University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

VisiBot: automated detection of IoT botnets using heuristic analysis techniques

Daniel Arthur (2086380A)

April 7, 2021

Abstract

The substantial demand for the Internet of Things (IoT) has led to the mass production of IoT devices vulnerable to common security vulnerabilities, such as login credential brute-forcing and Remote Code Execution (RCE). Bad actors have begun to exploit such vulnerabilities in IoT devices to build powerful IoT botnets to coordinate large-scale distributed cyber-attacks. The detection and mitigation of such botnets is a high priority for authorities and cyber-threat intelligence specialists, as such cyber-attacks have proven to impact countless services, organisations, and government entities significantly. However, the identification of modern botnets, such as those targeting IoT devices, has proven challenging for authorities due to continually changing communication protocols, target devices, attack vectors, and obfuscation techniques. Similarly, many systems proposed for botnet detection often target specific botnet variants and cannot be applied in globally distributed infrastructures necessary for identifying large-scale botnets.

In this paper, I propose VisiBot, a real-time, flexible, and fully automated botnet detection system capable of identifying centralised and de-centralised IoT botnets. Through the purposeful execution of malware samples from a globally distributed honeypot network, four identification heuristics are applied to a combination of static and dynamic analysis information to infer candidate Command & Control servers and Peer-to-Peer networks. Over a 35 day data-collection period, this system processed over 82,050 botnet events and successfully collected, executed, and analysed a total of 1,654 botnet malware samples. By combining novel detection heuristics with static, dynamic, and network analysis information extracted using automated IoT malware sandbox techniques, the proposed system successfully identified 1,303 candidate Command & Control (C2) servers and 6,876 Peer-to-Peer Nodes. As all analysis is fully automated, this framework allows for real-time identification and visualisation of several botnet characteristics, including the geographic botnet density, Peer-to-Peer interactivity, and Autonomous System interactions with candidate C2 servers.

Education Use Consent

I hereby grant my permission for this project to be stored, distributed and shown to other University of Glasgow students and staff for educational purposes. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Signature: Daniel Arthur Date: April 7, 2021

Acknowledgements

I want to extend my deepest gratitude to my final year project supervisor, Dr. Angelos K. Marnerides, for the continual guidance, advice, and support that I have received throughout my project. Throughout Angelos' consultation, I have learned significantly from his expertise and research contributions. The time spent working with him has been both a pleasure and an honour.

The development of VisiBot would not have been possible without the contributions of Bad Packets. I want to offer my deepest gratitude towards both the secretary and director of Bad Packets, Troy Mursch and Mathew Woodyard, for providing full access to the Bad Packets Cyber Threat Intelligence service and hosting platforms, offering excellent advice and support throughout my project. I would also like to thank Polina Stolpovskaya and associates at IPInfo for providing access to their various IP address API services and the support offered throughout my academic research. Additionally, thanks should also go to VirusTotal for providing special access to premium API features in support for my research, and MaxMind for providing access to their GeoIP2 databases service free of charge. I want to offer many thanks to Daniel Uhříček, developer of the open-source LiSa sandbox for his hard work and extensive support.

Finally, I am incredibly grateful for the support offered by my family, friends, and fellow students at the University of Glasgow throughout my final year of studies. I want to close my acknowledgements by thanking the staff at the University of Glasgow School of Computing Science, and more so the head of honours, Dr. Gethin Norman, for providing unparalleled support throughout the COVID-19 pandemic.

Contents

1 Introduction

On the 21st of October, 2016, a leading Domain Name Service, known as DynDNS, was targeted by Hacktivists in one of the most significant Distributed Denial of Service (DDoS) attacks recorded to date. Lasting several hours, the outages following the attack directly impacted many services, including Cloudflare, Amazon, Twitter, Netflix, and GitHub. [1] Unlike those seen before, the DynDNS DDoS attack was performed by thousands of Internet of Things (IoT) devices, coordinated by a botnet known as Mirai. A botnet is a network of private computers that have been infected with malicious software. All computer systems within the network, i.e. bots, are controlled as a group to perform malicious tasks, such as spam distribution or denial of service attacks. [2] It is estimated that the Mirai botnet infected over 500,000 IoT devices before the attack. Its fast propagation allowed over 100,000 infected IoT devices to participate in the attack against DynDNS, yielding a maximum attack magnitude of 1.2 Tbps. [3] Contrary to conventional botnets, which often operate using infected computer desktops, the Mirai botnet exclusively targets vulnerable IoT devices, including unprotected IP/CCTV cameras, baby monitors, and routers.

The Internet of Things (IoT) describes the intercommunication of embedded systems, aptly described as 'things', which actively communicate and exchange data with other devices. [4] Arm, a leading provider in processors for embedded systems, characterises IoT devices as "pieces of hardware, such as sensors, actuators, gadgets, appliances, or machines, that are programmed for certain applications and can transmit data over the internet or other networks." [5] Industrial IoT devices are often embedded within other mobile devices, environmental sensors, and industrial or medical equipment. However, commercial IoT devices are also commonly used in home and working environments, including Digital Video Recorders (DVR), IP-Cameras, Internet Routers, Smart Home Security Systems, and Voice Assistants. Such devices have become increasingly popular because of increased connectivity, low cost and power consumption, and integration with cloud computing platforms, artificial intelligence, and machine learning. [4] Lueth [6] approximates that 54% of the 21.7 billion devices connected to the internet in 2020 were that of IoT devices. Due to the rampant manufacturing of inexpensive IoT devices, it is estimated that, by the end of the year 2025, over 30 billion active IoT devices will be connected, amounting to almost four IoT devices per person.

1.1 Motivation

Following a recent surge in demand for IoT devices, several manufacturers have expanded to compete in a new and highly competitive market. Due to a lack of consideration for security and update-ability, the rapid production of IoT devices has resulted in the release of millions of vulnerable IoT devices. The Open Web Application Security Project (OWASP) reported that the top 10 most common IoT Vulnerabilities include weak, guessable, or hard-coded passwords. Additionally, IoT devices often have insecure network services or ecosystem interfaces, a lack of secure updating systems and device management, and operate using insecure data transfer protocols. [7] Known as Common Vulnerability Exploits (CVE), these vulnerabilities are frequently exploited by bad actors, such as botnet owners, to cultivate and propagate IoT-based botnets.

As many targeted IoT devices are often un-managed on an administrative level, intrusions often go unnoticed by the owner, allowing the device to participate in distributed cyber-attacks surreptitiously.

In light of the Mirai botnet source-code release in 2016, many anticipated that the internet would soon be flooded with Mirai-like botnets by highly vulnerable and easily hackable IoT devices. [8] As IoT device popularity increases, so too will the number of vulnerabilities. The evolution of botnets and web application frameworks has also resulted in an influx of online DDoSing services, allowing individuals to hire botnets for orchestrating attacks temporarily. Depending on the botnet's size, renting for DDoS attacks can cost up to several thousand dollars per day, enabling significant monetary gain for botnet owners. [9] The cost of identifying, mitigating, and recovering from botnet DDoS attacks is believed to be in the millions. DDoS attacks have a detrimental impact on both targeted services and their dependents and have been known to cause large-scale disruptions capable of affecting millions of users. Government agencies, commercial services, and Cyber Threat Intelligence agencies have great interest in identifying and alleviating such botnets and seek to employ large-scale botnet detection systems to allow for preemptive botnet detection. Many botnets operate using centralised Command & Control (C2) servers used to distribute commands amongst bots, thus, authorities have found success in mitigating botnets through identifying and shutting down such servers. By taking out the C2 server of a botnet, the central point of communication is interrupted and infected bots are rendered useless. However, botnet developers are now employing de-centralised communication protocols. By removing central points of contact through a employing a peer-based communication strategy, perpetrators can significantly reduce the likelihood of interference from authorities.

1.2 Aims and Objectives

Current botnet detection systems are often highly theoretical, target very specific communication protocols, and are often difficult to apply within real-time, automated, and globally distributed context. To alleviate these challenges, I propose VisiBot: a modular, automated, and distributable botnet detection and visualisation tool. In collaboration with Bad Packets, [10] the proposed system aims to identify and analyse internationally distributed IoT botnets by profiling and analysing potentially malicious network packets collected from a globally distributed IoT honeypot network. Using automated and distributable technologies, the VisiBot Processing System achieves real-time, extensible botnet detection through:

- Automatic honeypot packet collection, classification, and malware sample extraction
- Automated static and dynamic analysis of malware samples using the LiSa sandbox [11]
- Heuristic-based detection of candidate IoT botnet C2 Servers and P2P botnets
- Supplementary monitoring of geographic density and Autonomous System (AS) interactions of identified botnets, and web-based geographic clustering visualisation

1.3 Summary

Following a 35-day data-collection and evaluation period, the VisiBot Processing System classified packets originating from a total of 61,293 unique public IP addresses. A total of 9,923 payload URLs were identified, followed by the successful extraction and analysis 1,654 of malware samples. Additionally, a total of 1,303 candidate C2 servers and 6,876 Peer-to-Peer botnet nodes were identified using four identification heuristics. Throughout this paper, several topics will be discussed, including an overview of relevant background research, an analysis of the problem at hand, an explanation of the design overview and implementation, and an evaluation of results collected and the system's effectiveness whole.

2 Problem Analysis and Design Overview

2.1 Problem Analysis

Preemptive botnet detection has often challenged authorities, as real-time, large-scale traffic analysis is often expensive and very difficult to coordinate. Current detection systems are often reactive, as they often rely on post-attack data sources and network logs from Internet Service Providers and DNS services. This approach is not ideal, as attacks must first take place to identify offending networks. The preemptive botnet detection technique proposed by Moon et al. [12] proved significant in detecting botnets before attacks took place. The purposeful execution and analysis of malware enabled botnets to be identified and mitigated during their propagation periods. This system allows for the automatic collection and analysis of binaries using two dedicated processing modules.

However, several issues are also presented within the detection framework proposed by Moon et al.. A primary concern is that the data collection methods employed are only achievable through integration with an Internet Service Provider (ISP) infrastructure. Thus, it isn't easy to reproduce the same data collection process without internal access to an ISP service. Additionally, the amount and type of information sourced from the KORNET ISP service varied significantly in scope and relevance to botnet propagation. The vast majority of binaries analysed were sourced from spam emails and malicious websites reported by KORNET users, with a significantly smaller number of binaries collected through botnet honeypots. The analysis of binaries collected through a combination of extraneous data sources led to a trivial detection accuracy of 40%, stipulating that a more refined data-collection process is required.

As botnets have evolved to target vulnerable IoT devices, conventional honeypot detection systems, such as Honeyd [13], have proven ineffective in identifying and monetising from IoT botnet traffic. The methods proposed by [14] and [15] allow for the detection of such IoT traffic through configuring honeypot servers to mimic characteristics of commonly targeted IoT devices. Despite proving successful in collecting IoT malware, both implementations resulted in sparse data collection due to small-scale deployment and honeypot distribution.

The method proposed by [14] combines IoT honeypots and sandboxing environments using embedded Linux virtual machines, allowing for the purposeful infection of sandboxes and monitoring all incoming telnet based botnet communication. However, each honeypot system has to be frequently (and manually) reset as successful intrusions often result in unpredictable or unwarranted behaviour. Additionally, the honeypot machines would only monitor and redirect telnet-based intrusion attempts. This system overlooked important information by exclusively monitoring telnet traffic by analysing other commonly used botnet communication protocols, including SSH, HTTP, P2P, and IRC traffic. Since modern botnets frequently operate across various communication protocols, the exclusion of such protocols may result in the loss of valuable network traffic information for botnet identification.

Bastos et al. [16] and Ceron et al. [17] further build upon the concept of IoT honeypot system integration through extracting, executing, and examining a set of malware samples collected by distributed sandbox environments. The long-term distribution of honeypots across 15 Brazilian states allowed for mass collection of IoT malware binaries. However, the lack of global honeypot

distribution may result in collected data sets being skewed by regionally dominant botnets. Botnet activity is identified through a combination of static, dynamic, and heuristic analysis techniques using information obtained throughout the sandboxing process. In both implementations, malware samples are statically and dynamically analysed through being executed in a secure sandbox environment called Detux. [18] Throughout execution, all incoming and outgoing network traffic is logged for analysis. However, despite the sandbox having support for up to 5 CPU architectures, it is constrained in terms of functionality, only allows for basic static and dynamic analysis during runtime, and is not scalable out of the box. These factors limit the possibility of real-time static and dynamic malware analysis of honeypot traffic.

Dwyer et al. [19] employs a solution that allows for accurate botnet detection without the requirement of malware execution and analysis of malware samples. The proposed DNS feature classification technique gave high botnet detection accuracy for Mirai botnets; however, this solution cannot be applied to de-centralised botnet structures. As de-centralised botnets use P2P protocols to communicate instead of standard protocols such as IRC, HTTP and DNS, Peer-to-Peer botnets collectively communicate messages between connected peers rather than directly through a C2 server, mitigating the need for DNS-based communication.

Herwig et al. [20] illustrates how modern IoT botnets are being coordinated through de-centralised Peer-to-Peer networks via Distributed Hash Tables (DHT). Despite discussing remote code execution tactics commonly observed from propagating Hajime botnets, little insight is given regarding the payload extraction process. As malware URLs within remote code execution attempts are likely to be obfuscated, it is unclear if this system accounts for obfuscation when extracting malware samples for analysis. Herwig et al. also propose that Hajime bots can be identified using public keys as identifiers, as IP frequently addresses change ownership. However, both the transition of ownership and geographical distribution of IP addresses may prove significant for in-depth botnet analysis. Such IP address meta-data can help identify substantial patterns across geographic botnet density and malicious IP address ownership trading between Autonomous Systems.

As IoT devices become more popular amongst households and businesses and new vulnerabilities are exposed, the importance of real-time identification and profiling of botnet traffic has significantly increased. Various Internet Service Providers, services, and government entities seek to mitigate the problem of botnet-based cyber-attacks; nevertheless, the continual evolution of botnets has made cyber-attack mitigation exceedingly more difficult. Such parties require quick and efficient ways to identify, visualise, and monitor current botnet activity, such that botnets can be observed and countered during propagation. The above botnet identification methods show high reliability and accuracy for IoT botnet detection. Yet, many of the proposed solutions are computationally expensive, difficult to deploy, or do not account for botnets' evolutionary factors, as attackers are persistently seeking to exploit new vulnerabilities, attack new target devices, and minimise detection whilst doing so.

2.2 VisiBot Processing System

VisiBot is an alternative botnet detection framework that focuses on real-time, automatic malware extraction and analysis. Using a globally distributed honeypot service, the VisiBot Processing System employs automatic extraction and execution of IoT botnet malware through sandbox and heuristic analysis techniques using scalable and distributable technologies.

2.2.1 Honeypot Integration

The VisiBoT Processing System employs a vast network of honeypots strategically distributed across a diverse set of network providers located in North America, South America, Asia, Europe,

and Australia. The honeypot servers have been configured to emulate commonly targeted IoT devices to maximise incoming botnet activity and intrusion attempts. Once a honeypot detects an intrusion attempt, the corresponding packet information is sent to a central processing server for analysis and storage. Once received, the primary honeypot processing server analyses and classifies packets based on characteristics observed across various Common Vulnerability Exploits (CVE), post-data characteristics, and TCP sequences. By analysing the payload string, post data, and TCP sequences of incoming packets, the honeypot service can infer which CVEs are being exploited by matching string patterns. Once processed, the captured honeypot packets are made accessible through a REST API (Application Protocol Interface) accessible by the VisiBot Processing System.

2.2.2 Broker-based Packet Analysis

The packets collected from the external honeypot service are processed and analysed in parallel, enabling quick and efficient malware URL extraction and packet classification. Simultaneous packet analysis is achieved through the adoption of the Message Broker architectural design pattern. Also known as Integration Broker, a Message Broker is "a third-party intermediary that facilitates interactions between applications" [21], through the communication and transformation of messages between various independent programs by interacting with message broker through an API. A broker-based analysis system allows for the execution of time-consuming tasks to be distributed across multiple worker processes. As workers are entirely separate from the main application and interact with a broker through an API, they can be reliably scaled and distributed across multiple systems. Clients, such as the VisiBot Processing System's primary process, can connect to a broker and add, remove, or prioritise tasks, such as packet analysis tasks. Connected workers actively consume and perform tasks from a central priority queue.

Once a packet is received, the main VisiBot process creates a new packet analysis task, passing the task name and packet contents to the message broker to be later processed by a worker. The first stage of analysis involves malware payload URL extraction from the payload and post_data contents of the packet. The URLs used in remote code execution requests or downloading and infecting devices can be extracted using a combination of Regular Expression and sub-string extraction techniques. If no strings match common URL regex patterns, such as `https?:`, then the attacker may be obfuscating URLs through command-line tool argument sub-strings. This prevents the automatic scraping of malware URL and requires URLs to be rebuilt from parsed argument strings. Through parsing the arguments of command-line tools commonly used for downloading binaries, such as `wget, curl` and `tftp`, the processing system can extract important sub-strings. Such strings include arguments such as IP addresses, domains, paths, and ports, which are used to reconstruct a full URL.

After all potential malware URLs are extracted, the worker will perform packet classification by contextualising extracted URL information. If a URL is extracted and its corresponding IP address matches that of the packet's source IP address, the entity caught within the honeypot is deemed malicious and is labelled as a 'Malicious Bot'. Malicious Bots are frequently observed in Peer-to-Peer botnets, as they actively self-host malware and propagate their corresponding botnet by sending Remote Code Execution attempts to vulnerable devices. If the URL's IP address does not match the packet's source IP, the processing system infers that the source packet IP corresponds to a Report Server. Unlike malicious bots, report servers separate entities used to infect vulnerable devices reported by infected bots. Therefore, the IP corresponding to the extracted URL is classified as a Payload server used by the Report Server to download and execute malware from within infiltrated devices.

If no URLs are extracted, the packet's TCP sequence and target port are checked for botnet-like port-scanning characteristics. The port-scanning requests emitted by Mirai botnets are observed to include a TCP sequence fingerprint, which is used to infer if a packet is that of

Mirai-like scanning activity. Similar assumptions are inferred based on the packet's targeted port, such as sniffing open telnet ports 23 and 2323. If any scanning activity is identified, the packet's source IP is classified as a "Bot", a benign botnet entity. Otherwise, the packet is deemed insignificant and is discarded. Following this, the worker will begin to collect information for all classified IP addresses, including the geographic location, WHOIS lookup information, proxy/tor/VPN detection, etc... The relationships between IP addresses are also recorded in a directed graph `G = (N, E)`, where N is the set of all IP addresses (Nodes), and E is the set of all IP address relationships (Edges). For example, `N(Source IP) -> N(Report Server) -> N(Payload Server)`.

