment is performed using 18 IDEs from which we received consent. We estimate an average of 6 million requests possible with 32 IDEs with 32 instances over a minute using Formula H.1. The program that performs the HTTP floods is controlled by the number of threads performing the requests. We limit the number of threads to avert resource exhaustion on the IDEs.
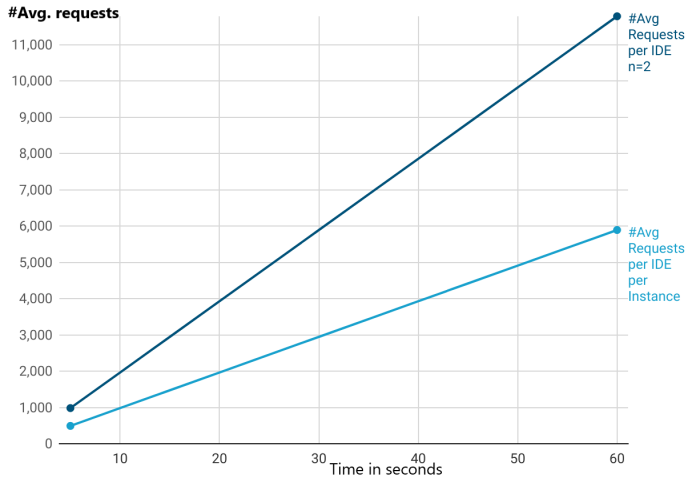
Fig. H.6: Estimated average requests per IDE instance by second
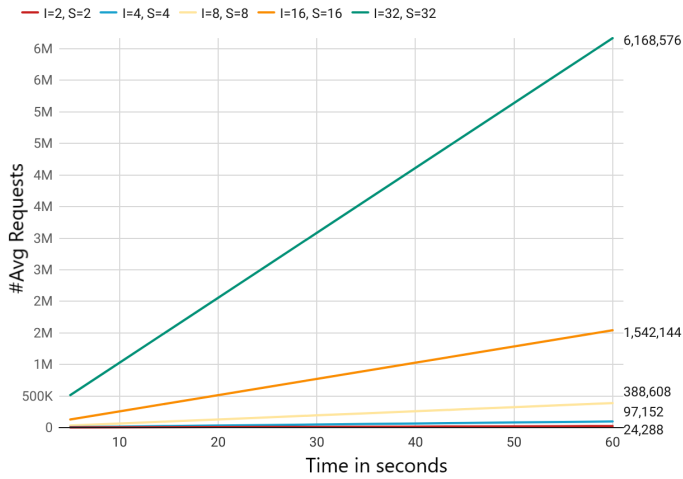
Fig. H.7: Estimated average requests received from multiple IDEs (I) and sessions (S)

Language-specific comparison

We compare the number of requests received from different languages supported by IDEs. The number of requests are obtained based on similar experiments that we carry out on Python-based IDEs. For the other languages, we perform the experiment with
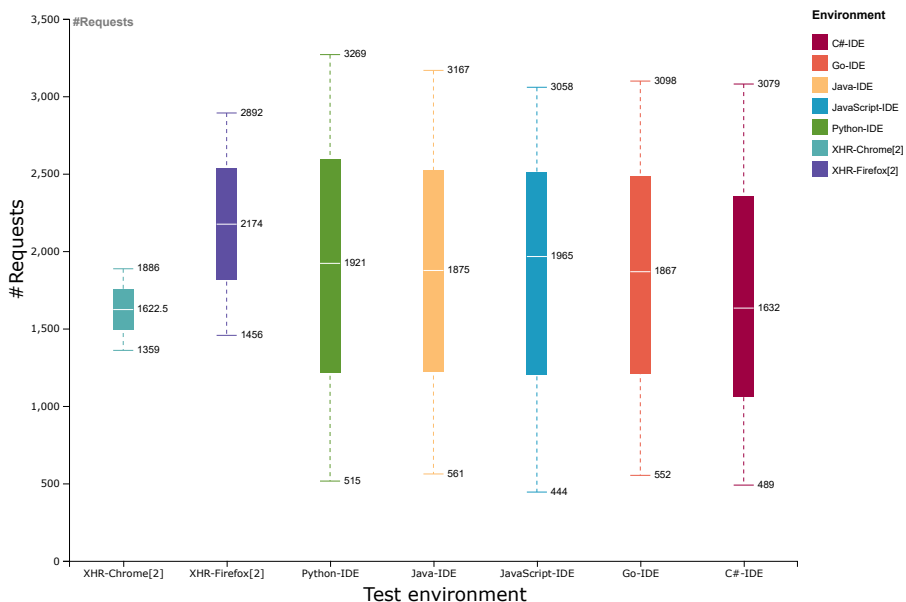
Fig. H.8: Average number of requests received from IDE-environments and a comparison with [2]

