Gotta catch 'em all

A Multistage Framework for Honeypot Fingerprinting

Srinivasa, Shreyas; Pedersen, Jens Myrup; Vasilomanolakis, Emmanouil

Published in:
ACM Digital Threats: Research and Practice

DOI (link to publication from Publisher):
10.1145/3584976

Publication date:
2023

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

Citation for published version (APA):
Srinivasa, S., Pedersen, J. M., & Vasilomanolakis, E. (2023). Gotta catch 'em all: A Multistage Framework for Honeypot Fingerprinting. *ACM Digital Threats: Research and Practice*, *4*(3), 1-28. Article 42. https://doi.org/10.1145/3584976

Gotta Catch ’em All: A Multistage Framework for Honeypot Fingerprinting

SHREYAS SRINIVASA and JENS MYRUP PEDERSEN, Aalborg University
EMMANOUIL VASILOMANOLAKIS, Technical University of Denmark

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 past years, several 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 several 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. Last, we discuss countermeasures against honeypot fingerprinting techniques.

CCS Concepts: • **Security and privacy** → **Network security**; **Intrusion detection systems;**

Additional Key Words and Phrases: Honeypots, fingerprinting, honeypot attacks, honeypot detection, honeypot evasion

ACM Reference format:
Shreyas Srinivasa, Jens Myrup Pedersen, and Emmanouil Vasilomanolakis. 2023. Gotta Catch ’em All: A Multistage Framework for Honeypot Fingerprinting. *Digit. Threat. Res. Pract.* 4, 3, Article 42 (October 2023), 28 pages.
https://doi.org/10.1145/3584976

---

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 [38]. Honeypots are not a stand-alone security mechanism but rather important supplements to existing infrastructure (e.g., firewalls and intrusion detection systems). Nevertheless, they offer a unique attack understanding and perspective while exhibiting a very low number of false positives.

This research was supported as part of COM3, an Interreg project supported by the North Sea Programme of the European Regional Development Fund of the European Union.

Authors’ addresses: S. Srinivasa and J. M. Pedersen, Aalborg University, A. C. Meyers Vænge 15, 2450 København, Denmark; emails: {shsr, jens}@es.aau.dk; E. Vasilomanolakis, Technical University of Denmark, Anker Engelunds Vej 1 Bygning 101A, 2800 Kgs. Lyngby, Denmark; email: emmva@dtu.dk.

2576-5337/2023/10-ART42 $15.00
https://doi.org/10.1145/3584976

The latter is due to the fact that any communication toward a honeypot is considered hostile—that is, 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 [29]. The first two categories offer different levels of *emulation* of protocols, whereas 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 outside 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 [10] and Cowrie [30]), Telnet (e.g., Cowrie [30]), HTTP (e.g., Glastopf [34]), FTP, SMB (e.g., Dionaea [42] and HosTaGe [44]), and also **Industrial Control System (ICS)** protocols like Modbus and S7 (e.g., Conpot [35] and HosTaGe [45]). Other research-based honeypots include AmpPot [25] that simulates UDP-based protocols like NTP and SSDP, which can be abused for DRDoS attacks, and RIoTPot [40] 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 [54]) 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 [6]) 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 several 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 nine prominent honeypot implementations. Our results come as an independent confirmation of previous studies [28, 47] 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. Last, 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 state-of-the-art and third-party (so-called metascan) 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 fewer 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 [37]. 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 [36] 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 the 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.

2.1.1 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 [26], XProbe2 [3], Metasploit [21], and Hydra [43] 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 TCP (Transmission Control Protocol) 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 system level. The integration of the information received from different levels improves detection accuracy.

2.1.2 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 [36] and Censys [12]) to obtain attributes like hosting provider and the 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 ICSs 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

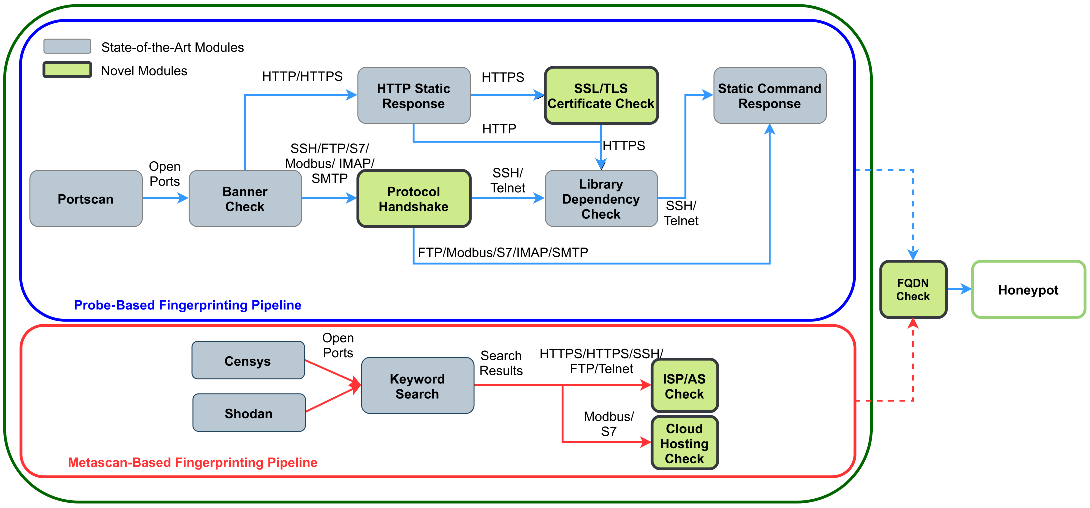

Fig. 1. Multistage framework for honeypot fingerprinting.