The final packet analysis stage requests a malware analysis task for the extracted URL(s) by sending an API request to a remotely hosted malware sandbox service. Pending malware analysis tasks are logged in a database, allowing the system to manage and process incoming malware analysis reports from the external sandbox. Once the analysis is complete, the VisiBot Processing System will receive a malware analysis report through a Web API and process it accordingly. An overview of the interaction between the VisiBot Processing System and sandboxing service is depicted below:

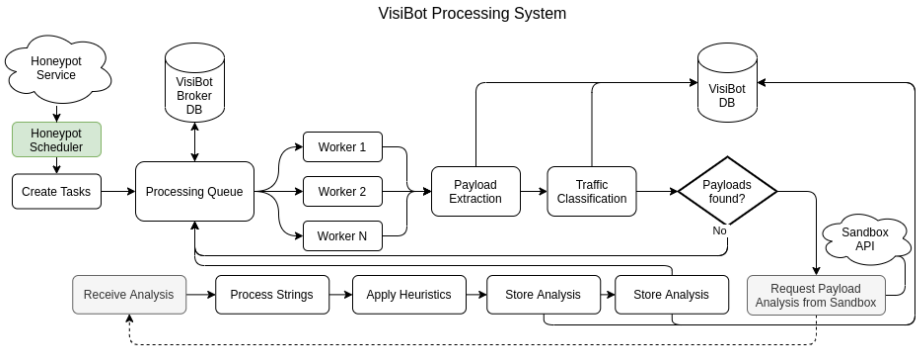


Figure 2.1: *High-level flow-diagram of the core VisiBot Processing System.*

2.2.3 Scalable Sandboxing Service

Like the VisiBot Processing System, the employed sandbox analysis system also utilises the Message Broker architectural design pattern. The VisiBot Processing System's tasks are performed by several scalable and distributable workers, allowing for real-time malware analysis. Hosted on a separate server from VisiBot, all malware samples are stored within a secure block-storage partition separate from the rest of the system. Upon receiving an analysis request from an external source, such as VisiBot, the sandboxing system will attempt to download and analyse a binary from the URL given in the request. If downloaded successfully, the main process will move the binary to a secure block-storage partition with limited file-system permissions, completely separate from the main system. Following this, an analysis task is added to the sandbox broker queue.

When a worker consumes an analysis task, a three-stage analysis process is performed on the binary. The first stage involves static analysis, in which information such as the target CPU architecture, endianess, and source-code strings of the binary is extracted. Following this, the binary is copied to a virtual machine instance running an embedded Linux operating system. The image used by the Virtual Machine (VM) is selected based on the detected CPU architecture of the binary. Several CPU architectures are supported, including MIPS, ARM, i386, aarch64, and x86/64, all of which are common to targeted IoT devices. Once a new Virtual Machine instance is created, the malware binary is executed and monitored for 30 seconds, after which

the virtual machine is reset. During the monitoring procedure, dynamic and network analysis are performed, logging all incoming and outgoing network traffic, processes opened files, and system calls generated by the malware sample. This information is used to create and send a report to the VisiBot Processing System following task completion. The report includes various relevant static, dynamic, and network information, including port usage statistics, DNS queries, HTTP requests, and hard-coded binary strings. Additionally, the sandbox also generates log and pcap files which can be used for further analysis.

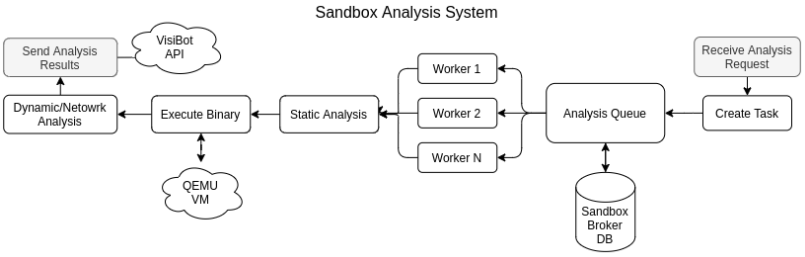


Figure 2.2: *High-level flow-diagram of malware sandbox analysis system.*

2.2.4 Heuristic Analysis

Once VisiBot receives a malware analysis report from the sandboxing service, heuristic analysis is performed on the sample report to identify candidate Command & Control servers and Peer-to-Peer botnet activity, similarly to that of Bastos et al. [16], Ceron et al. [17], and Herwig et al. [20]. The first heuristic is concerned with identifying Peer-to-Peer botnet activity by analysing DNS queries performed during network analysis. All logged DNS queries are matched against a list of domain names corresponding to Peer-to-Peer Distributed Hash Table services, including commonly used public services offered by bittorrent.com, transmissionbt.com, and utorrent.com. Modern P2P Botnets, such as Mozi, connect to public DHT services to mask peer-to-peer botnet interactions amongst legitimate peers. Thus, the identification of such public DHT DNS names is ideal for inferring the presence of peer-to-peer botnet traffic.

The second heuristic considers all transactional interactions with public IP addresses as potential C2/P2P activity. A transaction occurs when bytes are both sent to and received from an endpoint. Such data transactions are significant when identifying botnet traffic, as devices infected with botnet malware often attempt to perform handshakes and download updates/binaries from Command & Control servers, Payload servers, or peer-to-peer botnet nodes.

The third heuristic combines entities of static and dynamic analysis. If any connections are made to hard-coded IP addresses extracted during static analysis, the associated IP address is deemed a candidate C2 server. Lastly, the fourth heuristic uses external IP blacklist services to infer that any IP address endpoints logged during network analysis previously blacklisted for Command & Control activity are deemed malicious and are considered potential C2 servers. Once candidate C2 servers and P2P Nodes have been identified, the meta-data for each IP address is stored similarly to that of the first-stage classification process, storing geographical information, ASN records, etc. However, as the above heuristics are limited to the detection of *candidate* botnet servers, all endpoints must be further validated through proper handshake confirmation procedures. This process is yet to be implemented within the VisiBot Processing System. However, the system can be easily extended to do so. The modularity of the system, provided by broker-based task processing, ensures that additional analysis stages can be added to the VisiBot Processing System with ease. For a complete high-level overview of the VisiBot Processing System, including remote sandbox integration, see Appendix A.1.

2.3 Interactive Web Application

To allow for real-time visualisation of botnet propagation, geographic distribution, and density, an interactive web-based application is used to display the geographic coordinates of all botnet entities identified within the last 24 hours of monitoring. Using the geographic coordinates collected during IP address meta-data collection, botnet entities are pinned onto a world map, using a specific colour to represent different entities, such as Bots, Report servers, C2 servers, and P2P nodes. A variety of information is sourced directly from the VisiBot database and can be viewed through a front-end web interface that communicates with a back-end web server:

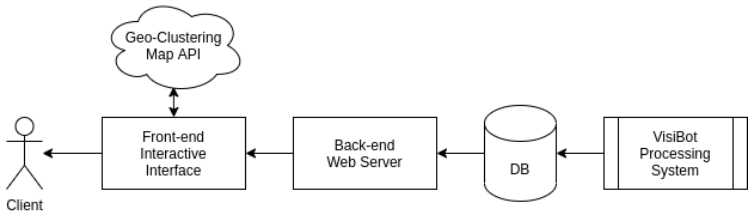


Figure 2.3: *High-level overview of the VisiBot Web Interface.*

Due to the amount of graphics being re-rendered whenever the map is updated, plotting thousands of geographic coordinates on a single world map can result in high resource usage. A greedy geographic clustering algorithm is used to cluster coordinates into groups based on the geographic distance between points, mitigating visualisation performance issues:

```
for each marker:
   if any cluster within cluster_distance of marker
      join(marker, cluster)
   else if any unclustered marker within cluster_distance of marker
       create_cluster(marker, unclustered_marker)
```

Listing 2.1: *Pseudocode for Greedy clustering algorithm based on an example by Agafonkin [22]*

Once clustered, the marker clusters are drawn in the map in place of individual markers. When selected, the map will zoom in and un-cluster markers outside of the specified clustering distance. When a marker is selected, the visitor can view details about the entity and visualise its relationships with other entities recorded by VisiBot.

3 Design Overview

3.1 Problem Analysis

Preemptive botnet detection has often challenged authorities, as real-time, large-scale traffic analysis is often expensive and very difficult to coordinate. Current detection systems are often reactive, as they often rely on post-attack data sources and network logs from Internet Service Providers and DNS services. This approach is not ideal, as attacks must first take place to identify offending networks. The preemptive botnet detection technique proposed by Moon et al. [12] proved significant in detecting botnets before attacks took place. The purposeful execution and analysis of malware enabled botnets to be identified and mitigated during their propagation periods. This system allows for the automatic collection and analysis of binaries using two dedicated processing modules.

However, several issues are also presented within the detection framework proposed by Moon et al.. A primary concern is that the data collection methods employed are only achievable through integration with an Internet Service Provider (ISP) infrastructure. Thus, it isn't easy to reproduce the same data collection process without internal access to an ISP service. Additionally, the amount and type of information sourced from the KORNET ISP service varied significantly in scope and relevance to botnet propagation. The vast majority of binaries analysed were sourced from spam emails and malicious websites reported by KORNET users, with a significantly smaller number of binaries collected through botnet honeypots. The analysis of binaries collected through a combination of extraneous data sources led to a trivial detection accuracy of 40%, stipulating that a more refined data-collection process is required.

As botnets have evolved to target vulnerable IoT devices, conventional honeypot detection systems, such as Honeyd [13], have proven ineffective in identifying and monetising from IoT botnet traffic. The methods proposed by [14] and [15] allow for the detection of such IoT traffic through configuring honeypot servers to mimic characteristics of commonly targeted IoT devices. Despite proving successful in collecting IoT malware, both implementations resulted in sparse data collection due to small-scale deployment and honeypot distribution.

The method proposed by [14] combines IoT honeypots and sandboxing environments using embedded Linux virtual machines, allowing for the purposeful infection of sandboxes and monitoring all incoming telnet based botnet communication. However, each honeypot system has to be frequently (and manually) reset as successful intrusions often result in unpredictable or unwarranted behaviour. Additionally, the honeypot machines would only monitor and redirect telnet-based intrusion attempts. This system overlooked important information by exclusively monitoring telnet traffic by analysing other commonly used botnet communication protocols, including SSH, HTTP, P2P, and IRC traffic. Since modern botnets frequently operate across various communication protocols, the exclusion of such protocols may result in the loss of valuable network traffic information for botnet identification.

Bastos et al. [16] and Ceron et al. [17] further build upon the concept of IoT honeypot system integration through extracting, executing, and examining a set of malware samples collected by distributed sandbox environments. The long-term distribution of honeypots across 15 Brazilian states allowed for mass collection of IoT malware binaries. However, the lack of global honeypot

distribution may result in collected data sets being skewed by regionally dominant botnets. Botnet activity is identified through a combination of static, dynamic, and heuristic analysis techniques using information obtained throughout the sandboxing process. In both implementations, malware samples are statically and dynamically analysed through being executed in a secure sandbox environment called Detux. [18] Throughout execution, all incoming and outgoing network traffic is logged for analysis. However, despite the sandbox having support for up to 5 CPU architectures, it is constrained in terms of functionality, only allows for basic static and dynamic analysis during runtime, and is not scalable out of the box. These factors limit the possibility of real-time static and dynamic malware analysis of honeypot traffic.

Dwyer et al. [19] employs a solution that allows for accurate botnet detection without the requirement of malware execution and analysis of malware samples. The proposed DNS feature classification technique gave high botnet detection accuracy for Mirai botnets; however, this solution cannot be applied to de-centralised botnet structures. As de-centralised botnets use P2P protocols to communicate instead of standard protocols such as IRC, HTTP and DNS, Peer-to-Peer botnets collectively communicate messages between connected peers rather than directly through a C2 server, mitigating the need for DNS-based communication.

Herwig et al. [20] illustrates how modern IoT botnets are being coordinated through de-centralised Peer-to-Peer networks via Distributed Hash Tables (DHT). Despite discussing remote code execution tactics commonly observed from propagating Hajime botnets, little insight is given regarding the payload extraction process. As malware URLs within remote code execution attempts are likely to be obfuscated, it is unclear if this system accounts for obfuscation when extracting malware samples for analysis. Herwig et al. also propose that Hajime bots can be identified using public keys as identifiers, as IP frequently addresses change ownership. However, both the transition of ownership and geographical distribution of IP addresses may prove significant for in-depth botnet analysis. Such IP address metadata can help identify substantial patterns across geographic botnet density and malicious IP address ownership trading between Autonomous Systems.

As IoT devices become more popular amongst households and businesses and new vulnerabilities are exposed, the importance of real-time identification and profiling of botnet traffic has significantly increased. Various Internet Service Providers, services, and government entities seek to mitigate the problem of botnet-based cyber-attacks; nevertheless, the continual evolution of botnets has made cyber-attack mitigation exceedingly more difficult. Such parties require quick and efficient ways to identify, visualise, and monitor current botnet activity, such that botnets can be observed and countered during propagation. The above botnet identification methods show high reliability and accuracy for IoT botnet detection. Yet, many of the proposed solutions are computationally expensive, difficult to deploy, or do not account for botnets' evolutionary factors, as attackers are persistently seeking to exploit new vulnerabilities, attack new target devices, and minimise detection whilst doing so.

3.2 VisiBot Processing System

VisiBot is an alternative botnet detection framework that focuses on real-time, automatic malware extraction and analysis. Using a globally distributed honeypot service, the VisiBot Processing System employs automatic extraction and execution of IoT botnet malware through sandbox and heuristic analysis techniques using scalable and distributable technologies.

3.2.1 Honeypot Integration

The VisiBoT Processing System employs a vast network of honeypots strategically distributed across a diverse set of network providers located in North America, South America, Asia, Europe,

and Australia. The honeypot servers have been configured to emulate that of commonly targeted IoT devices, to maximise incoming botnet scanning activity and intrusion attempts. Once a honeypot detects an intrusion attempt, the corresponding packet information is sent to a central processing server for analysis and storage. Once received, the primary honeypot processing server analyses and classifies packets based on characteristics observed across various Common Vulnerability Exploits (CVE), post-data characteristics, and TCP sequences. By analysing the payload string, post data, and TCP sequences of incoming packets, the honeypot service can infer which CVEs are being exploited by matching string patterns. Once processed, the captured honeypot packets are made accessible through a REST API (Application Protocol Interface) accessible by the VisiBot Processing System.

3.2.2 Broker-based Packet Analysis

The packets collected from the external honeypot service are processed and analysed in parallel, enabling real-time and efficient malware extraction and traffic classification. Real-time analysis is achieved through the adoption of the Message Broker architectural design pattern. Also known as Integration Broker, a Message Broker is "a third-party intermediary that facilitates interactions between applications" [21], through the communication and transformation of messages between various independent programs by interacting with message broker through an API. A broker-based analysis system allows for the execution of time-consuming tasks to be distributed across multiple worker processes. As workers are entirely separate from the main application and interact with a broker through an API, they can be reliably scaled and distributed across multiple systems. Clients, such as the VisiBot Processing System's primary process, can connect to a broker and add, remove, or prioritise tasks. Connected workers actively consume and perform tasks from a central priority queue.

Once a packet is received, the main VisiBot process creates a new packet analysis task, passing the task name and packet contents to the message broker to be later processed by a worker. The first stage of analysis involves malware payload URL extraction from the payload and post_data contents of the packet. The URLs used in remote code execution requests or downloading and infecting devices can be extracted using a combination of Regular Expression and sub-string extraction techniques. If no strings match common URL regex patterns, such as `https?:`, then the attacker may be obfuscating URLs through command-line tool argument sub-strings. This prevents the automatic scraping of malware URL and requires URLs to be rebuilt from parsed argument strings. Through parsing the arguments of command-line tools commonly used for downloading binaries, such as `wget, curl` and `tftp`, the processing system can extract important sub-strings. Such strings include arguments such as IP addresses, domains, paths, and ports, which are used to reconstruct a full URL.

After all potential malware URLs are extracted, the worker will perform packet classification using the URL extraction process's information. Assuming a URL's corresponding IP address matches that of the packet's source IP address, the entity caught within the honeypot is deemed a malicious, self-propagating bot that self-hosts malware and actively performs port scanning and attacks on vulnerable devices. If the URL's IP address does not match the packet's source IP, the processing system infers that the source packet IP corresponds to a Report Server. Unlike malicious bots, report servers separate entities used to infect vulnerable devices reported by infected bots. Therefore, the IP corresponding to the extracted URL is classified as a Payload server used by the Report Server to download and execute malware from within infiltrated devices. If no URLs are extracted, the packet's TCP sequence and target port are checked for port-scanning characteristics. As the Mirai botnet includes a TCP sequence within port scanning requests, this sequence is used to infer if the packet resembles that of Mirai-like scanning activity. The request's targeted port is also used to infer if the packet matches that of generic botnet port-scanning activity, such as sniffing open telnet ports 23 and 2323. If any scanning activity is identified, the packet's source IP is classified as a "Bot", a benign botnet entity. Otherwise, the

packet is deemed insignificant and is discarded. Following this, the worker will begin to collect information for all classified IP addresses, including the geographic location, WHOIS lookup information, proxy/tor/VPN detection, etc... The relationships between IP addresses are also recorded in a directed graph `G = (N, E)`, where *N* is the set of all IP addresses (Nodes), and *E* is the set of all IP address relationships (Edges). For example, `N(Source IP) -> N(Report Server) -> N(Payload Server)`.

The worker's final stage of the analysis task is to request a malware analysis task of the extracted URL(s) by sending an API request to a remotely hosted malware sandbox service. Pending malware analysis tasks are logged in a database, allowing the system to manage and process incoming malware analysis reports from the external sandbox. Once the analysis is complete, the VisiBot Processing System will receive a malware analysis report through a Web API and process it accordingly. An overview of the interaction between the VisiBot Processing System and sandboxing service is depicted below:

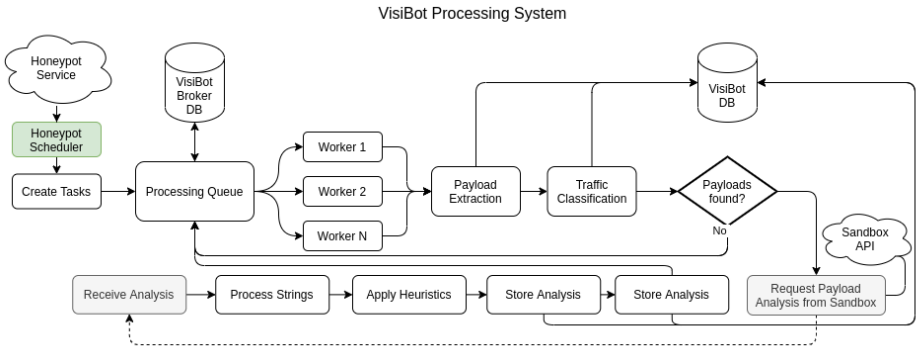


Figure 3.1: *High-level flow-diagram of the core VisiBot Processing System.*

3.2.3 Scalable Sandboxing Service

Like that of the VisiBot Processing System, the sandbox analysis system employed by VisiBot also utilises the Message Broker architectural design pattern. The VisiBot Processing System's tasks are performed by several scalable and distributable workers, allowing for real-time malware analysis. Hosted on a separate server from VisiBot, all malware samples are stored within a secure block-storage partition separate from the rest of the system. Upon receiving an analysis request from an external source, such as VisiBot, the sandboxing system will attempt to download and analyse a binary from the URL given in the request. If downloaded successfully, the main process will move the binary to a secure block-storage partition with limited file-system permissions, completely separate from the main system. Following this, an analysis task is added to the sandbox broker queue.