6 multi-language IDEs that we received consent for experimentation. Figure H.8 shows the number of requests received from different language supporting IDEs. We observe the highest number of requests from the Python supporting IDEs, in comparison with the other languages. To get a better understanding of the number of requests, we place the number of requests received from the Javascript program by Pellegrino et al. in the figure [2]. Note that this is not a direct comparison as the number of requests from the IDEs in our experiments were carried out for a period of 5 seconds and the method from Pellegrino et al. was recorded per second. However, we believe that by increasing the number of threads in our program can lead to similar results. In terms of economics, Pellegrino et al. use advertisements as a medium for executing the malicious embedded JavaScript on clicks, and hence incurs some costs. In our approach, we leverage accessible, open IDE execution environments with higher resources and negligible costs (zero) to execute the attacks.

Note that while in this evaluation, we evaluate the possibility of using IDEs that support the *Python* language, it is possible to achieve a higher number of requests by combining multiple IDEs that support other languages. While large botnets targeting DDoS attacks like the Meris Botnet have a significantly higher number of requests per second in comparison to our experiment, we believe that bots could employ vulnerable IDE instances armed with diverse attack types to increase the attack magnitude [24].

We further discuss the attack types and the impacts in the following section.

6   Discussion

Uncontrolled-IDE environments provide a degree of flexibility where the users can try performing varied attack types. Our experiments reveal that unfiltered networks of the IDEs allow different attack types. This section discusses some of the attack types that we try and describe the results.

6.1   Attack types

HTTP-Flood

We first evaluate our approach with HTTP-flood attacks from the IDEs. We execute the *Single Session HTTP Flood* to send a large number of requests from limited HTTP sessions. We observe that the CPU and memory of the victim (web server in our lab) are significantly depleted over the bandwidth of attacks received. A similar result was observed by performing *Single Request HTTP Flood* where multiple HTTP requests were made using a single session, masking them in a single packet. Programming languages offer multiple ways of creating HTTP requests. For example, the *Python* language offers the *requests*, *urllib* and *sockets* packages from which HTTP requests can be made. We experiment with all three variations of the packages and find approximately the same results with the maximum number of requests. However, we preferred to use the *sockets* package as it offered multiple options for setting the payload, and the max number of requests was achieved through controlled threading.

UDP-Flood

We try performing a controlled UDP-flood attack through the IDEs and find that some IDEs block UDP-based traffic. However, we were able to run UDP-based flood attacks from 18 IDEs in our experiment. We run the UDP-flood for a limited period of five seconds and observe the attack bandwidth ranging up to *1320 Mb/s* with the CPU load steadily increasing to an average of 22% per second. Figure H.9 shows the estimated bandwidth of attacks received over time from HTTP and UDP requests. The estimation is based on the attacks received during our controlled experiment. We find that UDP-based attacks caused a higher impact on the victim's resources, leading to quicker service disruption than the HTTP requests. Note that the attack bandwidth could be higher as we used a controlled test script in our experiment, and the requests originated from a single online-IDE instance.

Bandwidth in Mb/s
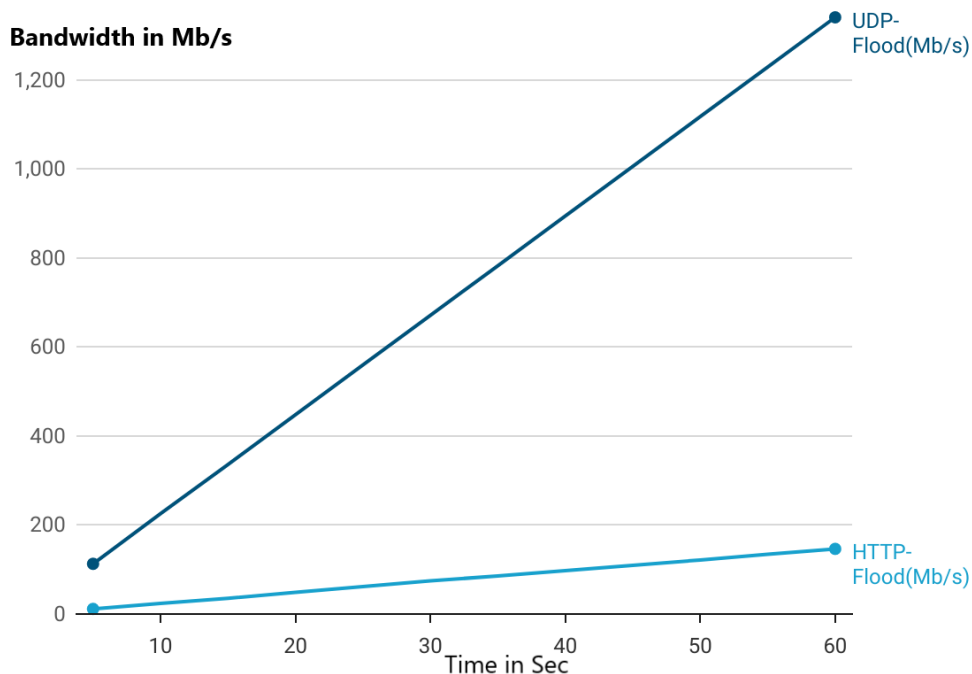
Fig. H.9: Estimated bandwidth comparison between HTTP and UDP flood