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, whereas 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 [12] uses the Recog engine [33] to detect device types using the information received by probing. The methods used in the proposed metascan pipeline are novel specifically toward 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 false positives from the results. We term the target system considered for fingerprinting as an *instance* for the rest of our article.

Figure 1 shows the proposed multistage fingerprinting framework. The framework contains two independent main pipelines of probe-based and 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, whereas the green boxes represent the novel methods. The novel methods are composed 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.

2.2.1 *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 that 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 [13] for this process (alternatively one could use Masscan [17]). 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 [24]. 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 [53]. State-of-the-art honeypot fingerprinting techniques by Vetterl and Clayton [47] and Morishita et al. [28] employ banner-based fingerprinting to detect honeypots. We combine this knowledge (see Table 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 3.3.4 in the evaluation). Fingerprinting honeypots only with banner checks is prone to false positives, and therefore we subject the instances to further protocol- 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 that 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 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 other works [28, 47] 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. Although 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 [31]. 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

Table 1. Protocol Handshake Deviation

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”, “0300002102f08032070000000000008 \n 00080001120411440100ff09000400110001”\n	0 × 32
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 2. Library References in Honeypots

Honeypot	Protocol	Library	Updated
Kippo	SSH	TwistedConch	May 2015
Cowrie	SSH	TwistedConch	May 2018
MTPot	Telnet	Telnetsrv	Dec. 2012
Cowrie	Telnet	TwistedConch	May 2018
Dionaea	HTTP	Custom	Sept. 2016
Glastopf	HTTP	BaseHTTPServer	Oct. 2016
Conpot	HTTP	BaseHTTPServer	March 2018

that attempt to establish a connection through limited parameters and observe the response for deviation for all emulated services. Table 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. Furthermore, some libraries referred by honeypots have not been well maintained. Vetterl and Clayton [47] have leveraged the use of libraries in honeypots to craft specific probes that return static values. This static information can be used to fingerprint the honeypots. Table 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 the work of Vetterl and Clayton [47] 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, the 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. Later, Table 14 shows the static response returned by honeypots for specific commands by our probes.

2.2.2 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 state of the art (e.g., [28]) 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 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 13 in the appendix 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 even 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.

2.2.3 FQDN Check. An 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 DomainTools, which provides an extensive database for WHOIS information [11]. 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, whereas 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.

2.2.4 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- and 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 3. The choice of honeypots is based on several factors. First, these honeypots are considered some of the most popular ones and most frequently deployed (e.g., see the ENISA recommendations in the work of Grudziecki et al. [18]). 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., [28, 47, 52]). Last, 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- 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. Last, 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 3) in a lab environment and test all probes that 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 1) of the probe-based phase. Following these tests, we evaluate the multistage framework against the known honeypot instances in the lab environment. All 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 [13] to scan a total of 2.9 billion IP addresses.[1] Our tests follow the flow of

---

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

Table 3. Honeypots Tested in Our Internal Lab Environment

Honeypots	Ports and 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 [51]
MTPot	Ports: 23 Services: Telnet	Base [8]

Figure 1. In other words, we first perform a probe-based scan and afterward perform an independent metascan by making use of Shodan and Censys [12, 36]. Our experiments were conducted in a period of 6 months. The experiment is carried out as three 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 6 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 state of the art (e.g., [28, 47, 52]). Hence, both our article and all existing related works are prone to false negatives.

3.3 Results

By first performing a ZMap scan, we derive Table 4, which 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 1.

3.3.1 Honeypot Identification. Overall, the framework detected a total of 21,855 honeypots. Figure 2 shows the honeypot instances detected over three sequential scans over a period of 6 months. Figure 2 also depicts the change in honeypot instances detected over the three scans. The instances of honeypots Gaspot, Conpot, and Amun (HTTP) were detected more in the third scan, whereas the others remained constant or reduced. This could be because of honeypots instances undergoing 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

Table 4. Number of Identified Instances and Protocols/Ports

Protocol	Port	No. of Systems on the Internet (in Millions)
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

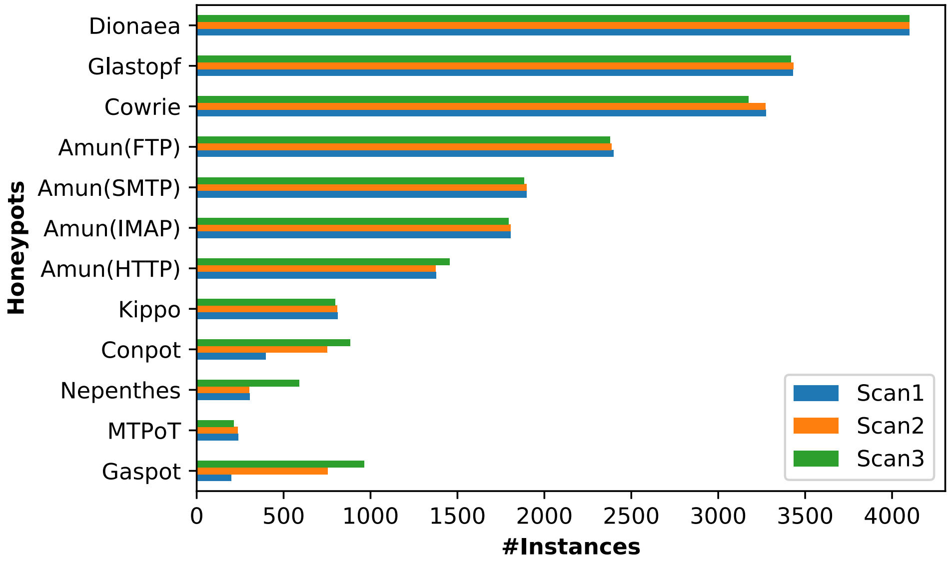

Fig. 2. Honeypots detected per scan.