When a worker consumes an analysis task, a three-stage analysis process is performed on the binary. The first stage involves static analysis, in which information such as the target CPU architecture, endianess, and source-code strings of the binary is extracted. Following this, the binary is copied to a Virtual Machine (VM) instance running an embedded Linux operating system. The image used by the VM is selected based on the detected CPU architecture of the binary. Several CPU architectures are supported, including MIPS, ARM, i386, aarch64, and x86/64, all of which are common to targeted IoT devices. Once a new Virtual Machine instance is created, the malware binary is executed and monitored for 30 seconds, after which the virtual machine is reset. During the monitoring procedure, dynamic and network analysis are performed, logging all incoming and outgoing network traffic, processes opened files, and system calls generated by the malware sample. This information is used to create and send a

report to the VisiBot Processing System following task completion. The report includes various relevant static, dynamic, and network information, including port usage statistics, DNS queries, HTTP requests, and hard-coded binary strings. Additionally, the sandbox also generates log and pcap files which can be used for further analysis.

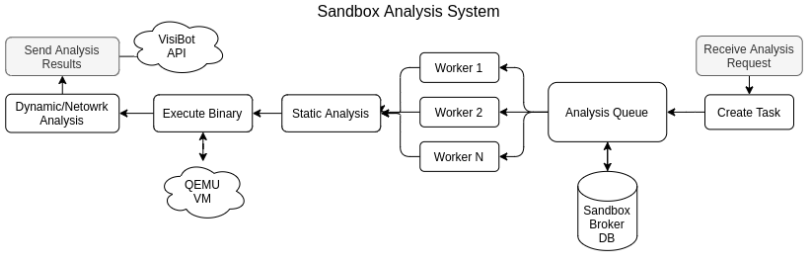


Figure 3.2: *High-level flow-diagram of malware sandbox analysis system.*

3.2.4 Heuristic Analysis

Once VisiBot receives a malware analysis report from the sandboxing service, heuristic analysis is performed on the sample report to identify candidate Command & Control servers and Peer-to-Peer botnet activity, similarly to that of Bastos et al. [16], Ceron et al. [17], and Herwig et al. [20]. The first heuristic is concerned with identifying Peer-to-Peer botnet activity by analysing DNS queries performed during network analysis. All logged DNS queries are matched against a list of domain names corresponding to Peer-to-Peer Distributed Hash Table services, including commonly used public services offered by bittorrent.com, transmissionbt.com, and utorrent.com. Modern P2P Botnets, such as Mozi, connect to public DHT services to mask peer-to-peer botnet interactions amongst legitimate peers. Thus, the identification of such public DHT DNS names is ideal for inferring the presence of peer-to-peer botnet traffic.

The second heuristic considers all transactional interactions with public IP addresses as potential C2/P2P activity. A transaction occurs when bytes are both sent to and received from an endpoint. Such data transactions are significant when identifying botnet traffic, as devices infected with botnet malware often attempt to perform handshakes and download updates/binaries from Command & Control servers, Payload servers, or peer-to-peer botnet nodes.

The third heuristic combines entities of static and dynamic analysis. If any connections are made to hard-coded IP addresses extracted during static analysis, the associated IP address is deemed a candidate C2 server. Lastly, the fourth heuristic uses external IP blacklist services to infer that any IP address endpoints logged during network analysis previously blacklisted for Command & Control activity are deemed malicious and are considered potential C2 servers. Once candidate C2 servers and P2P Nodes have been identified, the metadata for each IP address is stored similarly to that of the first-stage classification process, storing geographical information, ASN records, etc. However, as the above heuristics are limited to the detection of *candidate* botnet servers, all endpoints must be further validated through proper handshake confirmation procedures. This process is yet to be implemented within the VisiBot Processing System. However, the system can be easily extended to do so. The modularity of the system, provided by broker-based task processing, ensures that additional analysis stages can be added to the VisiBot Processing System with ease. For a complete high-level overview of the VisiBot Processing System, including remote sandbox integration, see Appendix A.1.

3.3 Interactive Web Application

To allow for real-time visualisation of botnet propagation, geographic distribution, and density, an interactive web-based application is used to display the geographic coordinates of all botnet entities identified within the last 24 hours of monitoring. Using the geographic coordinates collected during IP address metadata collection, botnet entities are pinned onto a world map, using a specific colour to represent different entities, such as Bots, Report servers, C2 servers, and P2P nodes. A variety of information is sourced directly from the VisiBot database and can be viewed through a front-end web interface that communicates with a back-end web server:

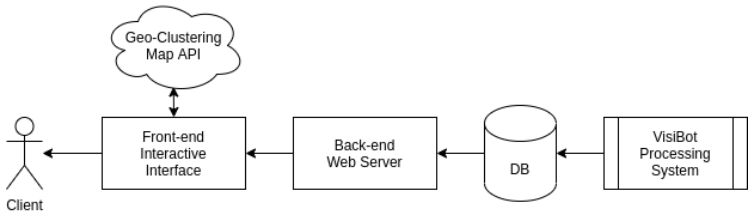


Figure 3.3: *High-level overview of the VisiBot Web Interface.*

Due to the amount of graphics being re-rendered whenever the map is updated, plotting thousands of geographic coordinates on a single world map can result in high resource usage. A greedy geographic clustering algorithm is used to cluster coordinates into groups based on the geographic distance between points, mitigating visualisation performance issues:

```
for each marker:
   if any cluster within cluster_distance of marker
      join(marker, cluster)
   else if any unclustered marker within cluster_distance of marker
      create_cluster(marker, unclustered_marker)
```

Listing 3.1: *Pseudocode for Greedy clustering algorithm based on an example by Agafonkin [22]*

Once clustered, the marker clusters are drawn in the map in place of individual markers. When selected, the map will zoom in and un-cluster markers outside of the specified clustering distance. When a marker is selected, the visitor can view details about the entity and visualise its relationships with other entities recorded by VisiBot. Using the directional graph information collected throughout the malware analysis process, the VisiBot Web Application can also render graph-based geographic visualisations of botnet entity interactions. When the connection feature is triggered for a specific geographic entity on the visualisation map, a series of undirected graphs are drawn between the selected entity and all other botnet entities. A selection of colour coordinated nodes and edges is drawn onto the world map using the latitude and longitude of each node's IP address associated with the currently selected node, allowing for the visualisation of geographic botnet distribution for a given botnet network. The network graph is generated for a specific IP address by performing a full graph search on the source and destination IP addresses of all connections logged by VisiBot.

4 Implementation

This chapter discusses the various implementation stages of the VisiBot Processing System and Interactive Web Application development and deployment. Such sections define the technologies, architectural design patterns, development practices, and deployment strategies used throughout the development process.

4.1 Distributed Real-time Processing System

4.1.1 Message-based task queues with Celery and Redis

Written in Python 3.9 [23], the VisiBot Processing System utilises several internal and external modules, services, and frameworks to create a flexible and reliable analysis system based on the Message Broker design pattern. I chose the Python programming language primarily due to the extensive availability of open-source community libraries, including various web development frameworks, API wrappers, and data analytic tools.

One such library used is Celery [24], a python-based asynchronous task queuing framework modelled on the Message Broker design pattern. Redis, an open-source data structure store commonly used for databasing, caching, and message brokering [25], In is used conjunction with Celery to enable the processing of vast amounts of messages across distributed worker instances. Through the queuing of programmatic tasks as messages, high quantities of synchronous, or otherwise, time-consuming tasks are actively consumed by Celery workers through interaction with a central broker system. Celery supports integration with various message brokers, including RabbitMQ, Amazon SQS, Redis, and Zookeeper. Within the context of this project, Redis was chosen as the primary message broker for VisiBot.

Celery tasks are created by an application and routed, stored, and eventually consumed through a broker system by a cluster of Celery workers. These workers can be distributed across several systems and configured automatically based on the queue's size and workload. Celery workers also support the execution of tasks in parallel. This enables the swift and concurrent execution of time-consuming tasks, which might otherwise bottle-neck synchronous applications. The VisiBot Processing System utilises Celery workers to analyse, identify, and extract botnet malware samples from honeypot sources and analyse them within automated sandbox environments. As this process can be time-consuming on synchronous systems, the adoption of a Message Broker design pattern through Celery and Redis ensures that analysis tasks are completed concurrently and reliably.

Flower, a python-based Celery monitoring tool by Movsisyan and contributors [26] offers a remotely accessible web application dashboard, which allows for the monitoring task queues, celery workers, and successful or failed tasks. As the stack traces for all unsuccessful tasks are logged by Celery, such traces are remotely accessible through accessing the Flower dashboard, allowing for a convenient means for identifying and debugging worker run-time exceptions from within a deployed environment. The flower web interface has been protected with user authentication, as it allows users to remotely manage workers and view potentially confidential information, such as stack traces. A screenshot of the dashboard is shown below:

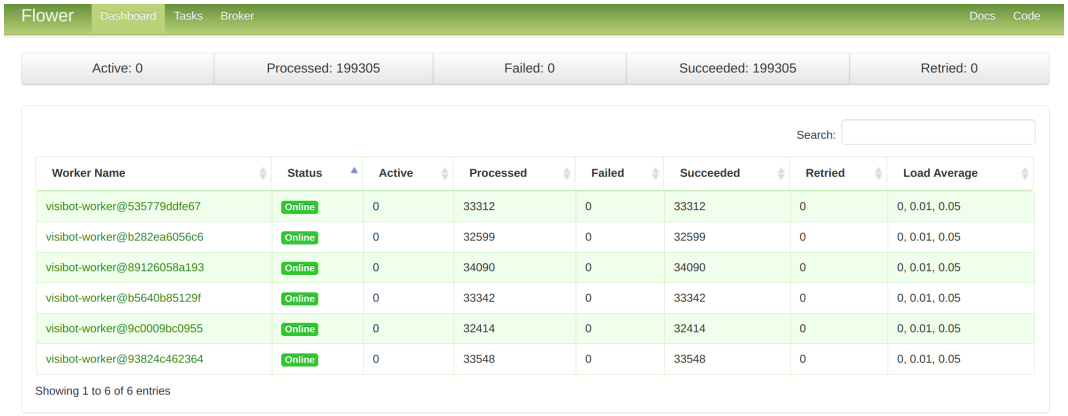


Figure 4.1: *Flower Dashboard. Celery workers are represented in a table as rows.*

4.1.2 Docker

As the proposed system utilises various applications, celery workers, servers, and external services, the setup and deployment process can introduce many complexities and anomalies due to various environmental factors such as different operating system, package managers, and system architecture. Thus, to streamline the setup and deployment process, all applications, workers, and services utilised by VisiBot have been encapsulated within a multi-container Docker application using the command-line tool Docker Compose [27].

A unique feature provided by Docker is that of image scalability. By containerising a single celery worker within a docker image, it can dynamically be scaled to allow multiple celery workers' instantaneous creation. The number of workers is scaled by appending the `docker-compose` command with the `–scale worker N` argument. For example, running the docker application with the argument `–scale worker 10` will prompt Docker to create 10 VisiBot Celery worker containers. Each container will initialise a new celery worker instance and connect to the Redis broker service, which is also containerised within a docker image. All other services and applications used by the VisiBot workers and processing system have also been containerised, including a locally accessible tor server, honeypot result scheduling system, Flask Web API, MaxMind GeoIP2 Updater, and a remotely accessible instance of Flower. Docker also enables shared access to storage between multiple containers through the creation of containerised volumes. Additionally, an `.env` file, initially populated with a set of default environment variables, is automatically read by docker-compose upon initialisation. This allows the user to change predefined aspects of the processing system, such as API keys and paths, without directly modifying any of the code.

As VisiBot workers use shared MaxMind GeoIP2 databases [28] during analysis tasks, a shared volume was created such that the VisiBot workers could query a shared database hosted within the GeoIP container. Upon initialisation, the GeoIP creates a Cron-job using `crontab` [29] which, executes a GeoIP2 updating utility every week. This utility writes the latest versions of all MaxMind GeoIP2 databases to `/usr/local/share/GeoIP2`. By modifying the docker configuration of VisiBot's `docker-compose.yml`, this path was defined as a shared path accessible to all workers. A visualisation of the relationships between all docker containers within the VisiBot Processing System is shown below:

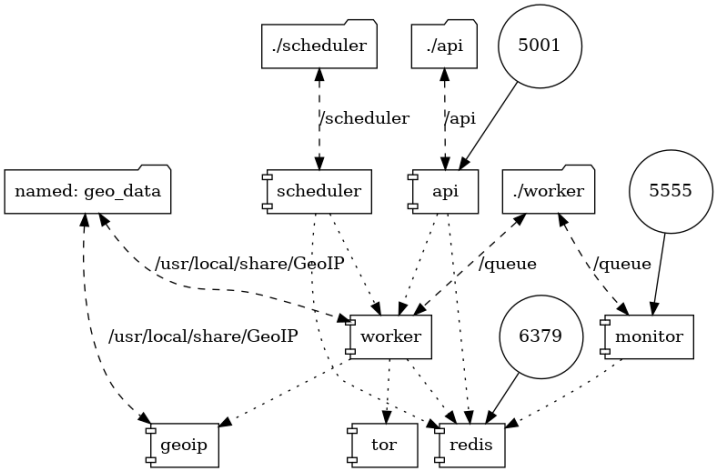


Figure 4.2: *Visualisation of VisiBot Docker-Compose Containers. Reverse Engineered using docker-compose-viz. Containers are represented as rectangles and exposed container ports as circles.*

4.2 VisiBot Processing System

4.2.1 Database Record Storage using MongoDB

Developed by MongoDB Inc. [30], MongoDB is a Document-Oriented No-SQL database with extensive cross-platform support, allowing for the storage of documents represented in JavaScript Object Notation (JSON), a highly supported human-readable standard format. The VisiBot Processing System uses MongoDB widely to manage and store complex, and in some cases, nested, data objects through MongoEngine, an open-source python object data mapper for MongoDB [31]. As Documents are represented in a standard JSON format, database entries can be easily created, managed, and accessed by mapping MongoDB documents to standard Python objects and types, such as dictionaries and lists. Additionally, as this file format is widely accepted by various other programming languages, libraries/frameworks, and software applications, data collections stored within MongoDB Documents can be easily imported/exported, manipulated, and analysed by various popular data science, analytics and visualisation libraries.

Like relational databases such as PostgreSQL and MySQL, entity relationships of various cardinalities can also be established within MongoDB. However, unlike traditional relational databases, associations such as One-to-One and One-to-Many can be represented using embedded MongoDB documents (JSON Objects). This allows for the nesting of documents such that relational attributes are easily accessible and represented logically. Additionally, MongoDB allows for representing One-to-Many relationships by creating and referencing primary keys attributes through document reference fields. VisiBot uses such relationships, enabling the linkage of documents containing information for botnet IP addresses, Autonomous Systems, malware analysis reports, and more. A complete Entity Relationship diagram of all of the VisiBot Processing database is shown in Appendix A.2.

All data collected by the VisiBot Processing System is securely stored, managed, and accessed through a remotely accessible instance of MongoDB hosted through the MongoDB Atlas [32].

4.2.2 Honeypot Data Collection

All honeypot information processed by VisiBot is provided by Bad Packets LLC [10], a leading provider of Cyber Threat Intelligence information on emerging threats, DDoS botnets, and network abuse. The honeypots hosted by Bad Packets are strategically deployed across a diverse set of network providers based in multiple countries and regions, including Australia, Brazil, Canada, France, Germany, India, Japan, Netherlands, Russia, Singapore, Taiwan, The Middle East, the United Kingdom, and the United States. Such honeypot servers are configured to emulate commonly targeted hosts specific to botnet traffic by mimicking various IoT devices, consumer-grade routers, enterprise VPN endpoints. Additionally, sinkhole domains previously used by threat-actors are employed to point potential botnet traffic to the honeypots.

Accessible through an authenticated REST API, the Bad Packets honeypot service is queried on an hourly basis from within a containerised task scheduler application written in Python. Using the python `requests` library [33], the BadPackets API is repeatedly queried for honeypot packet results. Each result is converted into a Python dictionary and is appended to a list containing all results. An example Bad Packets honeypot result is shown in Appendix A.3.

Once all results within the last hour have been collected, the scheduler connects to the VisiBot Celery broker (Redis), hosted within a neighbouring docker container, and begins to create a Celery analysis task for each packet result. Task creation is requested through the client executing `Celery.send_task`, passing the task name as a string and the result dictionary for analysis as an argument. Following this, tasks are created as pending messages within the broker and are actively consumed (executed) by available Celery workers. Once task creation has completed, the scheduler will wait until the next hour before extracting more results from Bad Packets, as new packet data is made available through the API at the beginning of every hour. All Celery can be viewed in the Flower web dashboard hosted from within a separate docker container:

```
VisiBot collection event activated. Collecting data from Bad Packets API.
Querying Bad Packets (last seen after 2021-03-01T21:00:00Z)
Queried 927/927 results
Creating processing tasks for 927 results.
Workers are now running tasks in the background. Visit http:/flower:5555 to view
    progress.
```

Listing 4.1: *Example output of VisiBot Processing Scheduler.*

Upon receiving a task from the broker, a Celery worker will perform an analysis of the received packet data. As the analysis procedure involves actively reading and writing from a database, each worker is configured to execute at most one concurrent job from the broker at any given time to avoid database connection locks. The honeypot packet analysis process is separated into multiple stages, spanning from malware payload URL extraction and traffic classification to sandbox analysis initialisation.

4.2.3 Malware payload extraction

As previously discussed, when a device is infected with botnet malware, it will begin to scan for open ports of vulnerable devices and will attempt to infect such devices by sending malicious HTTP requests that contain command-line injection code. The injected code, also known as a Remote Code Execution attempt, will likely try to download and execute a malware binary using command-line utilities such as `wget`, `curl`, or `tftp`. Such exploit attempts are commonly found within the `payload` or `post_data` of an offending packet. By extracting the URL of the botnet malware payload, the binary can later be retrieved, executed, and analysed.

Noise removal and de-obfuscation The first stage of malware payload extraction involves the removal of noise and elimination of the various obfuscation attempts employed by botnet developers. Due to the high number of different devices being targeted and exploits used, the structure and syntax of the command-line injection code captured by honeypots can vary significantly. Consider the following payloads strings:

```
GET /language/Swedish${IFS}&&cd${IFS}/tmp;rm${IFS}-rf${IFS}*;wget${IFS}
http://[redacted_ip]:54134/Mozi.a;sh${IFS}/tmp/Mozi.a&>r&&tar${IFS}/string.js
    HTTP/1.0

GET /shell?cd /tmp; cp /bin/busybox yeet; >yeet; chmod 777 yeet; nohup wget
http:/\/[redacted_ip]:80/arm7 -O yeet || nohup tftp -r arm7 -g [redacted_ip] -l
    yeet;
chmod 777 yeet;./yeet Windows; rm -rf yeeter >/dev/null 2>&1 HTTP/1.1
```

Listing 4.2: *Example of noisy and obfuscated botnet payloads. Examples of noise are highlighted in* ***bold****, and obfuscation in italics.*

As shown above, the first payload contains a lot of noise. The `${IFS}` variable is used as an input field separator in place of space characters. The second payload contains some light obfuscation employed to prevent regular expression patterns from matching URLs in the payload string. The first payload URL downloaded using `wget` has an escaped forward slash such that the pattern `http://` cannot be matched. Should the infiltrated system not have `wget` installed, the injected code also tries to download the binary using the `tftp` client. However, the malware payload URL is obscured using various command-line arguments, as the host of the URL is specified using the `-g` flag, the path using the `-r` flag, and the output using `-l`. These URLs can be re-built, but the presence of noise such as `${IFS}`-based spacing can make this process difficult. A shortlist of noisy characters and sub-strings is created through the manual inspection of several payload strings and post_data contents. The list is used throughout the cleaning process, removing any significant noise from both the payload and post_data packet strings. Each string is decoded such that any URL encoded values, such as `%20` spacing, are converted into plain text. Similarly, occurrences of the sub-string `${IFS}` are replaced with space characters, and the removal of inconsistent spacing before/after command-line semi-colon separators (`;`) is also implemented. Lastly, the removal and/or replacement of the characters `\"'‘&|()+` ensure that all URLs and `wget/curl/tftp` commands are clean.