Multi-vector attacks

Multi-vector attacks involve using multiple flood-type attacks to achieve maximum bandwidth. While multi-vector attacks are ideal for achieving higher bandwidth by weaponizing, the supporting IDEs are a high-risk environment. We try the possibility of combining the HTTP and UDP-flood attacks from the IDEs. Our experiment shows that almost 55% of the IDEs are vulnerable to multi-vector attacks.

Text-encoding attacks

Boucher et al. [6] recently proposed a new type of attack in which the source code is maliciously encoded to appear different to a compiler than to a human. The authors present a proof of concept in multiple languages: Python, Java, JavaScript, Go, C#, C++, and Rust. We try the exploits with the IDEs to see if they support text-encoding standards like Unicode to manipulate the compiler-view. Our experiment revealed that 98% of the online-IDEs were vulnerable to the attack. We consider such vulnerabilities as potential discrete techniques for exploits. However, this is out of the scope of this

work as we emphasize to availability rather than integrity attacks.

Other findings

We performed our experiment with the online-IDEs that support *Python* language scripting. Other than the unrestricted package imports, we also found unpatched versions of systems that are vulnerable to CVEs CVE-2020-14422, CVE-2020-8492, CVE-2019-9674, CVE-2013-1753 that can lead to Denial of Service attacks [25]. Furthermore, as some IDEs allowed the OS and system packages, we could obtain information about the host operating systems and the other packages. We further check for vulnerabilities using the version information we obtain and inform the owners about the vulnerabilities. Furthermore, we find user-authenticated IDE environments, which require user-signup, equally vulnerable as non-authenticated ones, though we exclusively consider the unauthenticated platforms in our experiment.

6.2   Comparison with amplification attacks

Amplification attacks involve an amplification factor that enhances the original attack vector to multiply the initial attack. While these attacks are known to amplify over UDP-based protocols like DNS, more recently, there is research where TCP-based protocols can be leveraged [26]. In this work, we use vulnerable IDE instances to contribute to an existing attack process by potentially weaponizing uncontrolled online-IDE environments. Although the IDEs can contribute more attack requests, it is not similar to an amplification factor. Our approach implies that the attacks can be magnified in numbers by spawning multiple IDE instances and threads on the go without compromising the system or injection of malware. However, similar to DoS attacks, the attacker's identity remains hidden as the source of the attacks traced back to the IDEs.

6.3   Implications

Our work suggests that uncontrolled online-IDEs can be leveraged as an open system for availability attacks by botnets. Through our observation, we find that many online-IDEs possess high, scalable resources that can be exploited for carrying out attacks on the Internet and for crypto mining purposes. Although the number of uncontrolled online-IDEs is not as significantly high as the number of vulnerable devices employed by massive botnets, we believe that uncontrolled online-IDEs can be used as a magnitude factor since they are an accessible resource. While there is no evidence of bots using such environments for attacks, we proactively identify the vulnerabilities and ethically disclose them to the owners to prevent such exploitation by adversaries. Furthermore, the primary reason for uncontrolled execution is a result of misconfigured environments. As online-IDEs are considered as a platform for learning, many security implications are overlooked in order to achieve similarity to that of local IDE environments.

Using uncontrolled IDEs provide attackers with discrete ways of launching availability attacks. In comparison to other vectors used for attacks, uncontrolled IDEs provide the following advantages: *(i)* uncontrolled online-IDEs provide direct execution environments without any compromise steps to be undertaken by the attackers, *(ii)*, online-IDEs are equipped with reasonable resources that can facilitate attacks, *(iii)* multiple sessions of online-IDEs can be created to increase the magnitude of the attacks in case of ephemeral instances, *(iv)* uncontrolled online IDE environments are simpler to find on the Internet and do not require aggressive probing to find vulnerabilities, *(v)* since we observe no CAPTCHA checks in the IDEs on our findings, attackers can avoid any bypassing mechanisms that limit the botnets.