technique. Figure 3 summarizes the honeypots detected by probe- 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 4 compares our findings with the state-of-the-art measurements from Vetterl and Clayton (*Bitter Harvest*) [47], Morishita et al. (*Detect Me*) [28], and Zamiri-Gourabi et al. (*Gas What?*) [52]. The figure shows the total honeypot instances detected by the state of the art in comparison to our approach. We note that the honeypots Nepenthes and Amun were not evaluated by Vetterl and Clayton [47]; in addition, Zamiri-Gourabi et al. [52] 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 several interesting findings. First, they independently confirm previous studies' conclusions with regard to the global (poor) state of honeypot deployments [47]. Second, our results come more than 1 year after the aforesaid studies: this provided

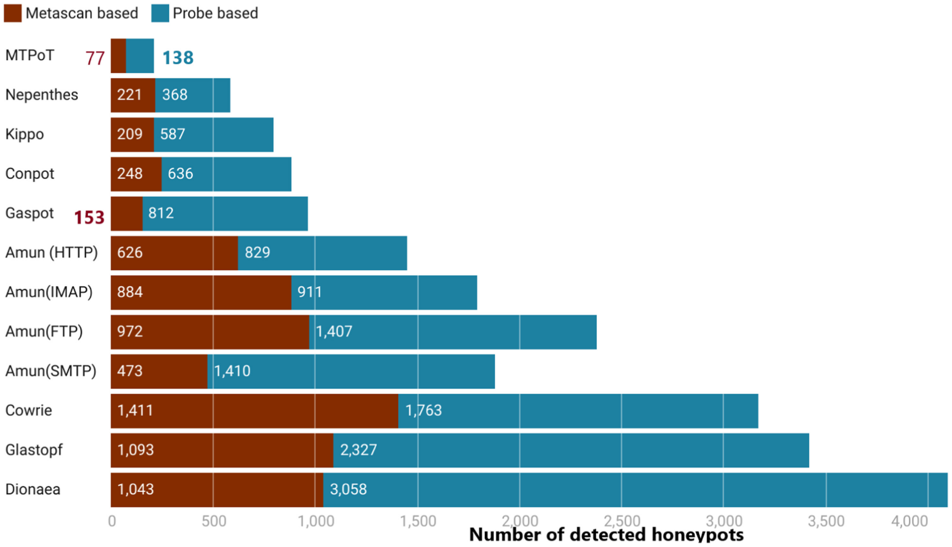

Fig. 3. Honeypots detected by type and technique.

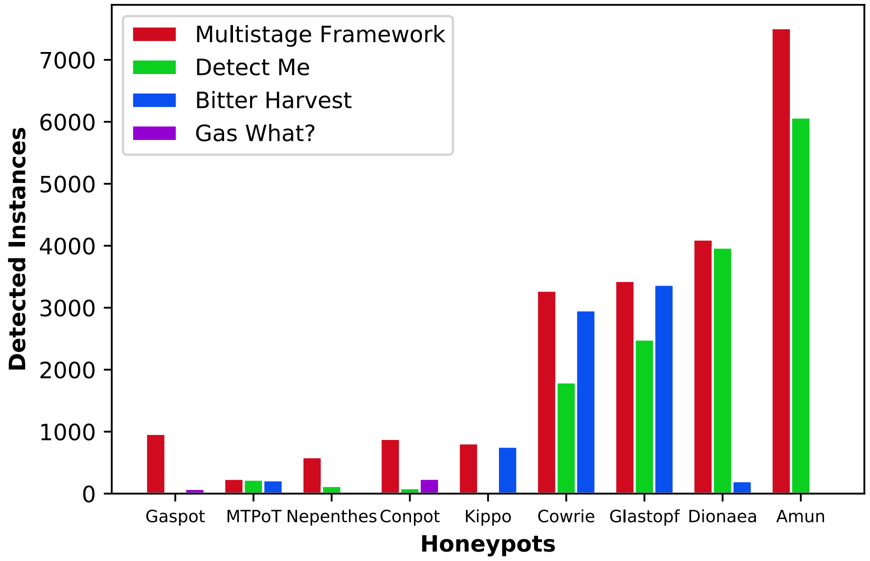

Fig. 4. Comparison to previous measurements in related work.

a relatively long period for honeypot administrators to react, whereas many honeypots (e.g., Conpot) have been updated to fix relevant vulnerabilities. Last, the multistage nature of our framework suggests that, in contrast to related work, we should encounter a very small number of false positives. In other words, IP addresses are only marked as honeypots when all (relevant) stages are confirmed.