Payload URL extraction and validation Following the removal of noise, both the payload and post_data strings are combined into a string which is parsed for payload URLs using several techniques. A conventional URL regular expression pattern is used to match all un-obfuscated URLs contained within the payload strings. However, as this expression matches all sub-strings that begin with `https?://`, a large proportion of partial or obfuscated payload URLs will not match this regular expression. For example, consider the following sub-strings:

- `curl website.com/bin;`
- `wget 127.0.0.1:8080/bin;`
- `tftp -g 127.0.0.1:8080 -r bin;`

Due to Regular Expression matching limitations, the extraction process also utilises other methods of URL extraction. The python library `urlextract` [34] partial URL strings containing domain names, such as `website.com/bin`, by matching any occurrences of Top Level Domains (TLD), such as `.com, .net` and `.org`, until a stop-character, such as white space, is reached. This method allows for the extraction and validation of domain names used for botnet propagation, as partial domain-based URL strings are identified and re-formatted using the standard HTTP

URL format. These strings do not indicate a URL schema; therefore, the `http://` schema is used by default.

The practice of obfuscating payload URLs through executable arguments was frequently commonly observed across inspected honeypot payload command-line injection strings, as shown in the example sub-strings above. However, the VisiBot Processing System is capable of performs automatic re-construction of argument-based obfuscated URLs, particularly from data-transfer command-line tools such as `curl, wget` and `tftp`. A utility function is used to identify the position at which commands such as `wget`, `curl`, and `tftp` are called, and the arguments following the start of the command are parsed until an end-of-line semi-colon (`;`) character is reached. Depending on the executable that is called, the function will look for and parse the contents of specific command-line arguments in order to extract the `host`, `port` and `path` of the obfuscated URL. For example, the helper function may identify usage of the `tftp` command and will extract the values passed within the `-g` and `-r` arguments, extracting the host, port, and path of the payload URL from the payload command-line injection string. Once extracted, the base URL is re-built through formatting extracted host, port, and path values within a standard http URL format:

```
url = "http://{host}:{port}/{path}".format("127.0.0.1", 8080, "bin")
>>> "http://127.0.0.1:8080/bin"
```

Following this step, GET requests are sent to each URL. The response data is parsed to identify if the URL leads to a binary executable, bash script, or non-executable file. All HTTP requests are routed through a Tor Proxy server hosted in a separate docker container from the celery workers for anonymity. All Non-executable URLs are disposed of, and identified binary executables are immediately added to the list of URLs awaiting validation. Binary execution is identified by checking if the HTTP request's content-type header begins with `application/`, or the contents of the request contains an Executable and Linkable Format (ELF) header. However, a current limitation is that botnet propagation bash scripts used in Remote Code Execution attempts cannot be executed within the LiSa sandbox environment. To combat this constraint, the worker will parse the contents of any bash script URLs and extract and store any nested URLs. Bash scripts often include links malware for various architectures, leading to VisiBot effectively extracting up to 6 (or more) binaries from a single bash script. Bash scripts are identified by checking if the header content-type starts with `text/` and the content of the request contains any of the following sub-strings: `["#!", "wget", "curl", "tftp"]`. The first sub-string, `#!`, represents a shebang character sequence read by Unix-based operating systems to indicate which interpreter should be used when executing a bash script [35]. For an example of a multi-binary bash script used for botnet propagation, see Appendix A.4.

Upon collecting all malware binary URLs, each the Top Level Domain (if present), IP address, and host corresponding to each URL is validated before proceeding with malware binary analysis. Using python libraries `tld` and `tldextract` [36, 37], the Top Level Domain of each URL is extracted and checked against a list of ignored TLDs, including `.gov, .mil` and `.arpa`. The host-name and IP address of each URL are identified using the built-in python `socket` library. [38] Once identified, the URL, host-name and IP address are used to create a new `MalwarePayload` database entry. If an instance of the URL already exists, its corresponding entry is updated, and the URL is dropped from the list of URLs awaiting malware analysis. When creating an entry, the worker will determine if the malware payload is self-hosted by the source IP address attempting to infect the honeypot by matching the packet's source IP address with the IP address identified from the extracted URL.

4.2.4 Botnet traffic classification

Following the extraction of URLs from a given honeypot packet, the worker will infer the classification of both the source and payload IP addresses based on the context of the extracted URLs and the command-line code injection string's contents. If one or more URLs is successfully extracted and at least one URL is self-hosted by the packet's source IP address, the worker will classify the source/payload IP address as a "Malicious Bot". Bad actors use malicious Bots to self-propagate botnets without the need for an external payload server. Malware binaries are downloaded onto the infected machine and used as a download host when infecting other devices.

If the IP address of the payload URL does not match the source IP, the payload host is classified as a "Payload Server", and the source IP address is classified as a "Report Server". As seen in botnet variants such as Mirai, as infected bots scan for vulnerable devices, they will report such findings to a report server. The report server will attempt to infect the device through command-line injection, using malware remotely hosted on a separate payload server. For simplicity, both infected hosts and report servers that attempt to infect devices using malware hosted on payload servers are classified as report servers.

Alternatively, if no URLs are extracted from the honeypot packet payload or post_data strings, the worker will use additional information provided by the Bad Packets API to determine if the packet contains characteristics representative of passive botnet activity, such as port scanning. By monitoring packets received by honeypots, all packets are evaluated in terms of target port usage and payload exploit characteristics to identify various Common Vulnerability Exploit (CVE) tags and descriptive categories such as "Mirai-like scan". Bad Packets detect Mirai-like scans by analysing the TCP sequence number of a given packet and checking if it is equal to the targeted device's IP address, as shown in the Mirai source code snippet shown below. [39]

```
tcph->seq = iph->daddr;
tcph->check = 0;
tcph->check = checksum_tcpudp(iph, tcph, htons(sizeof (struct tcphdr)), sizeof (struct tcphdr));
```

Figure 4.3: *Mirai TCP sequence source-code. Snippet provided by Bad Packets LLC [10]*

Using the additional tag information provided by Bad Packets, VisiBot can infer if a packet is associated with botnet scanning activity. Any honeypot packet that does not contain a payload URL is classified as 'Bot' if it demonstrates either Mirai-like or general port scanning activity, which is determined based on its TCP signature or target port.

4.2.5 Honeypot Packet Processing

Following the identification and classification of a packet's source IP address (and any associated payload servers), additional IP address metadata is collected using various heterogeneous data sources. The geographic information for each classified IP address is looked up using locally hosted MaxMind GeoIP2 databases [28], which is then stored in the database for future use. Such information includes the latitude and longitude, city, country, and the continent of a given IP address. Privacy and hosting information for a given IP address is also identified using the IPInfo Privacy Detection API [40], allowing for identification of servers provided by hosting services and the detection of a Proxy, VPN, or Tor service. Lastly, Autonomous System history information is collected using the `ipwhois` python module. This module performs an IP WHOIS lookup for a given address and returns Autonomous System information sourced from Internet Service Registries including AFRINIC, APNIC, and RIPE NCC [41, 42, 43].

Following the IP address metadata collection, a new event log entry is created for each IP address. Each entry indicates the event classification label and the time and date of occurrence. Additionally, all relationships between a given packet's source IP address and payload IP addresses

are stored within the database. IP address connections are specified by including the IP addresses of the source and destination of a connection and a 'last seen' timestamp. Lastly, the packet JSON data sourced from the Bad Packets honeypot service is also stored directly within the VisiBot database. Each record contains valuable information for botnet analysis, including the targeted port, request user-agent, Bad Packets tags, and event count. To visualise the overall structure of the VisiBot database, please refer to the entity-relationship diagram shown in Appendix A.2.

Following the above steps, the VisiBot worker will continue by iterating over the list of extracted payloads URLs and sequentially send an analysis task creation HTTP request to a remotely hosted malware sandboxing service. Within each API request, the worker specifies the URL for analysis and the execution time in seconds, with a default of 30 seconds. Following the successful creation of each malware analysis task, the sandbox server will respond with a unique `task_id` identifier, which is stored within the VisiBot database alongside the ID of the payload being analysed. Upon analysis completion, the sandboxing server will send an analysis report to the VisiBot Processing System through a REST API, which is further analysed for candidate botnet Command & Control server and Peer-to-Peer node identification. A complete low-level overview of the VisiBot Processing System is shown in Appendix A.5.

4.2.6 LiSa Sandbox Integration

Developed by Daniel Uhříček, the LiSa Sandbox [11] is an open-source, scalable, and automated Linux Sandbox platform. It supports the execution and analysis of malware binaries compiled to various CPU architectures, including MIPS, ARM, x86_64, i386, and aarch64. Like the VisiBot Processing System, LiSa is also a multi-container docker application. It invokes the message-broker design pattern using Celery and RabbitMQ as message broker [24, 44] and is accessible through both a web dashboard and REST API. By default, the LiSa sandbox allows for remote static and dynamic analysis of binary executables but can be extended to include additional analysis stages. The number of workers used to analyse binaries can be scaled in the same way VisiBot workers are scaled, using the `-scale worker=N` argument when starting the service using `docker-compose`. This allows for easy distribution of analysis tasks, enabling multiple binaries to be processed at a given time.

Static analysis information is directly extracted from uploaded binaries using Radare2 [45], a comprehensive binary analysis framework, allowing for the identification of a given malware binary's target CPU architecture, endianness, operating system, and more. Dynamic and network analysis is performed through the direct execution of a malware binary within a sandbox environment. Each binary is executed within the QEMU virtual machine [46] using an embedded Linux image that matches the detected binary CPU architecture. As LiSa supports five different architectures, an embedded Linux image has been created for each architecture using Buildroot [47]. If an unsupported architecture is detected, the task will purposefully fail, and the payload will be processed by VisiBot accordingly. A variety of dynamic analysis information is logged during execution, including packet data, machine log information, process trees, system calls, opened files, and program output. Using the collection of static and dynamic data collected, LiSa generates a report for each malware analysis task which contains network analysis information such as IP address endpoints, HTTP requests, DNS queries, telnet data and IRC messages. Users are also given the option to provide an API key for automatic aggregation of various anti-virus scanning services through VirusTotal [48]. As the VirusTotal scan reports of identified malware binaries are actively used throughout VisiBot analysis, this service was facilitated using an academic VirusTotal API key.

Modifications to LiSa Despite the extensive feature-set provided by the default LiSa sandbox configuration, some limitations were encountered when integrating the sandbox into the VisiBot Processing System. Several modifications were made to the LiSa sandbox source code via a public GitHub fork [49] to mitigate such issues. A significant issue was that binaries had to be

provided as file attachments by default when creating analysis requests through the LiSa REST API, meaning VisiBot would first have to download binaries before uploading them to LiSa. As this is not ideal, the LiSa sandbox API has been modified to accept an additional `url` parameter for the URL of a binary to be provided instead of the file itself, mitigating any security issues presented by downloading malware directly onto the VisiBot server.

Several malware binaries were observed to have been packed using packing software to obfuscate source code strings extracted during the LiSa static analysis process. Any attempt to extract strings from a packed malware binary would result in unreadable, compressed, and thereby heavily obfuscated output. Malware authors incorporate binary packing as a means of additional obfuscation, as packed binaries often contain strings such as IP addresses. These binaries are unpacked into the program's address space at run-time, [50] ensuring that analytical software tools, such as the `strings` command-line tool [51] used by LiSa, output nonsensical information. Modifications were made to the LiSa sandbox to enable the automatic detection and unpacking of malware samples using the UPX packing tool [52]. UPX a popular tool used for obfuscating the contents of Linux/Unix executables. This tool leaves an identifier signature in the strings of a packed malware, allowing for quick identification when parsing the output of the `strings` command executed by the LiSa analysis worker. If the substring "UPX!" is found in the strings of a binary, the worker will attempt to unpack the binary using the command `upx –decompress path/to/file`. Following this, the string values of the unpacked binary are collected by re-running `strings` on the binary.

Lastly, when integrating the LiSa sandbox with the VisiBot Processing System, various communication-based limitations were encountered. As the LiSa sandbox's source code only allows for one-way communication through a REST API, there was no efficient way for the VisiBot Processing System to detect when analysis tasks were complete. An initial solution was to repeatedly query the LiSa API for task status updates, yet this proved highly inefficient and ultimately introduced several reliability issues. Thus, the VisiBot Processing System hosts a minimal REST API written in Flask, [53] used by LiSa sandbox workers to send back completed analysis tasks. Adding a REST API to the processing system allows for a robust two-way communication system between VisiBot and the LiSa sandbox server.

Hosted within a separate docker container, the VisiBot Flask API accepts two types of HTTP request from the LiSa Sandbox via the endpoints */api/lisa-analysis/success/<task_id>* and */api/lisa-analysis/failure/<task_id>* respectively. Additional modifications have also been made to the LiSa sandbox such that success/failure reports are sent back to the VisiBot API following task completion. If an analysis task succeeds, LiSa will send the malware analysis report back to the VisiBot Processing System for further analysis. Whereas, if an analysis task fails, stack trace information is returned to VisiBot to handle failed payloads. Responses are handled by creating corresponding Celery tasks, which are eventually consumed and carried out by available VisiBot workers. Failed tasks are processed based on the type of exception generated during analysis. For example, if a `UnicodeDecodeError` was raised, this indicates that the extracted payload URL points to a non-executable file. Thus, the MalwarePayload document (and all other related documents) is removed from the MongoDB database.

4.2.7 Malware Analysis Information Extraction

Before applying identification heuristics, the worker will first attempt to parse the top-most recurring malware keyword from an aggregated list of anti-virus vendor scan results via VirusTotal. However, many anti-virus vendors do not follow the same naming convention when labelling the type of malware detected. Thus, additional steps are taken to standardise the keyword extraction process by removing punctuation, case sensitivity, and generic keywords such as `Trojan, Malware, Worm`, etc. What remains is a list of unique keywords used by vendors to characterise the type of malware detected. The most commonly recurring keyword across

all vendor results is used as the primary keyword identifier for the analysed malware. For an example of this process, please refer to Appendix A.6.

Following keyword extraction, the worker will parse each of the binary strings collected during the static analysis stage of the LiSa sandbox analysis, such that any strings of significance, such as IPv4 and IPv6 addresses, URLs, and domains, are extracted using regular expression. As storing the entire list of malware strings proved too memory consuming, this step was necessary to reduce the amount of unnecessary data stored within the database.

4.2.8 Application of heuristics

Using the malware analysis report generated by LiSa, any significant network traffic, such as that of candidate Command & Control Servers and possible Peer-to-Peer botnet nodes, is detected through the direct application of the following four heuristics. Each heuristic is applied to the static or network analysis information confined within the report for a given malware sample to infer if the host running the malware has actively communicated with external botnet entities. The four heuristics applied throughout the heuristic analysis process are as follows:

i The infected host performs a Peer-to-Peer DNS Query during network analysis.
ii The infected host performs a data transaction with a foreign IP address.
iii Interaction detected between the infected host and hard-coded IP address.
iv Interaction detected between the infected host and blacklisted Command & Control Server IP address.

Heuristic (i) As inferred by its description, the first heuristic is primarily used to distinguish between centralised and de-centralised botnets through identifying a common Peer-to-Peer botnet characteristic. By analysing the DNS queries performed by a given malware sample during sandbox analysis, we can infer that the malware communicates through Peer-to-Peer networks if any of the domains fall within a known list of DNS names associated with known Peer-to-Peer services. As de-centralised botnets often use Distributed Hash Tables as a means of communication between peer botnet nodes, a device infected with peer-to-peer botnet malware may attempt to look up a distributed hash table using a common BitTorrent service. For example, a botnet may query the domain **dht**.bittorrent.com as part of the distributed hash-table lookup. Thus, all DNS queries are compared against a shortlist of BitTorrent domain name wildcards commonly used by Peer-to-Peer botnets:

- `.utorrent.com`
- `.transmissionbt.com`
- `.bittorrent.com`
- `.debian.org`

If any of the above domain patterns are matched, the system assumes that the incoming traffic is likely associated with a Peer-to-Peer botnet; hence, the heuristic *i)* is satisfied. However, as this heuristic only indicates Peer-to-Peer network activity, it cannot be used exclusively for Peer-to-Peer botnet identification. Instead, it must be used in conjunction with heuristic *ii)* to infer if the traffic is that of botnet activity.

Heuristic (ii) A data transaction may occur during network analysis of a malware binary when the infected host sends a series of bytes to an external IP address and receives a stream of bytes in response. This activity may include hand-shakes or communication between an infected host and C2 server, downloading a binary from a remote Payload Server, or interactions between peer-to-peer botnet nodes. Thus, this heuristic allows for the distinction between benign and potentially malicious network traffic. IP addresses that actively send and receive information to/from an infected host (i.e. endpoints which participate in a data transaction) will satisfy this heuristic. If both heuristics *i)* and *ii)* are satisfied, it is concluded that the corresponding IP address

ii) may be a potential peer-to-peer botnet node. However, if only heuristic *i)* is satisfied, it is inferred that the IP address may be a potential (candidate) Command & Control server. Notably, this classification will consider any data transactions between an infected host and a Payload server as possible C2 activity, as the botnet owner may also use the Payload Server for Command & Control.

Heuristic (iii) As it is common for centralised botnet malware to include hard-coded IP addresses and domains used for C2 communication, the heuristic *iii)* infers that any IP address logged during network analysis that is contained within a list of hard-coded IPv4 addresses from the binary's source may be a potential Command & Control server. All endpoints are checked against a list of hard-coded IPv4 strings extracted from the previous stage of the LiSa analysis process. If any match occurs, then the heuristic *iii)* is satisfied, and the IP address will be classified as a candidate Command & Control server.

Heuristic (iv) If any IP address connections are logged during the network analysis as known blacklisted Command & Control servers, the heuristic *iv)* is satisfied. The VisiBot Processing system checks the blacklist status of each IP address endpoint encountered during malware analysis by performing lookups across various IP blacklist services, including those provided by Barracuda Central, abuse.ch, SpamRats, Sorbs DNSBL, and Spamhaus. [54, 55, 56, 57, 58] If any IP address has previously been blacklisted as a C2, the IP address is classified as a candidate Command & Control server.