6.4   Limitations

Our approach utilizes online-IDEs on the Internet for availability attacks. As these IDEs are hosted on private infrastructure, evaluating availability attacks is ethically challenging. To address the ethical challenges, the evaluation of our approach involves some limitations. Firstly, we use a limited number of identified IDE instances to evaluate our methodology. This limits the full potential of the possible impact of the attacks. Second, we use rate-limiting in our test code to not cause any possible disruptions in the IDE service. This limits the use of the resources on the online-IDEs infrastructure. Third, we use an estimation-based approach to predict the possible number of attack requests per second achieved by multiple IDE instances running in parallel, which does not provide enough accuracy in the calculations and may have a high error rate. Lastly, we acknowledge packet drops occurring at intermediary devices in some of our experimental trials and discard environments that affect the overall throughput. Through this work, we intend to disclose the impact of running such environments to the owners and proactively prevent misuse of resources. It is a challenge to measure the impact of our approach in an ethical manner. We extrapolate the measurements received on a limited time-based experiment to accommodate the impacts to the IDE owners. We further acknowledge that many unknown factors may influence the values in our experiments. Our method is an honest attempt to identify uncontrolled IDE environments and prevent their misuse.

7   Ethical considerations & countermeasures

It is challenging to test our methodology as it involves sending high traffic from the Internet that may disrupt availability. We follow several precautions to avoid such a scenario. In this section, we discuss the ethical considerations followed in our approach.

7.1   Attack testing

We follow multiple steps in our attack testing approach to ensure that the availability
of the online-IDEs or the intermediary networking systems are not compromised. The
ethical measures we follow in our methodology are summarized below.

Informed consent

We take consent from the online-IDE owners to perform our experiment. We obtain
consent from 18 IDE owners to perform our experiments. We assure the IDE owners of
non-malicious experiments and measures to prevent resource exhaustion. Furthermore,
we test a limited number of IDEs in our experiment, although we find many vulnerable
uncontrolled online-IDEs.

Limited threads (rate-limiting)

We run an HTTP-flood program to test the possibility of achieving maximum requests
from the IDEs. However, we limit the number of threads in our program to prevent
resource exhaustion. We use an estimation-based approach to ethically predict the
number of requests that can be achieved from the IDEs.

Lab infrastructure for testing

We set up a web server in our lab infrastructure to target all the requests from the IDEs.
However, we understand that this does not fully comply with the challenge of reducing
the disruption in the network due to the traffic from the IDEs. We try to reduce the
disruption by limiting the number of threads and the runtime of the experiment. The
website used for measuring the HTTP flood requests received from the IDEs contained
the necessary information about our experiment.

7.2   Responsible disclosure

We perform responsible disclosure to all the owners of identified uncontrolled online-IDE
execution environments. The disclosure informs the owners of the importance, criteria,
vulnerabilities, and proof of concept to test the environments independently. Further-
more, we perform an early disclosure to certain critical service providers (for example, a
leading database service) that have a high possibility of traffic, even if this entailed the
possibility of not using these environments for testing our approach. Additionally, we
ask for IDE owners' consent to experiment on uncontrolled environments before they
patch their systems. Until the time of submission of this paper, we hope that most of
the uncontrolled IDE environments are patched.

7.3 Countermeasures

In this section we propose and discuss countermeasures against the criteria defined for uncontrolled execution environments.

Restricted file operations

File operations are essential for the import and export of data. However, it is crucial to restrict the operations and limit access to critical paths of the file system by simply employing containerized environments. Many online-IDEs use file operations for importing the source files; it is also essential to perform validation to scan for potential malware. An adversary can leverage unrestricted operations to either download malware or spread the malware to external systems.

Limited package support

Adversaries can use packages to perform malicious operations on the host machine, like downloading malware, accessing the host's file system, and scanning the network. Developers use packages to support additional operations or import external libraries not part of the default package list. It is also crucial to limit the features of default libraries (for example, the sockets package) to restrict access to the network and limit the import of external libraries. While we acknowledge that limiting the functionality of libraries is a hard problem, we suggest to limit the attacks that leverage packages, by configuring network rate limiting in addition to memory and CPU resource limiting. Linux environments provide default tools for limiting the system resources per process. Administrators can further use containerization of individual user sessions to limit resource usage.

Bounded resource consumption

Limiting the resources per user and program is required. Unlimited resources can lead to disruptive operations on the host and be leveraged as an attack source. Also, limiting the number of threads that can be created ensures controlled resource usage. Furthermore, the use of timeouts that restrict the execution period ensures that a program does not run for extended periods and prevents flood-type attacks.

Sandboxed environments

Sandboxed environments ensure no access to external systems, and the user sessions are isolated from the other sessions running on the online-IDE. Each user has a dedicated isolated environment that is purged after the user session expires. Online-IDEs can leverage containerized environments to achieve sandboxing of individual user sessions and purge them after the end of the session.

Stateful user sessions

Online-IDEs run over a web service and can be configured to maintain stateful user sessions to track idle or disconnected users for stopping the program execution. This prevents bots from spawning multiple IDE instances to inject malicious code and exit the session to save resources on the bot client. Maintaining stateful user sessions can also help limit the number of sessions per user and limit resource usage.