3.3.2 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

Table 5. Detected Honeypot Versions

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	2,925
Glastopf 3.1.2*	3.1.2	4
	0.2.0	3,416
Dionaea 0.8.0*	0.8.0	2,259
	0.6.0	1,782

*Latest Version.

Table 6. Detected Honeypots Running on Default Configuration

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	3,149	25
Amun	7,455	57
Glastopf	3,416	4
Dionaea	4,064	37
Total	**21,305**	**351**

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 5.

3.3.3 *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 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 quite successful. We list the number of honeypot instances running with default configurations in Table 6.

Table 7. Vulnerable Instances of Identified Non-Honeypot Instances

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**

3.3.4 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 instances 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 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 (Table 7) with unpatched versions and default passwords among the non-honeypot systems identified. Based on this, we derive the following findings.

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 failed 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 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 [5]). 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 7.

3.3.5 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 following: (1) we will observe some

Table 8. Identification Gain vs. Blocked/Offline Instances

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

fluctuation in our results, (2) we will have some gain as new honeypots are introduced on the Internet, (3) we expect some of the networks to blocklist our scanners, and (4) we anticipate some honeypots not to be responsive due to them being taken down, maintenance, and/or network errors.

We scan the Internet with a different scanning host that has a 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 tradeoff 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 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 honeypot 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 that 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.

3.4 Shodan Honeyscore

The Shodan Honeyscore is a proprietary algorithm used to determine whether a crawled instance is a honeypot or not [36]. 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* honeypot IP's determined by our framework and compare the results with Shodan. Figure 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 the Honeyscore for

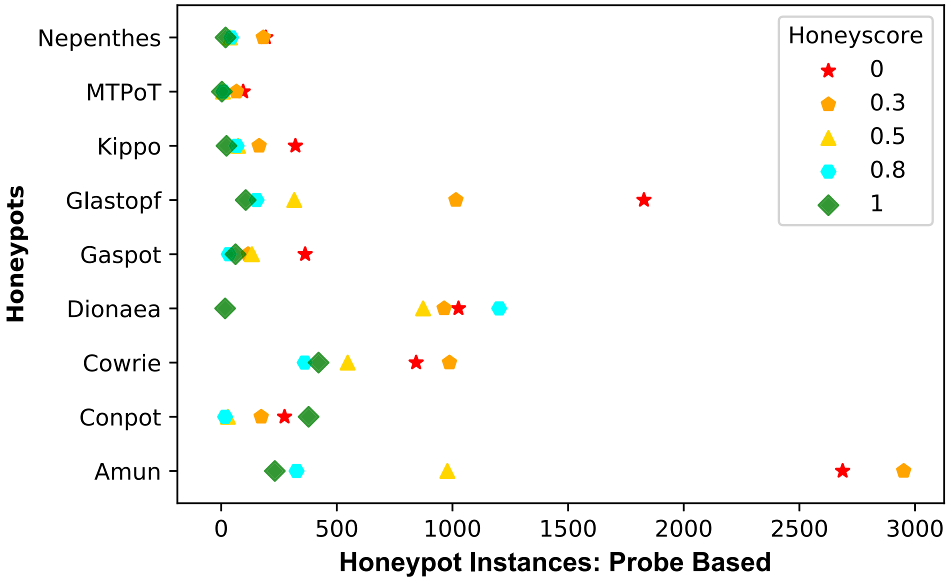

Fig. 5. Comparison with Shodan’s Honeyscore.

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.

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 that, we argue, significantly decrease the probability of false positives. Looking at the state of the art, Vetterl and Clayton [47] measure the detection accuracy using the responses received from the honeypots by generating a cosine similarity score, and Morishita et al. [28] 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 Morishita et al. [28] and Vogt et al. [49]. Morishita et al. [28] 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. [49] 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 [49]. In this context, we evaluate our results by comparing the identified honeypot IP addresses with the top 1 million domains from Alexa [2], Majestic [27], and Cisco Umbrella [7] 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 [16], Apache [15], Ubuntu [14], and Debian [9] 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 GitHub [20] for evaluating the 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, although the state of the art 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 Morishita et al. [28] 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.

4.1.1 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 set up 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.

4.1.2 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 3 months following the completion of our research.

4.1.3 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 [32], an international security research organization that focuses on honeypot research, to further disclose the fingerprinting mechanisms that we have identified.

4.1.4 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 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: although 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. Although 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 outside the scope of this article.

Las, although 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 are not competing with 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 the 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 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 [40] 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.

5.2.1 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 this 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.

5.2.2 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. However, most of the libraries used in honeypot implementations are 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 and Clayton [47] suggests 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.

5.2.3 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 address 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., [4, 46]). However, we consider this outside the scope of this article. Similarly, we will not discuss here fingerprinting of honeypot-like systems (e.g., honeytoken identification) [39]. We also note all works in the state of the art exclude high-interaction honeypots from their analysis.

Techniques for fingerprinting honeypots were first proposed early in 2005 by Holz and Raynal [22]. The authors state that limited simulation and virtualization cause restricted interaction on the honeypot system that leads to fingerprinting possibilities. Holz et al. [22] 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 OS level. This is mainly due to the limited availability of honeypots at the time of research.