Following heuristic analysis, each IP address is classified either as a candidate C2, a Peer-to-Peer Node, or benign traffic (which is ignored). Candidate C2 or P2P entries are added to the VisiBot database if any of the heuristic requirements are satisfied. A variety of additional IP address metadata is also collected, as previously performed in the packet analysis stage. New botnet event logs are also stored, and IP address connection records are created/updated between the associated payload IP and all C2/P2P IPs encountered during analysis. Lastly, the LiSa analysis report is refined to include only essential information and is stored within the database along with any significant strings extracted during the analysis process. Once this stage is complete, the worker will continue to process other tasks by consuming a new analysis task from the VisiBot broker queue.

4.3 Interactive Web Visualisation

A map-based web application is created to provide an interactive visualisation of all botnet traffic detected by the VisiBot Processing System within a 24-hour window. The web application was designed and implemented following the Model View Controller (MVC) design pattern, as the development of the front-end presentation layer and back-end data interaction layer are independent, allowing for a separation of concerns. The model layer used is the same MongoDB data store [30] utilised by the VisiBot Processing System. Users can interact with various aspects of the model through a front-end visualisation layer written in the Nuxt.js [59] JavaScript framework, which communicates with a back-end web API created using Express.js. [60] Express is a flexible web application framework written in Node.js, [61] which used as a controller interface between the database and front-end application:

The back-end Exprss.js REST API communicates with the VisiBot MongoDB database using Mongoose, an Object Data Modelling (ODM) library for MongoDB. [62] Upon a client sending an HTTP request to the API, the Express.js server will query the MongoDB database directly using Mongoose and returns documents to the client in the standard JSON format. As Express, Mongoose, and Nuxt.js are JavaScript-based, the JSON-based document structure used to represent MongoDB documents is natively supported across all three frameworks, allowing for extensive flexibility without introducing overhead such as JSON serialisation. In conjunction with Nuxt.js and Express.js, several `npm` packages were also used throughout development. The

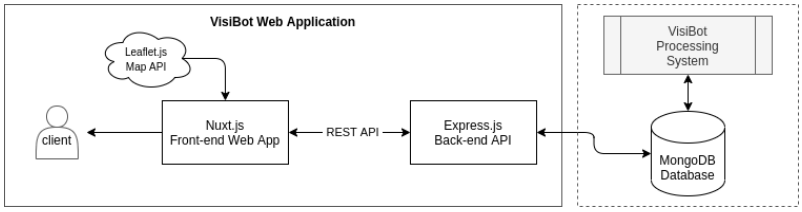


Figure 4.4: *Server Architecture Diagram: VisiBot Web Application*

BootstrapVue [63] library was used with Nuxt.js to develop a modern, flexible, and intuitive user interface using the Bootstrap CSS framework.

4.3.1 Geographic Clustering and Networks

An interactive, cluster-based map was developed using [64] to allow for real-time interaction with the various botnet entities identified by the VisiBot Processing System. This map displays a clustered representation of the geographic locations of all entities identified within the last 24 hours, with each IP address displayed as a coloured marker. The colour of the marker depends on the classification of traffic coming from the IP address. As the user zooms in on the map, the clusters dynamically become smaller in size, allowing for the user to interact with the full extent of the map without experiencing information overload or the performance side-effects of displaying too many markers at once. An example of the VisiBot Web Application interface is shown below:

Figure 4.5: *VisiBot Web Application*

When a cluster is selected, the map will automatically zoom in and disseminate the clustered markers into smaller clusters or individual markers, depending on the cluster density. When a marker is selected, the user is presented with three options. The first option allows the user to view a network of all of the associated connections that the current IP address has with other botnet entities identified by VisiBot. For example, a candidate C2 server's connections may include interactions with payload servers, report servers, or infected bots. These connections are generated using the `graphLookup` document aggregation [65] provided by MongoDB, allowing for full traversal of the connection tree of a given IP address. All coordinates collected using `graphLookup` for a given IP address are drawn as lines on the map, as demonstrated in the Peer-to-Peer connections example shown in appendix A.7.

4.4 Software Development Practices

Issue Tracking Trello, a Kanban-style web-based ticket tracking application, was used throughout development to manage the various deadlines, priorities, tasks, and to-do lists. As development tasks in Trello are represented as tickets, I managed my progress by sorting each task based on development status and priority. All tickets were actively moved between 'Backlog', 'In Progress', and 'Complete' columns, allowing me to keep track of the tasks I have yet to complete. Using a board-based ticket management system ensured I could manage my work without being overwhelmed by long to-do lists, as all tasks, priorities, and features are displayed in a highly coherent and user-friendly interface. More so, Trello also proved highly beneficial for tracking and managing various project-related deadlines and nested to-do lists, as each ticket can be assigned a due-date timestamp and is capable of encapsulating several sub-lists of tasks which can be updated using interactive check-boxes.

Quality Assurance Throughout development, linting tools and unit testing tools were used as a means of quality assurance. Several unit tests were written for the Bad Packets API using the unittest [66] python testing framework, ensuring reliable and robust code execution throughout the deployment of the project. Pylama, an auditing/linting tool for Python, [67] was frequently used to ensure standardised coding style and readability. Pylama wraps various linting tools into a single code auditing tool, including popular linters such as Pylint, PyFlakes, and pycodestyle. The combination of various linting tools enables strict enforcement of various practices followed by Python programming professionals, including official style guides, such as the PEP-8 style guide written by van Rossum et al. [68]. Additionally, ESLint, [69] a static analysis linting tool for JavaScript, was also used for actively identifying problems in both Nuxt.js and Express.js web applications, including factors such as consistency, readability, and correctness. Both Python and JavaScript linting tools proved excellent for identifying an abundance of errors and inconsistencies amongst the VisiBot code-base, minimising potential issues that may have otherwise surfaced from inconsistent or poorly written code.

Source Code Management All source-code managed using Git [70] version control and was also remotely hosted on a private repository using the GitHub [71] code-hosting and collaboration platform. Throughout the development of VisiBot, I strictly followed a trunk-based branching strategy such that all SCM changes were implemented as small, frequent commits to the master/main branch of the repository. As I was the sole developer throughout the implementation of VisiBot, branching strategies such as feature branching were considered unnecessary, as the likelihood of encountering merge conflicts within a single-developer work environment is highly unlikely. Before pushing any local changes to the remote repository, all code is automatically tested locally using Git pre-push hooks. [72] By hooking a shell script onto the `git push` command, the contents of the script is executed and evaluated before the push request is processed. If the script exits with an error, the user must fix the errors raised during the pre-push script execution before allowing for changes to be pushed to a remote branch. Within the context of VisiBot, code linting and unit testing stages were automatically executed locally before pushing to a remote GitLab repository using a pre-push script. This script executes various unit tests, and the `pylama` and `eslint` linting tools within `src/processing` and `src/webapp` directories consecutively. By doing so, quality assurance is maintained by ensuring that if either of the linting tools or unit tests fail, the push request will be aborted until all issues are resolved. All unit testing and linting tools are also automatically performed through a Continuous Integration (CI) Pipeline deployed using the Travis CI [73] Continuous Integration service. Integratable with GitHub, Travis CI is used throughout development to automate the linting and unit-testing quality assurance measurements whenever code is pushed to a monitored GitHub branch, such as the master branch. If the pipeline fails, all branch maintainers are alerted of the failure and are responsible for fixing any issues. The automatic testing of code ensures that even if a developer pushes code without testing locally, all code is tested before merging into the master branch.

5 Results and Evaluation

5.1 Data Collection

The VisiBot Processing System was deployed over a five-week (35-day) data collection period between 11/02/2021 and 11/03/2021, resulting in the processing and analysis of 61,293 honeypot packets collected from various strategically located honeypot servers hosted by Bad Packets. As the deployed honeypots mimic that of IoT device characteristics, the proposed system was able to identify multiple IoT botnets, including several prominent Peer-to-Peer botnets. Throughout the deployment period, a 3-worker LiSa Sandbox server was hosted on a separate network from the VisiBot Processing System, ensuring a separation of concerns. All objects generated during malware analysis, such as malware samples, program output logs, and packet capture (pcap) files, were actively stored on a secure block-storage partition. The block-storage partition ensures that if the sandbox were to become compromised, the server could be safely reset without presenting a loss of valuable information. Throughout the collection process, the LiSa sandbox operated using three celery workers and executed all binaries for a total of 30 seconds. Additionally, the VisiBot was configured to use six celery workers to allow for parallelised processing of honeypot data and LiSa analysis reports.

Following the analysis of 61,293 honeypot packets, VisiBot lead to the identification of 58,010 unique public IP addresses, including 1,303 candidate Command & Control servers and 6,876 P2P Nodes. Additionally, the system logged 4,000 Autonomous Systems, 82,050 botnet events, and extracted 9,923 malware payload URLs. Out of the payload URLs extracted, a total of 1,654 malware samples were successfully retrieved, consisting of a total of 150 unique MD5 hashes. Following malware analysis of all extracted samples via the VirusTotal aggregation service, a total of 62,203 out of 97,659 anti-virus vendor scans were positive. Throughout the data collection period, all significant honeypot and malware traffic was classified into a total of six categories using a combination of packet analysis and heuristic analysis techniques. The categorised distribution of all traffic processed by VisiBot is highlighted below:

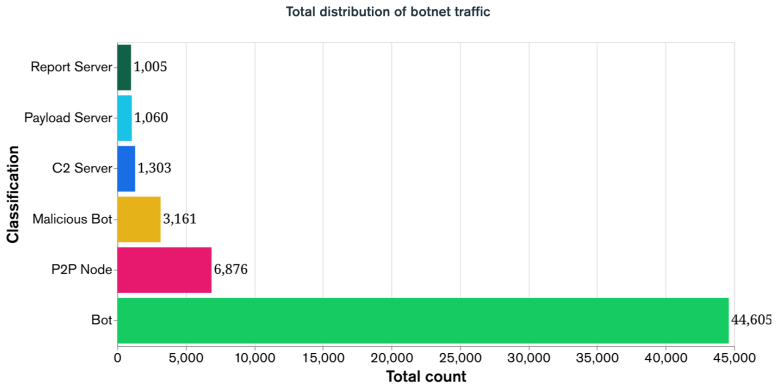


Figure 5.1: *distribution of classified botnet traffic.*

Despite a significant proportion of detected botnet traffic being that of benign Mirai-like port-scanning activity, a large proportion of traffic has been observed to originate from several Peer-to-Peer networks and malicious (self-propagating) bots. Notably, more malicious bots have been observed than Report Servers and Payload Servers, which are commonly observed across centralised Mirai botnets. This may allude to a potential change in contemporary botnet propagation strategies, as bots are now being utilised to self-host malware and actively perform Remote Code Execution attacks. However, as this type of activity is most commonly associated with Peer-to-Peer IoT botnets like Hajime, the above graph may reflect a recent transition from previously accepted centralised botnets to highly dominant de-centralised Peer-to-Peer botnets.

5.2 Geographic Botnet Distribution

Throughout the five-week automated processing period, Visibot collected a wide array of IP address metadata, including geographic and autonomous system information. Such data includes the city, country, and continent of all IP addresses encountered, as well as geographic coordinates sourced from a frequently updated MaxMind GeoIP2 database. The broker-based detection system successfully processed and classified a multitude of honeypot packets, encountering activity spanning several continents, including Europe, North America, South America, and Asia. The geographic metadata of identified botnet entities can be analysed to better understand how IoT botnets are currently propagating. Additionally, geographic analysis allows for the visualisation of botnet growth and geographic IoT botnet hot-spots. The below heat-map visualises geographic botnet density through the plotting of latitude and longitude coordinates collected during analysis. Areas of high botnet traffic density are represented using a yellow-red colour gradient, whereas diminished areas are represented as opaque green-blue gradients:

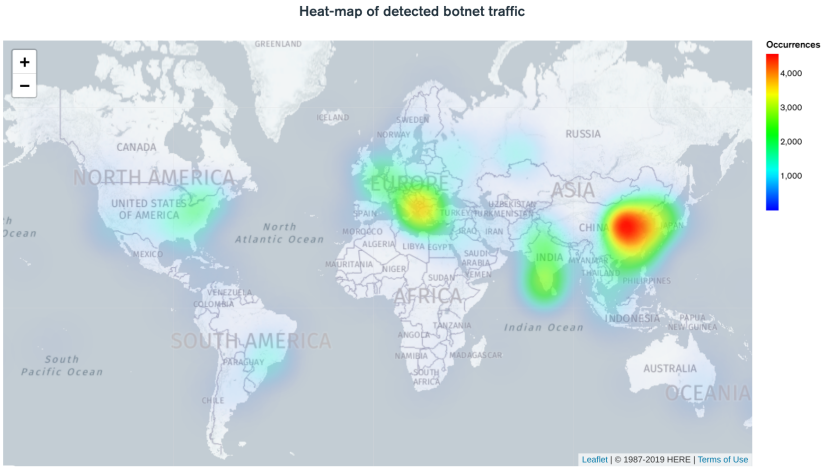
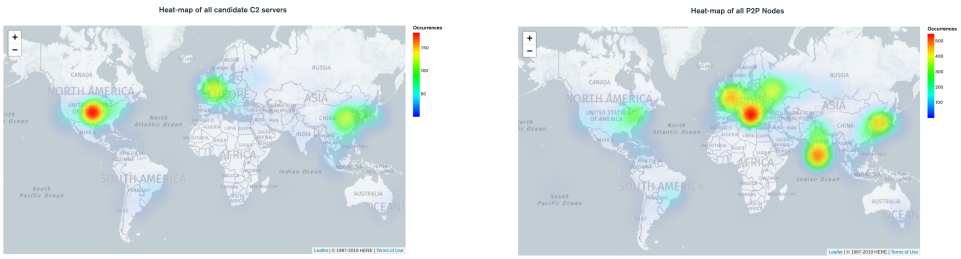


Figure 5.2: *Geographic distribution of all botnet traffic detected by the VisiBot Processing System.*

The above geographic analysis indicates significant traffic density originating from various regions. However, the density of botnet traffic originates from Asia, where China is highlighted as the largest region. As inferred by the traffic distribution histogram previously shown in Figure 5.1, a vast proportion of all traffic originating from China is that of Mirai-like port-scanning activity. Given that China is one of the leading countries for manufacturing IoT devices, and many of which have security vulnerabilities, it is likely that the quantity of IoT devices hosted in china has resulted in the country becoming an IoT botnet hot-spot. Similar but less significant geographic density patterns are also observed across India, Europe, North America, and Brazil.

However, when comparing the geographic distribution of identified Command & Control candidates and Peer-to-Peer nodes, several differences can be observed when compared to the overall botnet density map. Consider the below side-by-side comparison of the geographic density of all candidate C2 servers and P2P nodes identified by VisiBot:



(a) Candidate C2 Server Geographic Distribution *(b) P2P Node Geographic Distribution*

Figure 5.3: *Malware sample architecture and endianness represented side by side.*

As shown, the heat-map visualisation of candidate Command & Control server distribution shows very high botnet activity across the United States of America. Medium C2 density is also observed across Europe and China. There is a clear contrast in traffic distribution when comparing geographic C2 density against all botnet traffic. The high level of C2 activity originating from North America indicates that most globally distributed botnets are being controlled by servers or computer systems located in the United States, with some occurrences logged in Central Europe and China. Adversely, most identified Peer-to-Peer botnet traffic is highly prevalent amongst Central Europe and India, with a wider global distribution than Candidate C2 Servers. As Peer-to-Peer botnets operate through communicating with peer nodes, which are often distributed across several countries and continents, a wide geographical distribution can be observed. In contrast, C2 servers often only operate out of specific regions, which are in some cases local to the botmaster. The distinction between the geographic distribution of overall, C2, and P2P traffic is further highlighted when inspecting the distribution of associated Autonomous System registries:

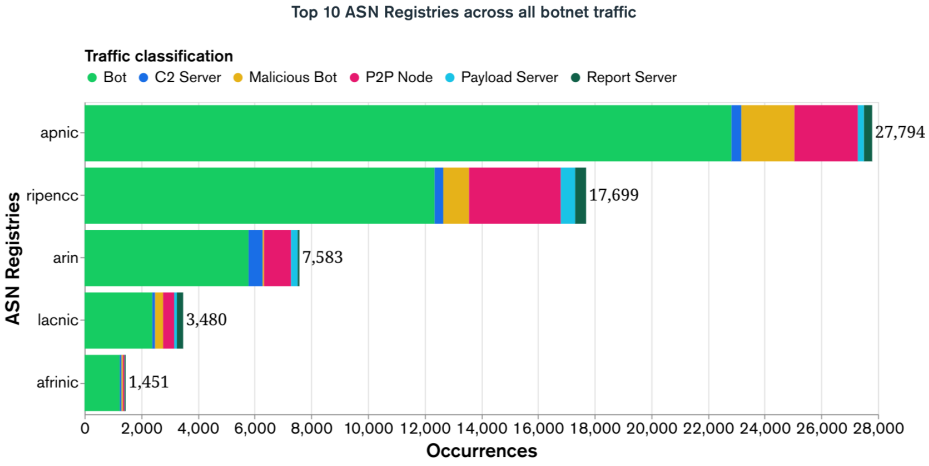


Figure 5.4: *distribution of classified botnet traffic across ASN registries.*

Both the APNIC and RipeNCC ASN registries are widely associated with Mirai-like scanning, malicious botnet, and Peer-to-Peer activity. The bulk of identified port-scanning, malicious, and Peer-to-Peer botnet activity originates from IP addresses provided by Autonomous Systems

operating within Asia-Pacific and Central/Eastern Europan regions. Adversely, the ARIN registry is more frequently associated with candidate C2 traffic and Peer-to-Peer traffic, with little occurrence of a malicious bot, report server or payload server activity seen across the United States, Canada, and North Atlantic. These findings are further validated by measuring the distribution of Autonomous Systems based on their country of origin:

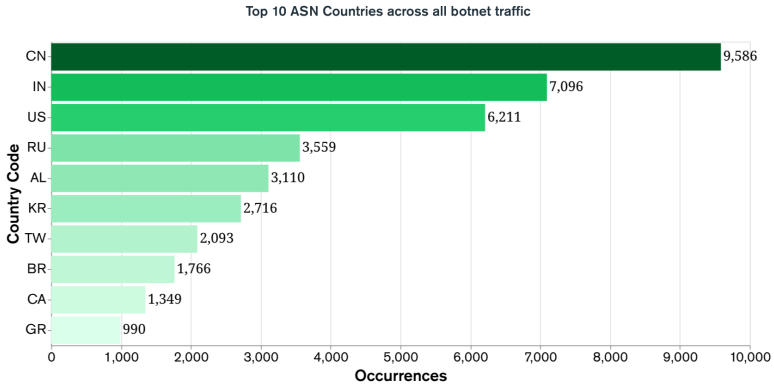
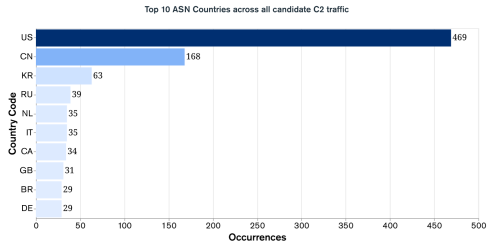
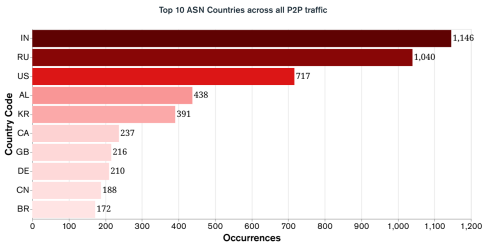


Figure 5.5: *Top 10 ASN country codes for overall botnet traffic.*

As inferred in Figure 5.2, the above graph confirms that the leading country of origin for overall botnet traffic is indeed China, followed by India, the United States, and Russia. Using this information, we can deduce that IoT device traffic originating from China is primarily that of port-scanning and malicious bot activity. However, when inspecting the geographic distribution of ASNs associated with C2 or P2P botnet traffic, the origin of such traffic differs significantly from the geographic distribution of overall traffic:



(a) *Top 10 Candidate C2 Server ASN Country Codes*



(b) *Top 10 P2P Node ASN Country Codes*

Figure 5.6: *A side-by-side comparison of the geographic distribution of Autonomous Systems associated with P2P and C2 botnet activity.*

The above comparison illustrates a clear contrast between the distribution of C2, P2P, and overall traffic across the geographic regions of associated Autonomous Systems. The distribution of C2 ASN countries of origin confirms that most detected candidate C2 traffic originates from the United States. Contrary to that of C2 ASN distribution, logged Peer-to-Peer traffic predominantly originates from Autonomous Systems located in India, Russia, the United States, Albania, and South Korea, affirming that P2P botnet networks are more widely dispersed than that of C2 and propagation activity. Notably, most of the overall botnet traffic originates from IP addresses associated with AS4837 (China Unicom Backbone) and AS9829 (Bharat Sinchar Nigam Ltd.), located in China and India, respectively. The VisiBot Processing system also detected a significant amount of C2 botnet traffic originates from North American providers AS16625 (Akamai Technologies, Inc.) and AS16509 (Amazon.com). Diversely, the bulk of detected P2P

traffic is more widely distributed across several ASNs, including AS8661 (Telekomi i Kosoves SH.A), AS9829 (Bharat Sinchar Nigam Ltd.), AS4766 (Korea Telecom), AS17465 (Asianet Satelite Communications), and AS12389 (Rostelecom). For the top 10 Autonomous Systems Numbers across C2, P2P, and overall botnet traffic, see Appendix A.10.