Other measures

We accessed the IDEs through the Tor network and found that 98% of the online-IDEs identified allowed access. To limit suspicious events, we suggest the use of *CAPTCHAs* to verify the source of traffic and also limit the execution of suspicious code. We further suggest online-IDEs to integrate *CAPTCHAs* for validating each user session irrespective of the network to limit the bot activity. Moreover, we strongly recommend that all the online-IDEs have user authenticated sessions to prevent unnecessary resource usage. Lastly, to defend against text-encoding attacks, we recommend following the countermeasures suggested by Boucher et al. [6] to prohibit the support for text directionality control characters both in language specifications and in compilers implementing these languages.

8   Conclusion

This work identifies online-IDEs that offer uncontrolled execution environments that can be leveraged to perform availability attacks. We perform an Internet-wide search for online-IDEs and filter them by executing a test script that satisfies the uncontrolled execution criteria. Furthermore, we perform experiments to verify the possibility of availability attacks through the online-IDEs by informed consent. The estimated impact of the attacks is calculated by measuring the requests obtained from the experiments. We emphasize the consequences of having uncontrolled execution environments and proactively conduct experiments to assess the impact through this work. Lastly, we perform immediate ethical disclosure to the IDE owners to prevent misuse of the environment. As future work, we plan to generate scripts that can be used for checking uncontrolled execution of online-IDE environments such as permissions set for code execution, maximum file size that can be written, paths accessed, and max network bandwidth to help the administrators.

References

[1] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the mirai botnet," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1093–1110. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis

[2] G. Pellegrino, C. Rossow, F. J. Ryba, T. C. Schmidt, and M. Wählisch, "Cashing out the great cannon? on Browser-Based DDoS attacks and economics," in *9th USENIX Workshop on Offensive Technologies (WOOT 15)*. Washington, D.C.: USENIX Association, Aug. 2015. [Online]. Available: https://www.usenix.org/conference/woot15/workshop-program/presentation/pellegrino

[3] ENISA. (2020) Enisa threat landscape 2020 - malware. ENISA. [Online]. Available: https://www.enisa.europa.eu/publications/malware

[4] P. Wainwright and H. Kettani, "An analysis of botnet models," in *Proceedings of the 2019 3rd International Conference on Compute and Data Analysis*, 2019, pp. 116–121.

[5] R. Saini, S. Bali, and G. Mussbacher, "Towards web collaborative modelling for the user requirements notation using eclipse che and theia ide," in *2019 IEEE/ACM 11th International Workshop on Modelling in Software Engineering (MiSE)*, 2019, pp. 15–18.

[6] N. Boucher and R. Anderson, "Trojan Source: Invisible Vulnerabilities," *Preprint*, 2021. [Online]. Available: https://arxiv.org/abs/2111.00169

[7] Z. Alizadehsani, E. G. Gomez, H. Ghaemi, S. R. González, J. Jordan, A. Fernández, and B. Pérez-Lancho, "Modern integrated development environment (ides)," in *Sustainable Smart Cities and Territories*, J. M. Corchado and S. Trabelsi, Eds. Cham: Springer International Publishing, 2022, pp. 274–288.

[8] L. Wu, G. Liang, S. Kui, and Q. Wang, "Ceclipse: An online ide for programing in the cloud," in *2011 IEEE World Congress on Services*. IEEE, 2011, pp. 45–52.

[9] P. Chinprutthiwong, R. Vardhan, G. Yang, Y. Zhang, and G. Gu, "The service worker hiding in your browser: The next web attack target?" in *24th International Symposium on Research in Attacks, Intrusions and Defenses*, ser. RAID '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 312–323. [Online]. Available: https://doi.org/10.1145/3471621.3471845

[10] J. Liu, Z. Zhao, X. Cui, Z. Wang, and Q. Liu, "A novel approach for detecting browser-based silent miner," in *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*. IEEE, 2018, pp. 490–497.

[11] J. Rüth, T. Zimmermann, K. Wolsing, and O. Hohlfeld, "Digging into browser-based crypto mining," in *Proceedings of the Internet Measurement Conference 2018*, ser. IMC '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 70–76. [Online]. Available: https://doi.org/10.1145/3278532.3278539

[12] D. Georgoulias, J. M. Pedersen, M. Falch, and E. Vasilomanolakis, "A qualitative mapping of darkweb marketplaces," in *Symposium on Electronic Crime Research (eCrime)*. IEEE, 2021.

[13] L. Wu, G. Liang, and Q. Wang, "Program behavior analysis and control for online ide," in *2012 IEEE 36th Annual Computer Software and Applications Conference Workshops*, 2012, pp. 182–187.