Wang et al. [50] present an approach to detect honeypots in advanced botnet attacks. 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 that is transmitted by the infected system through the bots. For example, the use of the *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 and Clayton [47] 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. 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. [28] 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-Gourabi et al. [52] detect GasPot [51], an ATG-based ICS honeypot through probes designed to fetch information about the default configuration and limited emulation of the protocols. 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. [23] probe remote systems and label the response data to train a machine learning model to classify systems as honeypots. The method follows a recursive probing process to obtain featured data for classification. The features include application-, network-, 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 and Chilamkurti [31] attempt to exploit some of the virtual network layers implemented in honeypots, using tools like NMap, to fingerprint them. 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.

Last, Sun et al. [41] propose a fuzzing-based technique for fingerprinting honeypots in industrial cyber-physical systems. 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 set up 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 9 summarizes the fingerprinting-related work. We note that the majority of 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 [13]. That said, the fingerprinting techniques suggested by some authors [28, 47, 48, 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, for discovering new attack trends, and for early warning systems. In this article, we proposed a framework for honeypot fingerprinting that includes new and state-of-the-art components and is able to identify thousands of honeypot instances for nine of the

Table 9. Overview of the Related Work

Authors and Year	**Fingerprinting Technique**	**IPv4 Scan**
Holz and Raynal, 2005 [22]	Static command response check	No
Wang et al., 2010 [50]	Static command response check	No
Hayatle et al., 2012 [19]	Static command response check	No
Aguirre-Anaya et al., 2014 [1]	Library dependency check, static command response check	No
Vetterl and Clayton, 2018 [47]	Banner check, protocol handshake check, library dependency check, static command response check	Yes
Vetterl et al., 2019 [48]	Banner check, library dependency check	Yes
Huang et al., 2019 [23]	Banner check, static command response check	No
Morishita et al., 2019 [28]	Banner check, HTTP static response	Yes
Zamiri-Gourabi et al., 2019 [52]	Default config, static response, protocol handshake	Yes
Papazis and Chilamkurti, 2019 [31]	Banner check, HTTP static response, static command check	No
Sun et al., 2021 [41]	Fuzzing, limited response	No

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 developers and administrators of the honeypots to make them aware of potential fingerprinting issues. However, based on the experience of previous work, we are not overly 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 Defense schemes. We plan to further investigate fingerprinting countermeasures in our future work.

APPENDICES

A MULTISTAGE FRAMEWORK FOR HONEYPOT FINGERPRINTING

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

Table 10. Banners Advertised by Honeypots (adapted from Vetterl and Clayton [47] and Morishita et al. [28]; see *Banner Check* in Section 2.2.1)

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

Table 11. HTTP Response from Honeypots (see *HTTP Static Response* in Section 2.2.1)

Honeypot	**HTTP Request**	**HTTP Response Contents**
Glastopf	GET/HTTP/1.0	1. My Resource 2. Blog Comments \n <label for=\"comment\">Please post your comments for the blog</label>\n \n <textarea name=\"comment\" id=\"comment\" rows=\"4\" columns=\"300\"></textarea>\n \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> It works! tim\.bohn@gmx\.net 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 Directory listing for / \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 12 shows the non-honeypot instances determined at stages in our framework based on honeypot types. Limited emulation in honeypots causes identification at different levels that are determined by the stages in our framework.

Table 12. Non-Honeypot Encounters by Stage (also see Section 3.3.4)

Honeypot	**Portscan**	**Failed Banner**	**Failed Static HTTP Response**	**Failed SSL/TLS Certificate Check**	**Failed Protocol Handshake**	**Failed Library Dependency Check**	**Not a Honeypot**
Kippo	4,361,857	4,324,502	NA	NA	34,887	1,656	4,361,045
Cowrie	4,361,857	4,318,645	NA	NA	37,836	2,100	4,358,581
Glastopf	57,062,712	56,385,819	673,462	NA	0	0	57,059,281
Dionaea	43,944,853	43,890,588	49,963	201	0	0	43,940,752
Nepenthes	10,391,953	10,391,645	NA	NA	3	0	10,391,648
Conpot	29,950	28,693	NA	NA	732	333	29,758
Gaspot	222,593	222,393	NA	NA	0	0	222,393
MTPot	2,923,651	2,923,412	NA	NA	0	0	2,923,412
Amun(SMTP)	6,020,828	6,018,931	NA	NA	0	0	6,018,931
Amun(IMAP)	4,152,084	4,150,278	NA	NA	0	0	4,150,278
Amun(FTP)	10,391,953	10,389,555	NA	NA	0	0	10,389,555
Amun(HTTP)	43,944,853	43,942,485	NA	NA	0	0	43,943,476
Total	**187,809,144**	**186,986,946**	**724416**	**201**	**73,458**	**4,089**	**187,789,110**

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

Table 13. Honeypot Keywords Search (also see *Keyword Search* in Section 2.2.2)

Honeypot	**Shodan**	**Censys**
Glastopf	My Resource	80.http.get.body: “ My Resource /”
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 14 shows the static response returned by honeypots for specific commands requested by our probes. Limited emulation or default configuration leads to static response from the honeypots.

Table 14. Overview of Honeypot Static Responses

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

In reference to Section 2.2.1.