5.3 Traffic Analysis

Despite a large proportion of benign botnet activity shown in Figure 5.1, such as port-scanning activity, the second most commonly occurring traffic type is that of Peer-to-Peer Botnet traffic. As P2P nodes are identified during the malware analysis stage of the VisiBot Processing System, this indicates current popularity in the use of de-centralised Peer-to-Peer based infrastructures, like that of IoT botnets such as Hajime. Additionally, the number of packets classified as malicious botnet activity is significantly higher than that of identified payload and report server activity. The concept of a self-propagating malicious bot is a relatively new phenomenon amongst IoT botnet traffic, as such bots actively attempt to infect vulnerable devices using self-hosted malware. In comparison, conventional Mirai bots often report the detection of vulnerable devices to a report server, which performs remote code execution using malware hosted on a payload server. As self-hosted malware mitigates payload servers' requirement and allows for de-centralised propagation, this may explain why recently observed botnets utilise malicious bots more often than report or payload servers.

In order to gain a better understanding the IoT devices which are currently being attacked, the VisiBot Processing System logs the detected packet categories and Common Vulnerability Exploits (CVEs) included within the packet information provided by the Bad Packets honeypot API. [10] The below table represents the top 10 most commonly observed botnet packet categories and CVEs across all traffic classified by the VisiBot Processing System:

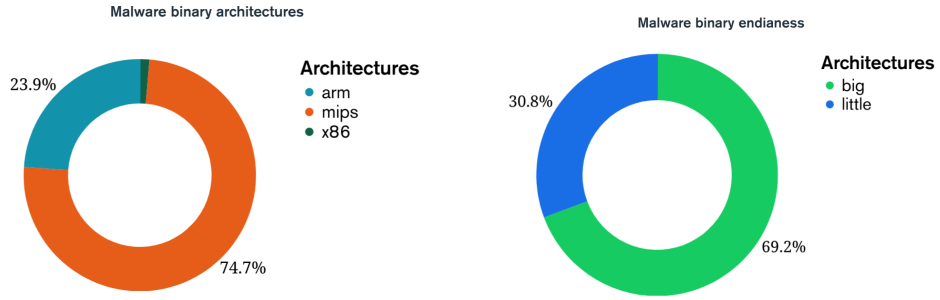
Table 5.1: *Top 10 CVEs and packet categories across all botnet traffic*

CVE	Category	Description	Count
-	Botnet Activity	Mirai-like Scan	43,552
-	IoT	MVPower DVR (JAWS) RCE	5,050
CVE-2016-10372	Router	Eir D1000 RCE	1,878
-	Router	Netgear RCE	632
CVE-2018-10561	Router	GPON RCE	593
CVE-2016-6277	Router	Netgear RCE	310
-	IoT	Vacron NVR RCE	298
-	IoT	Generic CCTV-DVR RCE	289
CVE-2013-7471	Router	D-Link UPnP SOAP RCE	73
CVE-2013-3568	Router	Linksys RCE	67

Botnet coordinators appear to be explicitly targeting IoT devices with known vulnerabilities through various attempts of Remote Code Execution (RCE). Malicious bots and/or report servers target devices such as routers, IP cams, and DVRs, all of which may be vulnerable to login credential brute-forcing attacks. Given the continued use of exploits known for a considerable amount of time, this indicates that there is still a significant amount of unpatched devices that attackers can infiltrate. Upon manual analysis of several attack packets collected by the honeypot, I observed that several infected devices were attempting to perform remote code execution using bash scripts hosted on remote payload servers. These bash scripts are used to ensure that the correct malware binary is executed on an infiltrated device, such that MIPS IoT devices are infected with a MIPS-compile binary and so on.

5.4 Malware Analysis

The previous observation of potential target CPU architectures is reaffirmed by the findings obtained through static analysis of several malware binaries collected during the data collection period of the VisiBot Processing System. As shown in figure 5.7, static analysis of the extracted binaries collected by the VisiBot Processing System indicates that ARM and MIPS CPU architectures are currently being targeted at a high rate, with only 1.3% of binaries targeting x86 devices. Similarly, botnets are mostly attacking IoT devices with big-endian memory addresses. This combined analysis indicates that the vast majority of detected traffic originates from IoT botnets, which explicitly target a wide range of devices distributed across two microprocessor architectures.



(a) Malware Architecture *(b) Malware Endianess*

Figure 5.7: *Malware sample architecture and endianess represented side by side.*

Given these findings, it is likely that the types of botnets identified by the VisiBot are likely Mirai or Bashlite variants, as they are exclusively targeting IoT devices running on microprocessor technology. The distribution of identified botnet variants can be calculated by scanning each binary using VirusTotal and extracting the top-most occurring keyword across the positive vendor scan results. When a malware vendor positively identifies a binary as malicious, the malware's name is often characterised as a string, such as `Type:Platform/MalwareFamily`. As the result strings for positive vendor scans are given through the VirusTotal API, information such as the malware family can be parsed and sorted based on occurrence, such that keyword identifiers are extracted. The below table shows the most commonly occurring VirusTotal keywords across all analysed malware binaries:

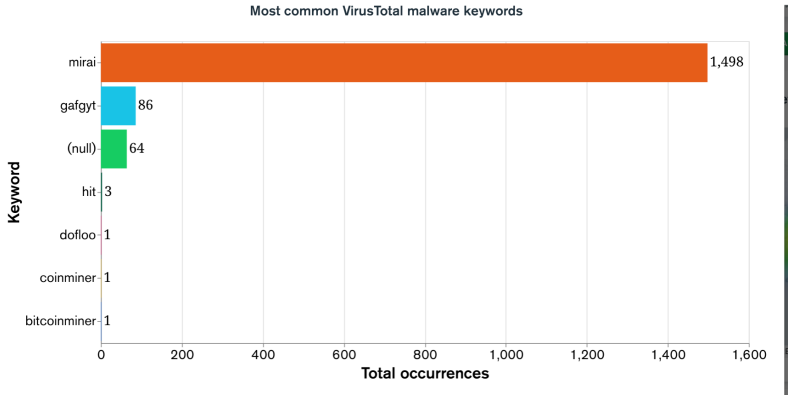


Figure 5.8: *distribution of malware samples based on VirusTotal keywords*

Although the vast majority of analysed binaries were detected as Mirai malware variants by the VirusTotal anti-virus vendors, some were identified as Bashlite (gafgyt) and coin-miners. The "hit" keyword in the above table is derived from the anti-virus result string `Trojan.Linux.Agent.HIT`. Despite testing positive, most VirusTotal anti-virus vendors could not detect the malware family of these binaries. Binaries that were not detected by the VirusTotal vendor scanning service were allocated the `(null)` placeholder keyword.

Network Analysis During the network analysis stage of LiSa sandbox analysis, several network interactions originating from the infected sandbox host were logged, including all HTTP requests, DNS look-ups, and IRC Traffic. The logged HTTP requests were very typical to that of Mirai and Bashlite botnets, as several attempts were made by the infected host to brute-force and perform remote code execution on identified vulnerable devices. However, a significantly low number of DNS queries were observed throughout analysis of all 1,654 binaries, as a total of only 10 unique DNS look-ups were performed across all infected hosts:

Table 5.2: *Table of all DNS Queries logged during malware analysis*

Domain	Record Type	Category	Total
dht.transmissionbt.com	A	P2P DHT	637
router.bittorrent.com	A	P2P DHT	463
router.utorrent.com	A	P2P DHT	452
bttracker.debian.org	A	P2P DHT	441
jebiga.babyhub.pw	A	C2 Server	6
hjar64zj6ygjd.com	A	C2 Server	2
cnc.fewbots.cc	A	C2 Server	1
g*y.energy	A	C2 Server	1
g*y.energy	AAAA	C2 Server	1
ipinfo.io	A	Benign	1

Notably, only four potential Command & Control server DNS recorders were identified, with most DNS lookups being associated with public Peer-to-Peer services and Distributed Hash Tables. Of the C2 DNS records identified, one domain name shows low vowel density and high Sharron entropy (randomness), common to that of conventional Mirai C2 domain names. Dwyer et al. [19] However, as shown, DNS based botnet detection would not have proved beneficial in this case as a small number of DNS lookups were observed. The querying of public P2P service domains is a relatively innovative characteristic of de-centralised IoT botnets, such as Hajime or Mozi. Once a bot is infected, it will attempt to access the botnet DHT through public BitTorrent services, such as those logged above. Such DNS queries were previously observed in Mozi botnet network activity by [74]. A high occurrence of Mozi malware amongst collected binaries can be observed in the word cloud generated in Appendix A.9, validating that a large proportion of the botnet traffic detected by the VisiBot Processing System originates from the Mozi botnet. The filenames `Mozi.a` and `Mozi.m` also signify devices running on **a**rch and **m**ips CPU architectures are being directly targeted by Mozi.

Malware Port Usage The UDP and TCP network statistics are shown in Appendix A.8 also reaffirms that the majority of detected botnet traffic is specific to the targeting of IoT, as devices are primarily targeted via Telnet (22/2323) and HTTP (80) via the TCP protocol. As many IoT devices have vulnerabilities in online dashboards, attackers often attempt credential brute-forcing and remote code execution (RCE) exploits on IoT dashboards/panels accessible through port 80. Other highly targeted TCP ports, such as 5555, 49152, and 37215, are ports commonly used by IoT devices such as IP cameras and routers. Highly targeted ports, such as 7574, are associated with cluster services such as the Oracle Coherence Cluster service, indicating that attackers may be actively trying to infiltrate cluster systems using known exploits. Notably, ports 443 and 22, commonly used with HTTPS and SSH, are very low on the list of targeted TCP ports. This

suggests that attackers are less concerned with the brute-forcing of SSH clients and routing of traffic through secure network layers such as HTTPS.

In contrast to TCP, the UDP ports 67, 1900, 53, and 6881 were also largely targeted by bots observed throughout malware sandbox analysis. The UDP port 53 is often listened to by worms and botnets awaiting incoming commands from the C2 servers. [75] Whereas, ports 67 and 1900 are typically used by DHCP servers and UPnP devices, as highlighted in Table 5.1. However, the vast majority of logged UDP traffic (shown in Figure A.6) is widely and randomly distributed across various ports. These unusual characteristics may be an indication of malicious bot activity associated with Mozi botnets. Mozi bots propagate the botnet through self-hosting malware on local webservers hosted at randomly selected UDP ports. The files hosted on the temporary webserver are downloaded when the malicious bot successfully brute-forces or performs remote code execution on a vulnerable IoT device. After some time, the port of the webserver changes to a different randomly selected port. The presence of such activity may explain why a high number of malware payload URLs were extracted from detected honeypot packets. Still, only a small percentage of binaries were retrievable. It is possible that the VisiBot Processing System wasn't quick enough in retrieving malware binaries from certain URLs. This issue may be caused by the limitation of having to query the honeypot service used by VisiBot on an hourly basis. Should packets be processed immediately upon identification, it is likely that the URL pointing to a randomly selected port will still be active, allowing for a higher number of samples to be collected.

P2P Payload Connections Analysis Generated using NetworkX [76] and the K-Components approximation algorithm [77], the below edge-connectivity graph visualises the various relationships between analysed malware binaries and identified Peer-to-Peer networks using the IP address connection relationship information logged throughout network analysis of analysed malware binaries:

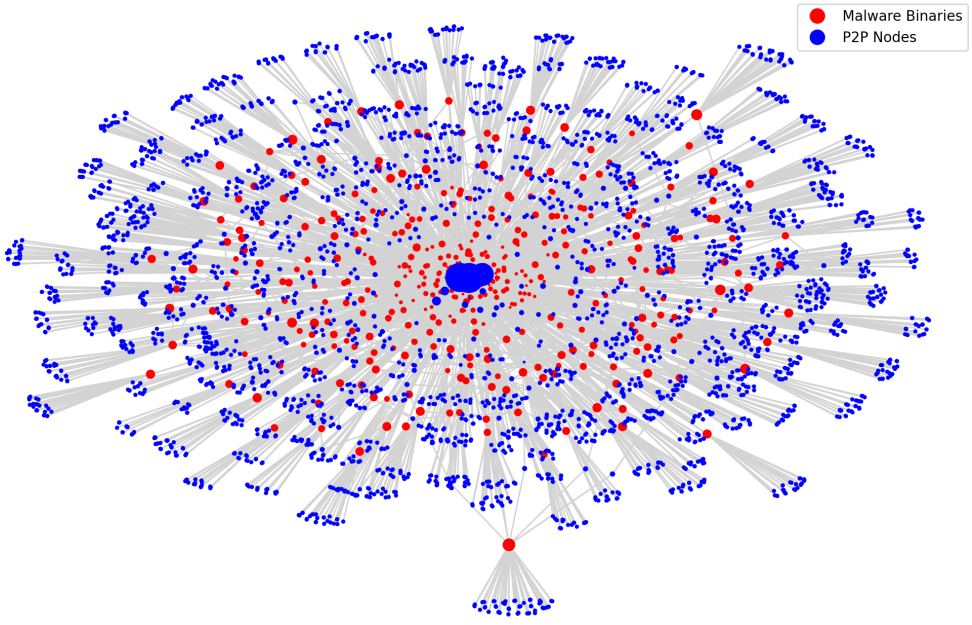


Figure 5.9: *Edge-connectivity graph of P2P Node and malware binary interactions recorded during network analysis. Edge connections approximated using K-Components algorithm.*

Several small-scale Peer-to-Peer botnets are shown throughout the perimeter of the graph, with several P2P networks being interlinked through some mediary P2P nodes. As shown, various of

the small-scale P2P networks detected during analysis are interlinked with each other, and with larger P2P networks spanning across the centre of the graph. The blue mass of Peer-to-Peer nodes at the centre of the visualisation indicates botnet propagation growth of one or more large P2P botnets over the 5-week data collection period, as several connections between P2P nodes have been established across multiple payload binaries which are likely to originate from the same botnet. However, as public distributed hash tables are often used in Peer-to-Peer botnets, it is likely that a high proportion of this traffic is legitimate, as bots often interact with legitimate peers to throw-off potential network analysis. [20] A summary of the number of nodes associated with malware payloads is shown below:

Table 5.3: *Summary: Number of P2P Nodes identified during malware binary analysis.*

Min	1st Quartile	Median	Mean	3rd Quartile	Max
1.0	5.0	12.0	12.75	18.0	69.0

Candidate C2 Server ASN History Analysis The relationships between identified candidate C2 servers and Autonomous Systems are also visualised using the same K-Components approximation algorithm used above. Using the recent ASN history of all candidate C2 Servers, sourced from various WHOIS records via ipwhois [78], the below network graph illustrates the historical interactions, such as ownership or temporary holdings, between Autonomous Systems and identified candidate C2 servers:

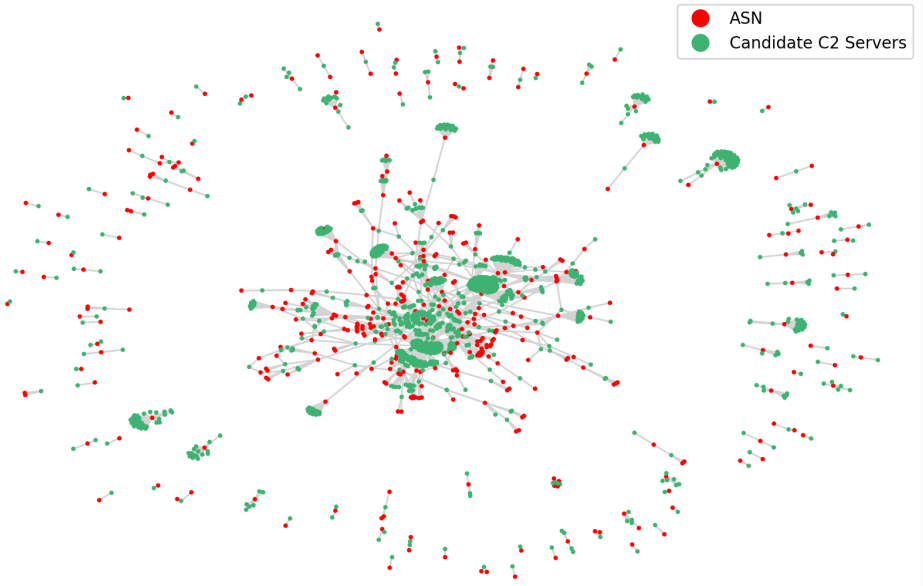


Figure 5.10: *Edge-connectivity graph of P2P Node and malware binary interactions recorded during network analysis. Edge connections approximated using K-Components algorithm.*

Table 5.4: *Summary: Number of ASNs associated with 1,157 candidate C2 Servers*

Min	1st Quartile	Median	Mean	3rd Quartile	Max
1.0	3.0	5.0	5.437	9.0	10.0

As shown above, the central network cluster indicates that several ASNs are holding/exchanging ownership of IP addresses associated with Command & Control activity detected by VisiBot. Additionally, it is observed that a high number of C2 IP addresses are owned by a select few

Autonomous Systems, with a mean average of 5 ASNs associated with identified candidate Command & Control servers. An iterative graph-based monitoring tool can be built using this information to observe the interactions between C2 Servers and ASNs. Such observations may allow for insight into the trading/holding process of IP addresses associated with botnet activity amongst Autonomous Systems.