[14] V. Kiriansky, D. Bruening, and S. Amarasinghe, "Secure execution via program shepherding," in *11th USENIX Security Symposium (USENIX Security 02)*. San Francisco, CA: USENIX Association, Aug. 2002. [Online]. Available: https://www.usenix.org/conference/11th-usenix-security-symposium/secure-execution-program-shepherding

[15] M. Pass. (2021) Icecoder. [Online]. Available: https://github.com/icecoder/ICEcoder

[16] Microsoft. (2022) Visual studio code. [Online]. Available: https://github.com/microsoft/vscode

[17] Coder. (2022) Code-server. [Online]. Available: https://github.com/coder/code-server

[18] L. Siira. (2022) Atheoside. [Online]. Available: https://github.com/Atheos/Atheos

[19] J. Zhang, J. Notani, and G. Gu, "Characterizing google hacking: A first large-scale quantitative study," in *International Conference on Security and Privacy in Communication Networks*, J. Tian, J. Jing, and M. Srivatsa, Eds. Cham: Springer International Publishing, 2015, pp. 602–622.

[20] Censys, "Censys search," 2021. [Online]. Available: https://censys.io/

[21] SHODAN, "Shodan," 2021. [Online]. Available: https://www.shodan.io/

[22] A. B.V. (2022) Ace editor. [Online]. Available: https://github.com/ajaxorg/ace

[23] J. F. Balarezo, S. Wang, K. G. Chavez, A. Al-Hourani, and S. Kandeepan, "A survey on dos/ddos attacks mathematical modelling for traditional, sdn and virtual networks," *Engineering Science and Technology, an International Journal*, vol. 31, p. 101065, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2215098621001944

[24] CloudFlare, "Meris botnet," 2021. [Online]. Available: https://blog.cloudflare.com/meris-botnet/

[25] C. Details, "Cve details," 2021. [Online]. Available: https://www.cvedetails.com/vulnerability-list/vendor_id-10210/product_id-18230/year-2020/opdos-1/Python-Python.html

[26] M. Kührer, T. Hupperich, C. Rossow, and T. Holz, "Hell of a handshake: Abusing TCP for reflective amplification DDoS attacks," in *8th USENIX Workshop on Offensive Technologies (WOOT 14)*. San Diego, CA: USENIX Association, Aug. 2014. [Online]. Available: https://www.usenix.org/conference/woot14/workshop-program/presentation/kuhrer

Part III

Epilogue

Conclusion

Adversaries constantly develop new attack tactics and techniques to evade security systems and cause significant damage, causing an increasing reliance on defense-based detection systems. The detection of novel attack techniques is challenging with the changing threat landscape. Our studies show that cyberattacks have been modeled to be more intelligent and discreet. The conclusion from the empirical analysis is outlined in the following.

Cyber deception systems are vital in capturing attacks and facilitating further analysis. Deception is a tactic that both adversaries and defenders leverage in their strategies. The value of cyber deception systems like honeypots is based on their ability to deceive adversaries and capture their attack techniques. Although honeypots are a good resource for gathering attack data, they are vulnerable to detection attacks. This thesis addresses the pitfalls of honeypots to enhance their value by improving their deception and capturing capabilities.

The thesis follows a two-phase approach to the research on improving and evaluating the use of cyber deception systems. In the first phase, state-of-the-art honeypots are surveyed and tested for endurance against fingerprinting attacks. A combined, multistage fingerprinting framework is proposed that covers a wide scope of honeypots and has a better detection rate than the state-of-the-art. Furthermore, novel honeypot fingerprinting techniques are proposed, which are a first in the area. In the second phase, RIoTPot, a hybrid-interaction and modular honeypot is proposed to address the limitations of limited interaction and extensibility. The honeypot is evaluated by performing a longitudinal study that aims to assess the impact of the operational parameters of a honeypot, like interaction level, simulation environment, deployment infrastructure, and geo-location. Our study further evaluates the honeypot to understand the attack landscape on the directory services targeted extensively. Overall, the findings from the thesis can be summarized as follows.

Attacks against cyber deception (RQ1 and RQ2): Our analysis of the state-of-the-art deception-based systems against fingerprinting attacks reveals that many open-source honeypots are vulnerable and can be detected. Moreover, we observe that there

is still a gap in the design, modeling, and implementation of honeypots based on the purpose. Our studies' results indicate that many misconfigured open-source honeypots are exposed to the Internet, which can be detected with less effort, and administrators are passive towards these deployments. Recent research reveals a trend in honeypot fingerprinting and entails that honeypot developers and administrators must maintain their deployments. Although honeytokens are low maintenance and risky, they mustn't be detected by adversaries. Our work shows the gaps in implementation from a popular honeytoken service and presents methods to detect them using passive techniques. We follow ethical guidelines, disclose our findings to the responsible honeypot and honeytoken developers, and attempt to contact the administrators of the misconfigured deployments. Overall, this research presents effective techniques to detect honeypots and honeytokens that can be considered for improvement and advancement in their deceptiveness.