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

Table 15. Honeypots Detected per Scan

Honeypot	**Scan 1**	**Scan 2**	**Scan 3**	**Total**	**Total (Active)**
Dionaea	4,101	4,101	4,101	**4,101**	4,101
Glastopf	3,431	3,433	3,420	**3,433**	3,420
Cowrie	3,276	3,272	3,174	**3,276**	3,174
Amun(FTP)	2,398	2,388	2,379	**2,398**	2,379
Amun(SMTP)	1,897	1,897	1,883	**1,897**	1,883
Amun(IMAP)	1,806	1,806	1,795	**1,806**	1,795
Amun(HTTP)	1,377	1,375	1,455	**1,455**	1,455
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

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 through 5, the algorithm retrieves the certificate from the web server by accepting the IP and port of the instance and checks for common attributes like subject organization, country, and issuer. These attributes have static values assigned by the honeypot developers. In lines 7 through 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 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 shows the checks done in the metascan-based pipeline. The algorithm checks or open ports, and performs a 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 2: Metascan-Based Pipeline

```
input  :ports                                              /* ports */
output :findHoneypot                   /* Honeypots on the Internet */
begin
    ip ← metasearch(ports)
                                           /* Shodan and Censys Search */
    foreach ip do
        kw = keywordSearch(ip)
        if kw then
            foreach ip do
            if checkfqdn(ip) then
                return hasFqdn = false
            if !hasfqdn & port=502|102 then
                if cloudCheck(ip) then
                    return isHoneypot = true
                isHoneypot
            if !hasfqdn then
                if isResearch(ip) then
                    return isResearch = true
                isHoneypot
            endIf
        endFor
    end
```

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.

ALGORITHM 3: Metascan Search

```
input  :port                                    /* search parameter */
output :instanceIP                     /* Instances with open ports */
begin
    instances[] = null
    shodanSearch(port)
    foreach ip do
        instances[].append(ip, port)
        return instances[]
        endFor
    censysSearch(port)
    foreach ip do
        instances[].append(ip, port)
        return instances[]
        endFor
    end
```

Last, Algorithm 4 represents the protocol handshake check procedure described in Section 2.2.1. 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 4: Protocol Handshake Check

```
input  :instance[],                                   /* Instance [ip, port, protocol, isDeviated] */
output :instance[isDeviated]                                               /* Handshake is deviated */
begin
    CheckHandshake(instance[])
    foreach ip in Instance[] do                                                 /* For each instance */
        if port=22/2222 & protocol="SSH" then
            request("SSH-2.0-OpenSSH\n\n\n\n\n\n\n\n\n\n")
            if response = "bad Packet length" or "protocol mismatch"
            return Instance[isDeviated] = true
            else request("SSH-2.0-OpenSSH_6.0p1 Debian-4+deb7u2\n")
            if response = "protocol mismatch\n"
            return Instance[isDeviated] = true
        elseif port=102 & protocol="S7" then
            request(H,0300002102f08032070000...)
            if response = "0x32"
            return Instance[isDeviated] = true
        elseif port=502 & protocol="Modbus" then
            request(function_code:None, slave_id:0, request:0000000000050..)
            if session.state = "Disconnected"
            return Instance[isDeviated] = true
        elseif port=25 & protocol="SMTP" then
            request(PASS:Test)
            if response = "220 OK"
            return Instance[isDeviated] = true
        elseif port=143 & protocol="IMAP" then
            request(RCPT TO:TEST)
            if response = "221 Bye Bye"
            return Instance[isDeviated] = true
        elseif port=21 & protocol="FTP" then
            request(ftp (ip))
            if packet.windowSize=4096 & session.disconnect.timeout=45
            return Instance[isDeviated] = true
        elseif port=23/2323 & protocol="Telnet" then
            request(telnet (ip))
            if packet.response= "You have connected to the telnet server"
            return Instance[isDeviated] = true
        elseif port=80/8080/443/8443 & protocol="HTTP/HTTPS" then
            request(GET /HTTP/1.0 (ip))
            if response.packet.header.server= "nginx" or "Apache/1.3.29" or "BaseHTTP/0.3 Python/2.5.1" or
            "Microsoft-IIS/5.0"
            return Instance[isDeviated] = true
```

REFERENCES

[1] E. Aguirre-Anaya, G. Gallegos-García, Nicolás Solano Luna, and Luis A. Villa Vargas. 2014. A new procedure to detect low interaction honeypots. *International Journal of Electrical and Computer Engineering* 4 (2014), 848–857.

[2] Amazon. 2020. *Alexa—An Amazon Company*. Amazon. https://www.alexa.com/topsites.

[3] Ofir Arkin, Fyodor Yarochkin, and Meder Kydyraliev. 2003. *The Present and Future of Xprobe2—The Next Generation of Active Operating System Fingerprinting*. SYS-Security Groups.

[4] John Bethencourt, Jason Franklin, and Mary Vernon. 2005. Mapping Internet sensors with probe response attacks. In *Proceedings of the 14th USENIX Security Symposium (USENIX Security'05)*. USENIX Association, Baltimore, MD. https://www.usenix.org/conference/14th-usenix-security-symposium/mapping-internet-sensors-probe-response-attacks.

[5] Harold Booth, Doug Rike, and Gregory Witte. 2013. *The National Vulnerability Database (NVD): Overview*. Technical Report. National Institute of Standards and Technology.

[6] Bogdan Botezatu. 2018. New Hide'n Seek IoT Botnet Using Custom-Built Peer-to-Peer Communication Spotted in the Wild. Retrieved March 1, 2023 from https://www.bitdefender.com/blog/labs/new-hide-n-seek-iot-botnet-using-custom-built-peer-to-peer-communication-spotted-in-the-wild/.