5.5 Evaluation

Over a 35-day experimentation period, the distributed honeypot system, automated malware extraction, and botnet classification processes employed by VisiBot enabled the detection of over 58,010 unique IP addresses and the collection of 1,654 malware samples comprised of 150 unique MD5 hashes over 35 days. In comparison, the IoTPoT honeypot system proposed by Pa Pa et al. [14] identified a total of 16,934 visiting IP addresses and collected 43 unique malware samples over a 39-day evaluation period. The significant difference in the number of unique IP addresses encountered highlights the advantages of using a strategically distributed honeypot network. The extensive, worldwide, and intelligent honeypot services provided by Bad Packets LLC [10] permitted the processing of thousands of honeypot packets per day, enabling the significantly higher collection of malware samples over a shorter evaluation period.

Given the significant number of malware URLs extracted during the experimentation period, the payload URL extraction employed by VisiBot has proven highly effective. The combination of regular expression pattern matching and manual re-building of URLs obfuscated using command-line arguments allows for the efficient and computationally inexpensive identification of sample URLs used in botnet remote code execution attacks. However, a considerable proportion of the extracted malware URLs were no longer active upon reaching the malware analysis stage. Out of the 9,923 URLs identified, only 1654 malware samples were collected due to limitations within the honeypot collection process.

During the 35-day experimentation period, VisiBot encountered a significant number of malicious bots originating from the Mozi botnet variant, which were observed to be performing remote code execution attacks using self-hosted malware samples. However, as the corresponding port for hosting malware is frequently changed, a small window of time is given to download the binary before the URL expires. As the VisiBot Processing System retrieves honeypot data on an hourly basis from an external honeypot service, the hour-long delay between intrusion detection and analysis often results in such malware URLs expiring before binaries can be downloaded and analysed. A possible solution is to use a real-time stream or socket-based honeypot API, allowing packets to be analysed within a short time frame of being detected by a given honeypot. By doing so, a more significant proportion of self-hosted payload URLs can be collected.

Implementing a scalable and distributed sandbox analysis system allows for automatic static, dynamic and network analysis of malware samples extracted from the honeypots utilised by VisiBot. In doing so, the VisiBot Processing System was able to identify significant botnet traffic following the heuristic analysis of 1318 out of 1654 malware binaries, yielding an identification rate of 79.87%. Opposed to these results, the botnet detection techniques and heuristics employed by Bastos et al. [16] permitted the C2 server detection of 1010 out of 1050 malware samples, resulting in a detection rate of 96% across all samples. However, both systems are comparably quite different. The method proposed by Bastos et al. [16] is exclusively concerned with the identification of Centralised Command & Control servers from the conventional IoT botnets, Mirai and Bashlite. It does not consider approaches for identifying de-centralised botnets. Whereas the VisiBot Processing System is primarily concerned with identifying both centralised and de-centralised IoT Botnets, allowing for the detection of C2 servers and Peer-to-Peer botnet traffic generated by relatively new botnet variants such as Mozi and Hajime.

Nevertheless, one significant drawback of the VisiBot Processing System, in its current state, is

that there is no validation process in place for confirming the validity of identified candidate C2 and P2P traffic through the likes of executing hand-shaking procedures. Many de-centralised botnets communicate with legitimate public Peer-to-Peer networks, and centralised botnets are known to communicate with fake C2 servers to mislead authorities. Thus, the legitimacy of the C2 or P2P candidates identified by VisiBot should be assumed, as such entities must first be validated through the invocation of several botnet handshake procedures. Through performing handshakes with candidate C2 servers, the legitimacy of each C2 can be inferred based on the handshake response from its server. However, this functionality is not yet present within the VisiBot Processing System but can be extended in future iterations to include such validation procedures.

A considerable number of candidate C2 servers and P2P Nodes were detected through the VisiBot Processing System, indicating that the chosen heuristics effectively differentiate between benign and malicious botnet traffic. However, the lower detection rate indicates when compared to other systems suggests that some of the heuristics used were less effective than anticipated. A high number of packed malware binaries were observed during the data collection period. Botnet owners were primarily using the UPX packing tool [52] to compress binaries and obfuscate source code. Despite making modifications to the LiSa sandboxing server to enable the unpacking of UPX binaries, many samples could not be successfully unpacked. This resulted in the collection of obfuscated binary strings during static analysis, which proved too noisy for extracting hard-coded IP addresses. Following this, the heuristic-based detection of connections to hard-coded IP addresses proved the least effective across all four heuristics. However, with further improvements to binary un-packing procedures, this heuristic will likely prove more advantageous for identifying candidate Command & Control servers.

Throughout the 35-day data collection period, the VisiBot Processing System has proven highly reliable and robust, having automated extensive analysis procedures for over 61,000 honeypot events. The logging of botnet entity interactions, traffic classifications, sample analysis information, geographic data, and autonomous system history allows for many insights to be made regarding the current state and potential evolutionary patterns of IoT Botnets. The collection and analysis of malware binaries has proven effective in identifying the source of control amongst centralised and de-centralised botnets. However, analysis of target binary CPU architectures and frequently observed Common Vulnerability Exploits also give insight into how IoT botnets are propagating and the types of exploit mechanisms that botnet developers are employing. Graph-based visualisations of C2 and Autonomous System interactions also provide insight into the interplay between Autonomous Systems and botnet IP addresses. The visualisation of Peer-to-Peer connections observed between malware payloads also can be used to measure the rate at which such botnets are propagating. Additionally, visualisations can assist with identifying centralised interactions between P2P node IP addresses. Collectively, the variety of information collected by the VisiBot Processing System is highly significant within Cyber Threat Intelligence.

6 Conclusion

The combined process of automated traffic classification, payload extraction, and sandbox-based malware analysis has proved highly effective for identifying and visualising IoT botnet traffic. VisiBot has shown to be robust, extensible, and capable of scaling demand by employing a globally distributed honeypot network and Message Broker-based analysis system. The distribution of honeypot and malware analysis tasks across multiple workers allows for real-time and parallelised packet classification, payload URL extraction, and malware analysis. The collection of botnet entity IP address meta-data, such as geographic location, Common Vulnerability Exploits, and Autonomous System history, provides sufficient knowledge for visualising and characterising current botnet behaviours and mannerisms, including the ubiquity and density of globally distributed botnets, leading countries of C2 and P2P botnet activity, and ASN trading patterns associated botnet IP addresses.

Classifying over 58,010 unique IP addresses, VisiBot was able to identify, examine, and visualise 1,303 candidate Command & Control servers and 6,876 Peer-to-Peer nodes associated with 1,654 malware samples over a 35-day evaluation period. Unlike previous implementations, the heuristics used within the VisiBot Processing System permit identifying both centralised and de-centralised botnets through automated sample extraction, sandboxing and a combination of analysis techniques. As the project is written in Python and containerised using docker, its various components are highly extensible. The VisiBot analysis stage can be extended to include additional heuristics and detection for new botnet characteristics, communication protocols, and obfuscation techniques. In its current configuration, the simple heuristics used by VisiBot allowed for the identification and analysis of a relatively new and highly dominant Peer-to-Peer botnet known as Mozi. The system also detected several other botnets, including Mirai, Bashlite, and various unknown variants. However, some issues were encountered throughout the data collection and analysis, such as malware sample unpacking, execution of extracted bash scripts, and the collection of malicious bot samples.

6.1 Future Work

As the proposed system has yet to implement a handshake/validation process, VisiBot cannot determine the legitimacy of detected candidate C2 servers and P2P traffic. The heuristic analysis procedure can be expanded in future iterations to accommodate the automated execution of handshake procedures against such candidates. However, as the handshake procedure often varies between botnets, additional static and dynamic analysis heuristics may be required to infer the type of botnet variant before attempting a handshake. As described by Bastos et al. [16], Mirai and Bashlite Command & Control servers can be validated through reverse engineering a malware sample, inspecting its handshaking procedure, and performing the handshake with the candidate. This validation process is equally necessary for Peer-to-Peer botnets, as Mozi and Hajime purposefully attempt to obfuscate Peer-to-Peer botnet activity through interacting with public networks, protocols, and Distributed Hash Tables used by legitimate peers. [74] The DHT handshake technique proposed by Herwig et al. [20] could also be applied to future iterations of the VisiBot Processing System such that detected P2P Nodes are validated through the automatic exchanging of public keys or other handshake signatures.

Despite proving effective for candidate C2 and P2P detection, some identification heuristics were less effective than others throughout the evaluation period. Namely, the heuristics based on hard-coded IP addresses of malware binaries proved challenging to utilise due to limitations caused by the unpacking of collected malware samples. The VisiBot Processing System had difficulties unpacking several malware binaries packed with tools such as UPX [52], leading to the extraction of obfuscated/nonsensical strings during static analysis. By developing a more robust malware unpacking procedure with improved detection and alternative unpacking mechanisms, we can ensure the constant collection of crucial strings, such as IP addresses, which are actively used throughout the heuristic analysis process. By doing so, heuristics that depend on static analysis information, such as hard-coded strings, will not be hindered by obfuscation techniques such as binary packing.

Additionally, the current identification heuristics can be expanded to utilise additional data sources collected during LiSa analysis. All network traffic is logged within Packet Capture (pcap) files during the LiSa sandboxing stage. The captured packet information can be further analysed using heuristic analysis techniques to increase C2 and P2P detection accuracy. The VisiBot Processing System also collects several log files during sandbox analysis, including program output logs and machine logs. This information can be used to infer botnet characteristics and further increase the detection accuracy of VisiBot.

Lastly, a significant limitation of the VisiBot Processing System is the honeypot data retrieval process from the Bad Packets [10] honeypot service, as new results are only made available on an hourly basis. In some cases, this waiting period limits the number of malware samples extracted from remote code execution attempts conducted by malicious bots from botnets such as Mozi. Sample URLs are often hosted on local HTTP servers using temporary ports; thus, extracted URLs have a short and unpredictable life expectancy. This issue may be resolvable in future iterations of VisiBot by employing a stream-based honeypot collection system instead of an HTTP API. The transition to a streaming API would allow packets to be processed as soon as the honeypot network detects/processes them, mitigating any additional waiting time between queries. The immediate processing of honeypot packets would allow for quicker extraction of unpredictable payload URLs, which, as observed, are becoming increasingly more common amongst de-centralised botnets.

A Appendices

A.1 High-level Overview of VisiBot Processing System

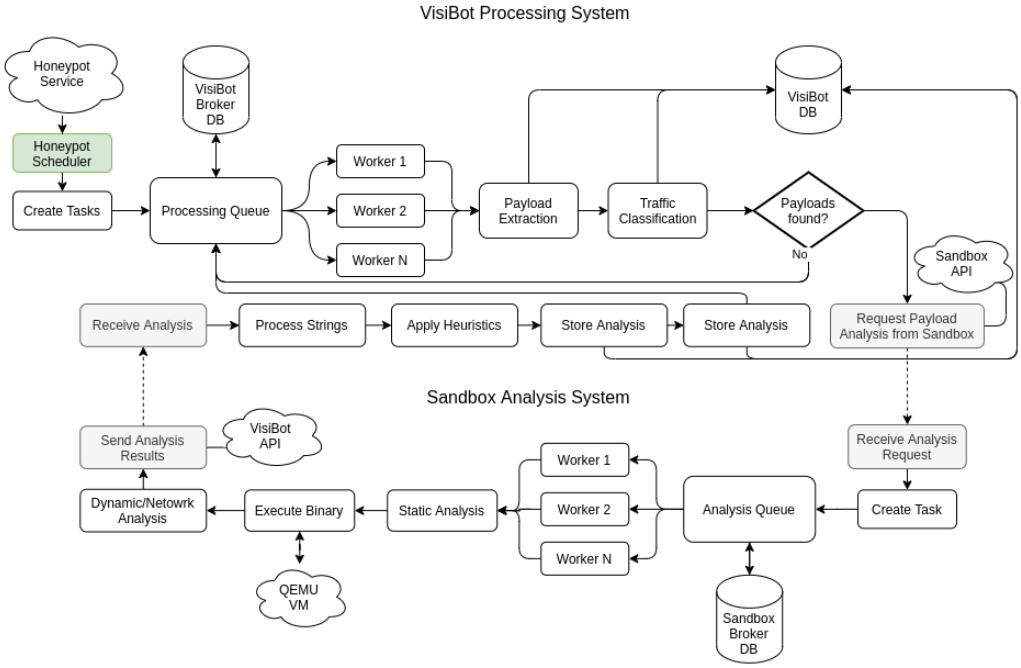


Figure A.1: *High-level flow-diagram overview of the VisiBot Processing System with sandbox integration.*

A.2 VisiBot MongoDB ER-Diagram

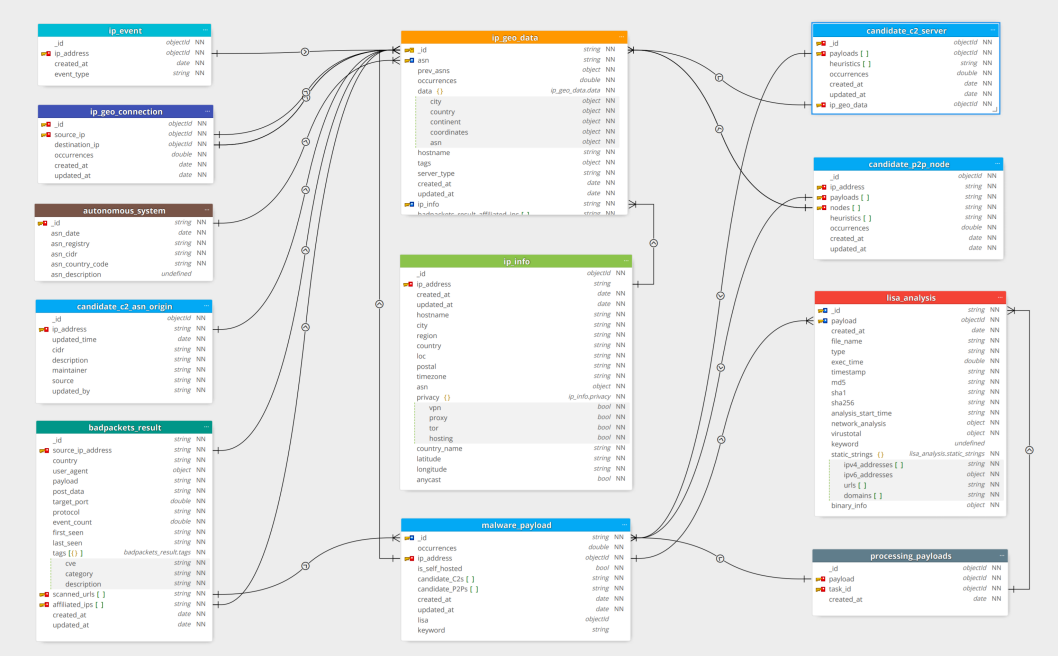


Figure A.2: *Reverse Engineered ER Diagram: VisiBot MongoDB Database.*

A.3 Example Bad Packets Honeypot Result

```
{
   "event_id": "fa3d87579d641...",
   "source_ip_address": "127.0.0.1",
   "country": "CN",
   "user_agent": "",
   "payload": "GET /board.cgi?cmd=cd /tmp;rm -rf *;wget
       http://127.0.0.1:58262/Mozi.a;chmod 777 Mozi.a;/tmp/Mozi.a varcron
       HTTP/1.0",
   "post_data": "",
   "target_port": 8080,
   "protocol": "tcp",
   "tags": [
      {
         "cve": "",
         "category": "IoT",
         "description": "Vacron NVR RCE"
      }
   ],
   "event_count": 1,
   "first_seen": "2021-01-29T22:13:02Z",
   "last_seen": "2021-01-29T22:13:02Z"
}
```

Listing A.1: *Example Bad Packets Honeypot Result, represented in JSON format.*

A.4 Example Multi-Binary Payload Bash Script

```
#!/bin/sh
cd /tmp;wget http://[redacted]/Prodigy.arm4;chmod +x Prodigy.arm4;./Prodigy.arm4
    ipcam
cd /tmp;wget http://[redacted]/Prodigy.arm5;chmod +x Prodigy.arm5;./Prodigy.arm5
    ipcam
cd /tmp;wget http://[redacted]/Prodigy.arm6;chmod +x Prodigy.arm6;./Prodigy.arm6
    ipcam
cd /tmp;wget http://[redacted]/Prodigy.arm7;chmod +x Prodigy.arm7;./Prodigy.arm7
    ipcam
```

Listing A.2: *An example bash-script used for botnet propagation. Once executed, the script will attempt to download and run 4 binaries of varying architectures.*

A.5 VisiBot MongoDB ER-Diagram

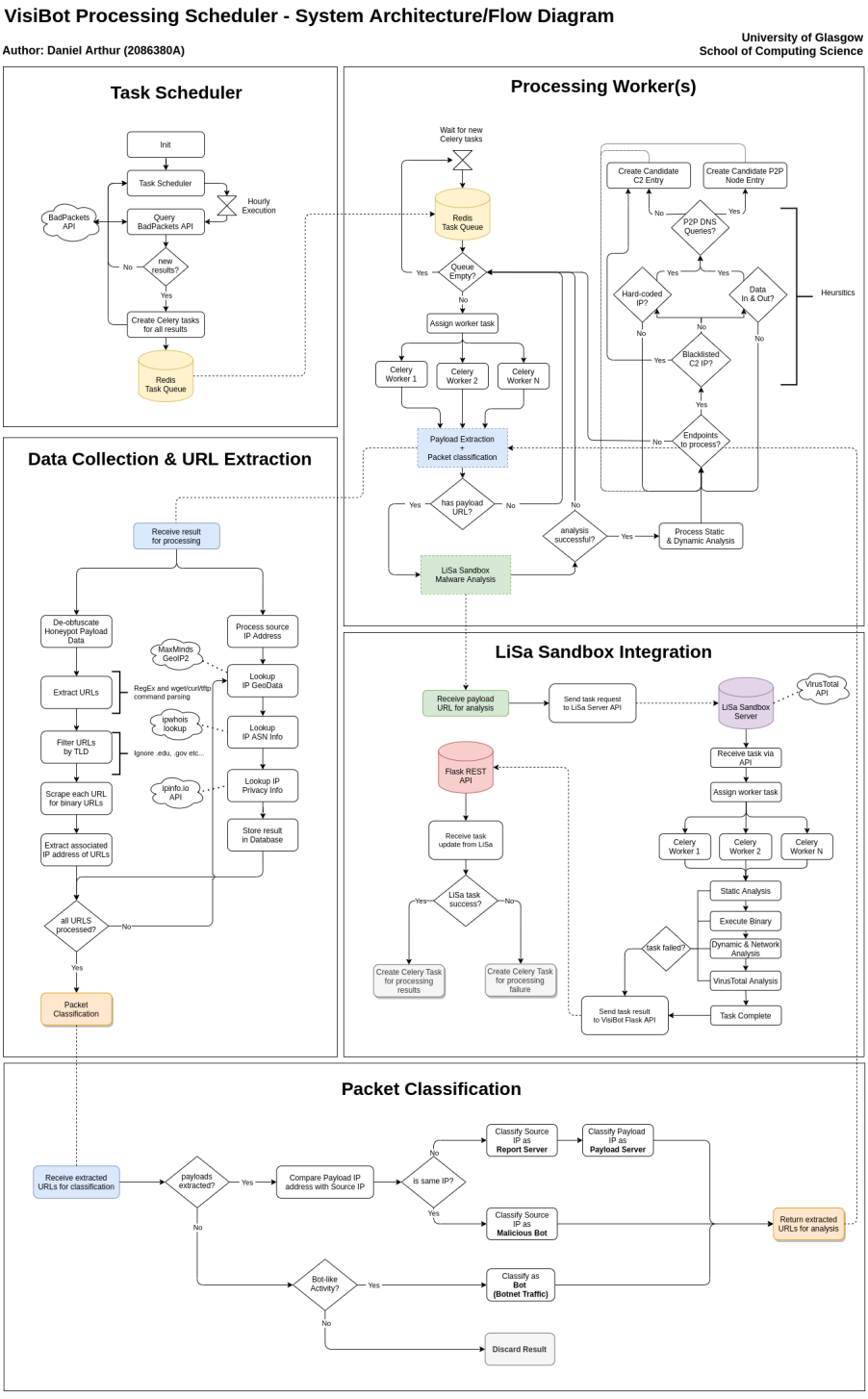


Figure A.3: *Overview flowchart diagram of the VisiBot Processing System.*

A.6 Example of VirusTotal Keyword Extraction

```
"virustotal": {
   "scans": {
      "AV_1": {
         "detected": false,
         "result": null,
         "update": "20210306"
      },
      "AV_2": {
         "detected": true,
         "version": "...",
         "result": "Trojan.GenericKD.44832991",
         "update": "20210307"
      },
      "AV_3": {
         "detected": true,
         "result": "Trojan.Mirai.Linux.78021",
         "update": "20210304"
      },
      "AV_4": {
         "detected": false,
         "result": null,
         "update": "20210305"
      },
      "AV_5": {
         "detected": true,
         "result": "a variant of Linux/Mirai.A",
         "update": "20210307"
      },
      "AV_6": {
         "detected": true,
         "result": "Backdoor.Linux.MIRAI.",
         "update": "20210307"
      },
      ...
   }
}
```