Cyber deception (RQ3):   We consider open-source deception-based systems, mainly honeypots, in our empirical analysis. Our analysis reveals multiple gaps in these honeypots where they are either poorly maintained or simulate specific services. As a result, most of these honeypots are modeled as low or medium interaction entailing limited simulation. In this thesis, we aimed to bridge this gap by proposing RIoTPot (Paper D), a honeypot that offers hybrid interaction and modular architecture. The hybrid interaction breaks the binary interaction-level paradigm by providing flexibility in simulating services on multiple levels. Furthermore, the modular architecture facilitates operating high-interaction honeypots and provides extensibility to add more services. This contribution addresses limitations arising from the binary interaction paradigm and the limited simulation levels. We evaluate RIoTPot by conducting both targeted (Paper E) and longitudinal studies (Paper F) and studying the attack landscape. Our results indicate that running honeypots in high-interaction levels received higher exclusive traffic than low and medium-interaction levels and the influence of operational parameters.

Internet security measurements (RQ3):   The measurement studies conducted during the thesis provide insight into the attack landscape and an overview of the prevailing threats. The studies were focused on studying the attacker methods and identifying the potential threats by finding misconfigured services. The studies had two facades that followed a measurement-based approach. The first followed a measurement-based approach to scan the Internet to find misconfigured services, and the second to analyze the attacks gathered on honeypots. The resulting datasets from the honeypot and the Internet measurements studies aim toward the thesis's goal to create datasets that are valuable to the threat research community. Our findings from the Internet scans reveal a large number of misconfigured services span across IT, IoT, and the OT landscape. The measurements and analysis from the honeypot studies provide extensive

insight into the attack trends and the landscape, including targeted attacks, multistage attacks, and pivot attacks. Internet security measurements provide compelling datasets for the analysis of threat research. We want to continue our work in this area by using all the lessons from this thesis on improving the scanning scope, removing false positives, and characterizing the end systems.

Reflections I want to assert that the thesis and the research mindset expanded my knowledge horizons. Working with deception-based systems is complex due to varied perspectives. Several challenges were encountered and addressed during the thesis. I present some of my selective reflections on these challenges.

1. **Honeypot Fingerprinting:** In the research on honeypot fingerprinting, I realized that characterizing and profiling end systems with limited data is challenging. This challenge becomes a paradox, especially if the goal is to determine if the end system has a deceptive layer. The state-of-the-art techniques are limited in scope and, most of the time, do not apply to active deception systems.

2. **Honeypot operation and studies:** Running and operating honeypots follow the complexity curve of Internet scanning. While professional services are operating Internet-scanning (Shoda, Censys) and Honeyfarms (Greynoise) and make it look effortless, it is bewildering as a researcher from the University. There are constant challenges from the SOC teams, ISPs, and IT about running vulnerable services and the entailing threats (even after informing). IP churning is common for Internet scanning and operating honeypots to preserve freshness. The limited infrastructure, resources, and authoritative permissions can be taxing for a University researcher. This is one of my hardest battles during this thesis, making the resulting datasets valuable. Another challenge with developing deception-based systems is the psychological aspect of sequentially thinking with a hacker mindset and as a defender. This aspect is underrated and crucial for developing effective deception-based systems.

3. **Internet scanning:** Internet scanning is more challenging and complex than it looks, especially in a privacy-centered region like Europe. In addition, it is common to get flagged by the authorities, ISPs, CERTs, and most importantly, the IT Team of the university. I would emphasize that even with profound coordination, informed consent, and disclosure, the scanning process always has limitations. The lessons learned from the Internet scanning process are extensive, and we quickly tried developing alternate approaches with less noisy techniques to achieve our goal. My study abroad at the University of Cambridge and the valuable guidance from Dr. Richard Clayton helped enormously with Internet scanning. We aim to leverage this skill to continue our Internet scans and expand our research in this area.

4. **Data analysis:** The amount of data gathered from the studies can be overwhelming. With such large datasets, filtering the noise before triage is very important. However, with honeypots, every interaction can be considered suspicious; hence, implementing filters and discarding information can be tricky. Furthermore, working with large datasets requires persistent, scalable storage, analysis, and visualization systems. It is essential to plan and model a resilient architecture for the research that fits the study. Lastly, creating labeled datasets is valuable for future research; planning and schematizing for long-term maintenance are necessary.