[7] Cisco. 2020. Umbrella Popularity List. Cisco. Retrieved March 1, 2023 from https://umbrella-static.s3-us-west-1.amazonaws.com/index.html.

[8] Cymmetria. 2016. MTPot. Retrieved March 1, 2023 from https://github.com/Cymmetria/MTPot.

[9] Debian. 2020. Debian Project. Debian Project. Retrieved March 1, 2023 from https://www.debian.org/mirror/list.

[10] Decester. 2000. An SSH Honeypot. Retrieved March 1, 2023 from https://github.com/desaster/kippo.

[11] DomainTools. 2022. *DomainTools Whois Lookup*. Retrieved March 1, 2023 from https://whois.domaintools.com/.

[12] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. 2015. A search engine backed by Internet-wide scanning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS'15)*. ACM, New York, NY, 542–553. https://doi.org/10.1145/2810103.2813703

[13] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. 2013. ZMap: Fast Internet-wide scanning and its security applications. In *Proceedings of the 22nd USENIX Conference on Security (SEC'13)*. USENIX Association, 605–620.

[14] Ubuntu. 2020. Official Archive Mirrors for Ubuntu. Retrieved March 1, 2023 from https://launchpad.net/ubuntu/+archivemirrors.

[15] The Apache Foundation. 2020. *The Apache Software Foundation*. Apache.org. https://apache.org/history/mirror-history.html.

[16] GNU.org. 2020. *GNU Operating System*. GNU.org. Retrieved March 1, 2023 from https://www.gnu.org/prep/ftp.en.html.

[17] Robert David Graham. 2014. MASSCAN: Mass IP Port Scanner. Retrieved March 1, 2023 from https://github.com/robertdavidgraham/masscan.

[18] T. Grudziecki, P. Jacewicz, Ł. Juszczyk, P. Kijewski, and P. Pawliński. 2012. *Proactive Detection of Security Incidents, Report*. ENISA. https://www.enisa.europa.eu/publications/proactive-detection-report.

[19] Osama Hayatle, Amr Youssef, and Hadi Otrok. 2012. Dempster-Shafer evidence combining for (anti)-honeypot technologies. *Information Security Journal: A Global Perspective* 21, 6 (2012), 306–316.

[20] Hipo. 2020. *GITHUB*. Hipo. Retrieved March 1, 2023 from https://github.com/Hipo/university-domains-list.

[21] Filip Holik, Josef Horalek, Ondrej Marik, Sona Neradova, and Stanislav Zitta. 2014. Effective penetration testing with metasploit framework and methodologies. In *Proceedings of the 2014 IEEE 15th International Symposium on Computational Intelligence and Informatics (CINTI'14)*. IEEE, Budapest, Hungary, 237–242.

[22] T. Holz and F. Raynal. 2005. Detecting honeypots and other suspicious environments. In *Proceedings from the 6th Annual IEEE SMC Information Assurance Workshop*. IEEE, West Point, NY, 29–36. https://doi.org/10.1109/IAW.2005.1495930

[23] Cheng Huang, Jiaxuan Han, Xing Zhang, and Jiayong Liu. 2019. Automatic identification of honeypot server using machine learning techniques. *Security and Communication Networks* 2019, 5 (2019), 1–8.

[24] Liz Izhikevich, Renata Teixeira, and Zakir Durumeric. 2022. Predicting IPv4 services across all ports. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM'22)*. ACM, New York, NY, 503–515. https://doi.org/10.1145/3544216.3544249

[25] Lukas Krämer, Johannes Krupp, Daisuke Makita, Tomomi Nishizoe, Takashi Koide, Katsunari Yoshioka, and Christian Rossow. 2015. AmpPot: Monitoring and defending against amplification DDoS attacks. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*. 615–636.

[26] Gordon Lyon. 2021. NMap Network Mapper. Retrieved March 1, 2023 from https://nmap.org/.

[27] Majestic. 2021. *The Majestic Million*. Majestic. Retrieved March 1, 2023 from https://majestic.com/reports/majestic-million.

[28] Shun Morishita, Takuya Hoizumi, Wataru Ueno, Rui Tanabe, Carlos Gañán, Michel J. G. van Eeten, Katsunari Yoshioka, and Tsutomu Matsumoto. 2019. Detect me if you... oh wait. An Internet-wide view of self-revealing honeypots. In *Proceedings of the 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM'19)*. IEEE, Arlington, VA, 134–143.

[29] Marcin Nawrocki, Matthias Wählisch, Thomas C. Schmidt, Christian Keil, and Jochen Schönfelder. 2016. A survey on honeypot software and data analysis. *arXiv:1608.06249* [cs.CR] (2016).

[30] Michel Oosterhof. 2016. Cowrie SSH/Telnet Honeypot. Retrieved March 1, 2023 from https://github.com/micheloosterhof/cowrie.

[31] Kon Papazis and Naveen Chilamkurti. 2019. Detecting indicators of deception in emulated monitoring systems. *Service Oriented Computing and Applications* 13, 1 (2019), 17–29.

[32] The Honeynet Project. 2021. The Honeynet Project. Retrieved March 1, 2023 from https://www.honeynet.org.

[33] Rapid7. 2021. Recog. https://github.com/rapid7/recog.

[34] L Rist. 2009. *Glastopf project*. The Honeynet Project. Retrieved March 1, 2023 from https://www.honeynet.org/projects/old/glastopf/.

[35] Lukas Rist, Johnny Vestergaard, Daniel Haslinger, A. Pasquale, and J. Smith. 2013. CONPOT ICS/SCADA Honeypot. *The Honeynet Project*. Retrieved March 1, 2023 from http://conpot.org/.

[36] SHODAN. 2021. Honeypot or Not? Retrieved March 1, 2023 from https://honeyscore.shodan.io.

[37] Lance Spitzner. 2000. *Passive Fingerprinting*. Tech Solvency.

[38] L. Spitzner. 2003. The Honeynet Project: Trapping the hackers. *IEEE Security Privacy* 1, 2 (2003), 15–23.

[39] Shreyas Srinivasa, Jens Myrup Pedersen, and Emmanouil Vasilomanolakis. 2020. Towards systematic honeytoken fingerprinting. In *Proceedings of the 13th International Conference on Security of Information and Networks (SIN'20)*. ACM, New York, NY, Article 28, 5 pages. https://doi.org/10.1145/3433174.3433599

[40] Shreyas Srinivasa, Jens Myrup Pedersen, and Emmanouil Vasilomanolakis. 2021. RIoTPot: A modular hybrid-interaction IoT/OT honeypot. In *Proceedings of the 26th European Symposium on Research in Computer Security (ESORICS'21)*. Springer, Darmstadt.

[41] Yanbin Sun, Zhihong Tian, Mohan Li, Shen Su, Xiaojiang Du, and Mohsen Guizani. 2021. Honeypot identification in softwarized industrial cyber-physical systems. *IEEE Transactions on Industrial Informatics* 17, 8 (2021), 5542–5551. https://doi.org/10.1109/TII.2020.3044576

[42] Dino Tools. 2010. Web Honeypot. Retrieved March 1, 2023 from https://github.com/DinoTools/dionaea/.

[43] Hauser Van and Kessler Roland. 2021. *Kali Tools*. THC.org. Retrieved March 1, 2023 from https://www.thc.org/thc-hydra/.

[44] Emmanouil Vasilomanolakis, Shankar Karuppayah, Max Mühlhäuser, and Mathias Fischer. 2014. HosTaGe: A mobile honeypot for collaborative defense. In *Proceedings of the 7th International Conference on Security of Information and Networks (SIN'14)*. ACM, New York, NY, 330–333. https://doi.org/10.1145/2659651.2659663

[45] Emmanouil Vasilomanolakis, Shreyas Srinivasa, Carlos Garcia Cordero, and Max Mühlhäuser. 2016. Multi-stage attack detection and signature generation with ICS honeypots. In *Proceedings of the 2016 IEEE/IFIP Network Operations and Management Symposium*. IEEE, Los Alamitos, CA, 1227–1232. https://doi.org/10.1109/NOMS.2016.7502992

[46] E. Vasilomanolakis, M. Stahn, C. G. Cordero, and M. Mühlhäuser. 2016. On probe-response attacks in collaborative intrusion detection systems. In *Proceedings of the 2016 IEEE Conference on Communications and Network Security (CNS'16)*. IEEE, Florence, 279–286.

[47] Alexander Vetterl and Richard Clayton. 2018. Bitter harvest: Systematically fingerprinting low- and medium-interaction honeypots at Internet scale. In *Proceedings of the 12th USENIX Workshop on Offensive Technologies (WOOT'18)*. USENIX Association, Baltimore, MD. https://www.usenix.org/conference/woot18/presentation/vetterl.

[48] Alexander Vetterl, Richard Clayton, and Ian Walden. 2019. Counting outdated honeypots: Legal and useful. In *Proceedings of the 2019 IEEE Security and Privacy Workshops (SPW'19)*. IEEE, San Francisco, CA, 224–229. https://doi.org/10.1109/SPW.2019.00049

[49] Ryan Vogt, John Aycock, and Michael J. Jacobson Jr. 2007. Army of botnets. In *Proceedings of the Network and Distributed System Security Symposium (NDSS'07)*. NDSS, San Diego, CA.

[50] Ping Wang, Lei Wu, Ryan Cunningham, and Cliff C. Zou. 2010. Honeypot detection in advanced botnet attacks. *International Journal of Information and Computer Security* 4, 1 (2010), 30–51.

[51] Kyle Wilhoit and Stephen Hilt. 2015. *The GasPot Experiment: Unexamined Perils in Using Gas-Tank-Monitoring Systems*. Black Hat.

[52] Mohammad-Reza Zamiri-Gourabi, Ali Razmjoo Qalaei, and Babak Amin Azad. 2019. Gas what? I can see your gaspots. Studying the fingerprintability of ICS honeypots in the wild. In *Proceedings of the 5th Annual Industrial Control System Security Workshop (ICSS'19)*. ACM, New York, NY, 30–37. https://doi.org/10.1145/3372318.3372322

[53] GitHub. 2020. ZMap: Sample Applications. Retrieved March 1, 2023 from https://github.com/zmap/zmap/wiki/Sample-Applications.

[54] C. C. Zou and R. Cunningham. 2006. Honeypot-aware advanced botnet construction and maintenance. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'06)*. IEEE, Los Alamitos, CA, 199–208. https://doi.org/10.1109/DSN.2006.38

Received 6 July 2022; revised 21 October 2022; accepted 15 December 2022