5. **Data sharing and collaboration:** The GDPR makes it highly challenging to share the resulting datasets from the research as they contain information on vulnerable systems, honeypots, or attack sources that have to be pseudo-anonymized because of IP addresses, thereby limiting collaboration. As this data cannot be publicly shared, following a strict embargo process is essential. We received numerous collaboration requests and have successfully pursued them by finding a way to collaborate with our datasets.

The Interreg COM^3 Project (Thesis funding project): The European Interreg COM^3 project funded this thesis. The thesis contributed to its goal of creating cybersecurity in small and medium enterprises. The contributions were in the form of awareness courses designed with a focus on Cybersecurity awareness through honeypots, Data Breaches and Privacy leak awareness, and Threat Identification & Self-Assessment for SMEs. The courses are developed to suit the online learning format and are integrated into the project outcome. Furthermore, many awareness programs were undertaken for outreach during the project. The findings from the papers are used to create awareness of the impact of cyberattacks. However, the pandemic imposed limitations on outreach activities. I understood the challenges of SMEs in rural regions and the knowledge gaps that exist in the top-level management of enterprises. Working with international partners in the project gave me a broader perspective of the cybersecurity challenges persisting in SMEs.

Summary: Cyber deception is an effective tool to study the attacks and further can be used for characterizing the attack source and techniques. While it may be less effort to simulate any service or a target, it is important to configure it to suit the goal of the deployment. As with the definition of deception-based systems, the goal is to deceive the attackers into capturing the attack activity for further analysis. However, we emphasize that the adversaries must not be undermined as their techniques evolve constantly. It is crucial to keep pace with this evolution to advance the effectiveness of deception-based systems. Furthermore, it is imperative to have a collaborative platform for sharing novel threats and the data gathered to help the threat research community. We acknowledge the efforts of The Honeynet Project towards aggregating attack data collected from

several honeypot deployments and making it available to researchers.

One of the most effective methods of protection against cyberattacks is by creating security awareness. The Interreg COM^3 project that funded this thesis aimed at creating cybersecurity awareness programs for small and medium enterprises. Through this thesis, we created awareness programs by introducing honeypots for detecting and presenting a realistic experience of cyber attacks. The results from the studies performed in this thesis helped foster an emphasis on cybersecurity measures in enterprises. We acknowledge the Interreg COM^3 project for the funding and for taking an active step in creating cybersecurity awareness.

As there is time complexity (3 years) involved with this thesis, some of the work is still in its development phase. The ideas include leveraging the attacks (labeled dataset) gathered during the experiments to train ML models for effective threat detection and setting up a platform that acts as a malware repository for researchers that provides samples based on filters on protocols and services. Meeting fellow researchers at academic conferences always fueled interesting ideas and enough motivation to pursue more profound research. One avenue we wanted to explore is diving into the human side of the attackers through experiments and the gathered attack dataset. One of the many outcomes of this thesis is the artifacts and tools produced (open-source) that can help researchers to conduct studies to understand diverse threat landscapes. Lastly, our study promotes deception-based systems for proactively detecting attacks and constantly gathering attack data on malicious actors.

"There is nothing more deceptive than an obvious fact." - Sir Arthur Conan Doyle

Directions for Future work

The thesis statement aimed to empirically analyze cyber deception systems to identify their value and the limitations that undermine them. However, in this thesis, our scope was limited to honeypots and honeytokens. Honeypots are an effective strategy to gain new insights and facilitate threat research. They can be used to collect valuable data that helps model attacker behavior. However, determining the right level of deception and scope is crucial.

The idea of honeypots can be explored further to extend their bounds and capabilities as an integrated component. Honeypots offer flexibility that can be leveraged to analyze the attack landscape in diverse areas. Deceptive decoys can be beneficial in either modeling or integrating the essence of honeypots into existing systems without actually using a whole system. Honeytokens are an excellent example of the essence of cyber deception in a minimal form. They can be further extended to include other digital entities. Other deception-based systems, like moving target defense, hold great value and can be explored further regarding their applicability. Honeytokens are emerging research because of their simplified operation. As a result of our research, we can suggest areas for improvement in how honeytokens can be improved to gather actionable information.

Research in Counter-deception techniques is necessary to siphon the development of defensive deception techniques. Fingerprinting research helps understand the adversarial mindset, which is beneficial in enhancing deception-based systems. With the increase in cyber threat intelligence public databases and datasets, passive fingerprinting has a broad scope.

One of the core contributions of the thesis is the creation of datasets derived from our studies. These datasets are valuable and form the ground for future research in threat analysis. The datasets could be used with training from machine-learning models for effective threat classification and research. Furthermore, the datasets can be used for extending AI-based detection systems. Overall, honeypots and honeytokens provide researchers with a labeled dataset, creating avenues for further research.

Deception is a legacy misleading technique. Over the years, living things have evolved this technique and applied it in many forms. The art of deception will continue evolving, and defensive security teams must try to stay ahead with effective deception techniques.