Listing A.3: *Example of keyword extraction from VirusTotal AV scan results. Positive scans and extracted keywords are highlighted in* ***bold****. This example illustrates the extraction of the top-most occurring keyword* ***mirai*** *for the given malware sample.*

A.7 VisiBot Web Application: Peer-to-Peer visualisation

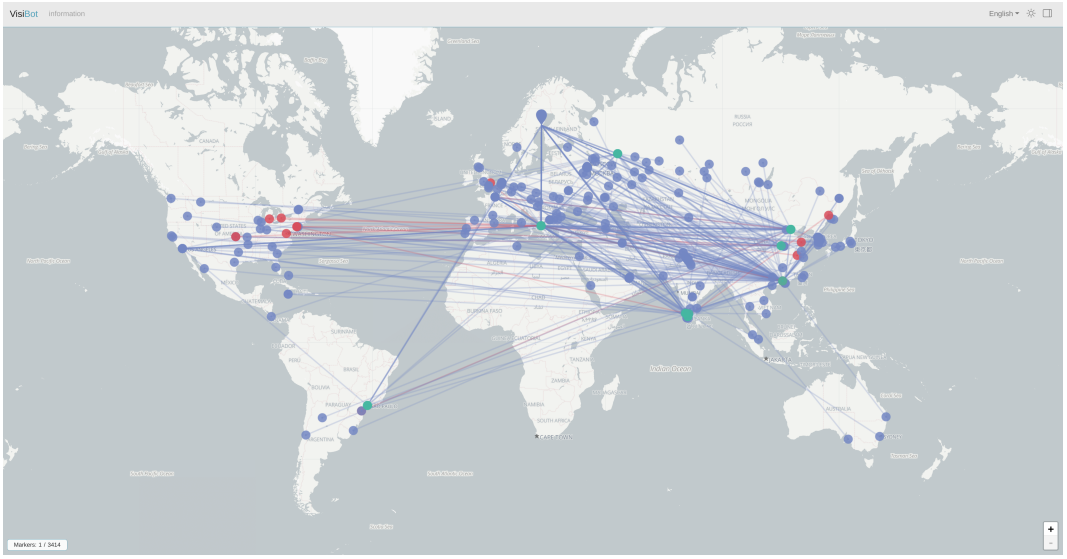


Figure A.4: *Screenshot of Peer-to-Peer connections shown within the VisiBot Interactive Map.*

A.8 Malware Analysis Port Statistics (TCP vs UDP)

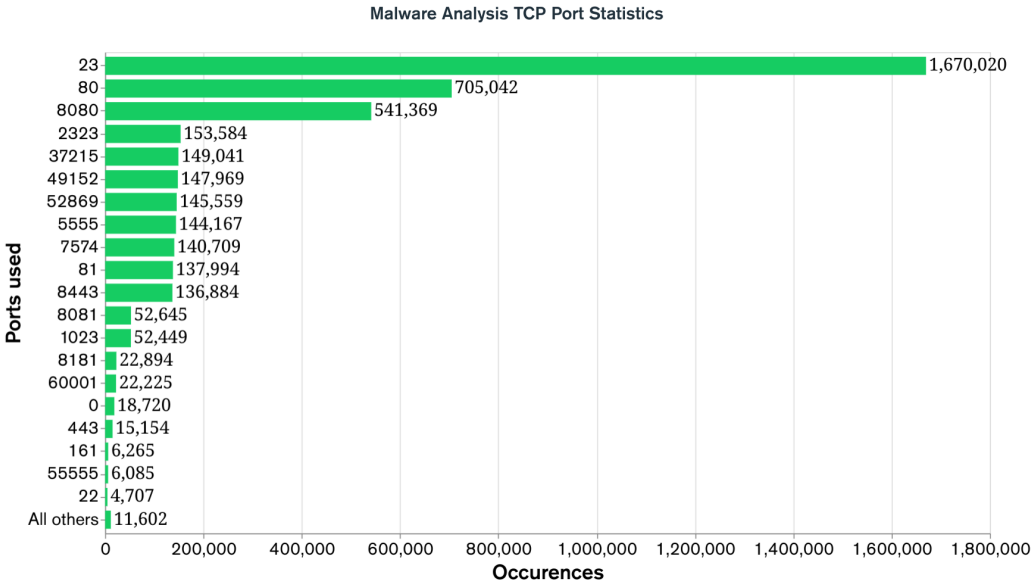


Figure A.5: Malware analysis TCP port usage statistics

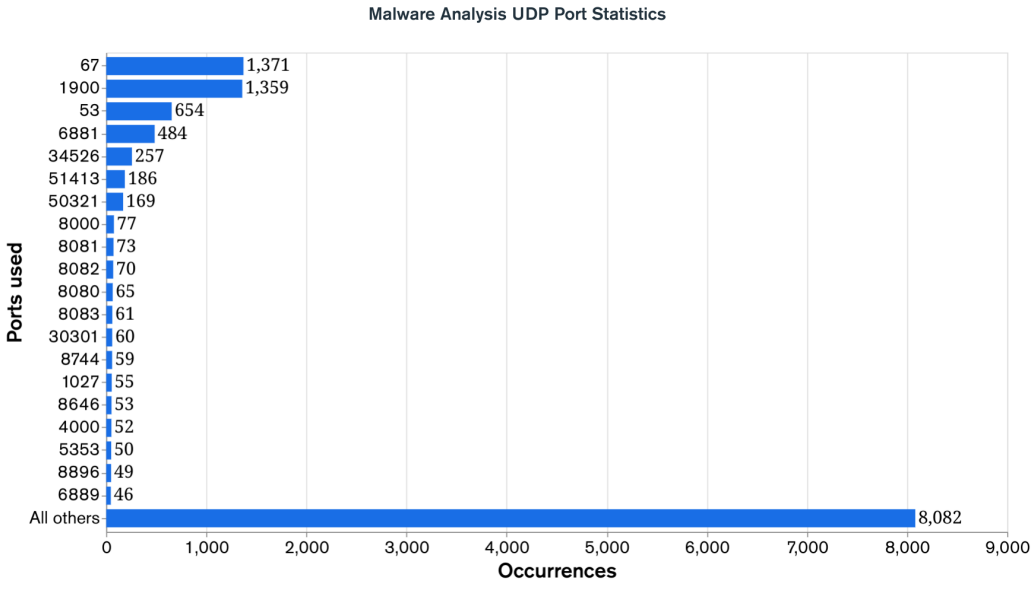


Figure A.6: Malware analysis UDP port usage statistics

A.9 Word-cloud of malware binary filenames.

Figure A.7: *Word-cloud of malware binary filenames collected by VisiBot. Larger word-size represents higher occurrence.*

A.10 Autonomous Systems most frequently associated with botnet traffic.

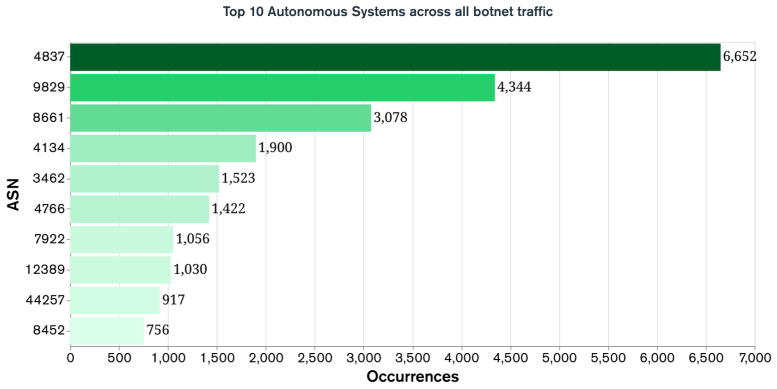


Figure A.8: *Top 10 Autonomous Systems based across all traffic.*

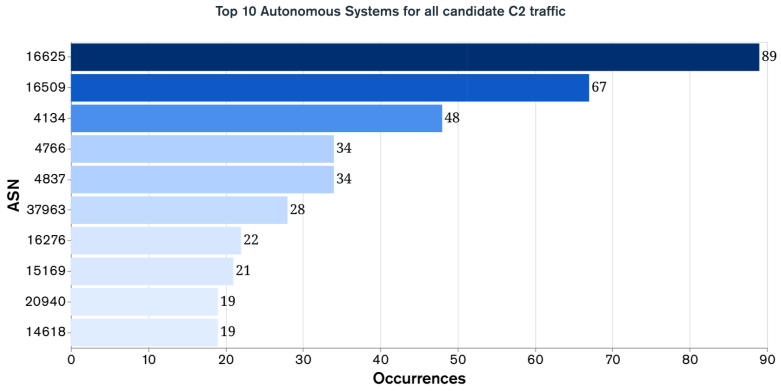


Figure A.9: *Top 10 Autonomous Systems based across all candidate C2 traffic.*

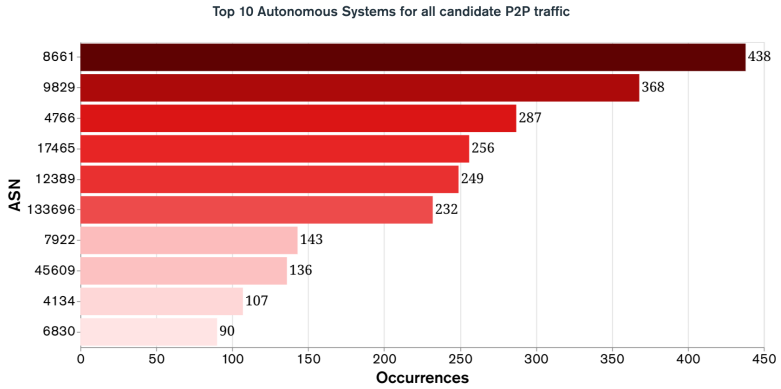


Figure A.10: *Top 10 Autonomous Systems across all P2P traffic.*

6 Bibliography

[1] Mark Maunder. Dyndns is currently being ddos'd – may affect your site, 2016. URL https://www.wordfence.com/blog/2016/10/dyndns-currently-ddosd-may-affect-site/. [Accessed March 24, 2021].

[2] Lexico. Definition of botnet, 2021. URL https://www.lexico.com/definition/botnet. [Accessed April 7, 2021].

[3] Michael Mimoso. What is iot?, 2016. URL https://threatpost.com/dyn-ddos-could-have-topped-1-tbps/121609/. [Accessed March 24, 2021].

[4] Oracle. What is iot?, 2021. URL https://www.oracle.com/uk/internet-of-things/what-is-iot/. [Accessed March 24, 2021].

[5] Arm. Iot devices, 2021. URL https://www.arm.com/glossary/iot-devices. [Accessed March 24, 2021].

[6] Knud Lasse Lueth. State of the iot 2020: 12 billion iot connections, surpassing non-iot for the first time. https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/, 2020. [Accessed March 24, 2021].

[7] Nirmal Misra. Owasp's top 10 iot vulnerabilities, 2021. URL https://www.deviceauthority.com/blog/owasp-s-top-10-iot-vulnerabilities. [Accessed March 24, 2021].

[8] Brian Krebs. Source code for iot botnet 'mirai' released, 2016. URL https://krebsonsecurity.com/2016/10/source-code-for-iot-botnet-mirai-released/. [Accessed March 24, 2021].

[9] C.G.J Putman, Nieuwenhuis, and J. M. Lambert. Business model of a botnet. *26th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, 2018. doi: 10.1109/PDP2018.2018.00077. URL https://doi.org/10.1109/PDP2018.2018.00077.

[10] Bad Packets LLC. Bad packets® cyber threat intelligence. https://badpackets.net/threat-intelligence/, 2017–2021. [Accessed March 24, 2021].

[11] Daniel Uhříček. Lisa - linux sandbox. https://github.com/danieluhricek/LiSa, 2019–2020. [Accessed March 24, 2021].

[12] Young Hoon. Moon, Eunjin. Kim, Suh Mahn. Hur, and Huy Kang. Kim. Detection of botnets before activation: an enhanced honeypot system for intentional infection and behavioral observation of malware. *Next Generation Communication and Network Security*, 5: 1094–1101, 2012. doi: 10.1002/sec.431. URL https://doi.org/10.1002/sec.431.

[13] Niels Provos. Detux sandbox, 2008. URL http://www.honeyd.org/. [Accessed March 24, 2021].

[14] Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. Iotpot: A novel honeypot for revealing current iot threats. *Journal of Information Processing*, 24:522–533, 2016. doi: 10.2197/ipsjjip.24.522. URL https://dx.doi.org/10.2197/ipsjjip.24.522.

[15] Manos Antonakakis, Tim April, Michael Bailey, Matthew Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zone Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. *26th USENIX Security Symposium*, pages 1093–1100, 2017. doi: 10.5555/3241189.3241275. URL https://dl.acm.org/doi/10.5555/3241189.3241275.

[16] Gabriel Bastos, Artur Marzano, Osvaldo Fonseca, Elverton Fazzion, Cristine Hoepers, Klaus Steding-Jessen, Chaves Marcelo H.P.C, Italo Cunha, Dorgival Guedes, and Wagner Meira. Identifying and characterizing bashlite and mirai c&c servers. *2019 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6, 2019. doi: 10.1109/ISCC47284.2019.8969728. URL https://doi.org/10.1109/ISCC47284.2019.8969728.

[17] João Marcelo Ceron, Klaus Steding-Jessen, Cristine Hoepers, Lisandro Zambenedetti Granville, and Cíntia Borges Margi. Improving iot botnet investigation using an adaptive network layer. *Sensors 2019, 19, 727*, 2019. doi: 10.3390/s19030727. URL https://doi.org/10.3390/s19030727.

[18] Vikas Iyengar and Rahul Binjve. Detux sandbox, 2016. URL https://github.com/detuxsandbox/detux. [Accessed March 24, 2021].

[19] Owen P Dwyer, Angelos K Marnerides, Vasileios Giotsas, and Troy Mursch. Profiling iot-based botnet traffic using dns. *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019. doi: 10.1109/GLOBECOM38437.2019.9014300. URL 10.1109/GLOBECOM38437.2019.9014300.

[20] Stephen Herwig, Katura Harvey, George Hughey, Richard Roberts, and Dave Levin. Measurement and analysis of hajime, a peer-to-peer iot botnet. *Network and Distributed Systems Security (NDSS) Symposium 2019*, 2019. doi: 10.14722/ndss.2019.23488. URL https://dx.doi.org/10.14722/ndss.2019.23488.

[21] Gartner, Inc. Definition of ib (integration broker), 2021. URL https://www.gartner.com/en/information-technology/glossary/ib-integration-broker. [Accessed March 24, 2021].

[22] Vladimir Agafonkin. Leaflet.markercluster 0.1 released, 2012. URL https://leafletjs.com/2012/08/20/guest-post-markerclusterer-0-1-released.html. [Accessed March 24, 2021].

[23] Python Software Foundation. What's new in python 3.9. https://docs.python.org/3/whatsnew/3.9.html, 2021. [Accessed March 24, 2021].

[24] Ask Solem and contributors. Celery - distributed task queue. https://docs.celeryproject.org/en/stable/index.html, 2009–2021. [Accessed March 24, 2021].

[25] Redis Labs. Redis. https://redis.io/, 2009–2021. [Accessed March 24, 2021].

[26] Mher Movsisyan and contributors. Flower. https://github.com/mher/flower, 2016–2021. [Accessed March 24, 2021].

[27] Docker, Inc. Overview of docker compose. https://docs.docker.com/compose/, 2013–2021. [Accessed March 24, 2021].

[28] MaxMind, Inc. Geoip2 databases. https://man7.org/linux/man-pages/man5/crontab.5.html, 2002–2021. [Accessed March 24, 2021].

[29] Paul Vixie and contributors. crontab(5) — linux manual page. https://www.maxmind.com/en/geoip2-databases, 2012–2020. [Accessed March 24, 2021].

[30] MongoDB Inc. Mongodb. https://www.mongodb.com/, 2009–2021. [Accessed March 24, 2021].

[31] Harry Marr and contributors. mongoengine. https://github.com/MongoEngine/mongoengine, 2013–2021. [Accessed March 24, 2021].

[32] MongoDB Inc. Mongodb atlas. https://www.mongodb.com/cloud/atlas, 2021. [Accessed March 24, 2021].

[33] Kenneth Reitz and contributors. requests - pypi. https://pypi.org/project/requests/, 2011–2021. [Accessed March 24, 2021].

[34] Jan Lipovský. urlextract - pypi. https://pypi.org/project/urlextract/, 2020. [Accessed March 24, 2021].

[35] CyberCiti. Shebang. https://bash.cyberciti.biz/guide/Shebang, 2020. [Accessed March 24, 2021].

[36] Artur Barseghyan and contributors. tld. https://github.com/barseghyanartur/tld, 2021. [Accessed March 24, 2021].

[37] John Kurkowski and contributors. tldextract. https://github.com/john-kurkowski/tldextract, 2020. [Accessed March 24, 2021].

[38] Python Software Foundation. socket - low-level network interface. https://docs.python.org/3/library/socket.html, 2021. [Accessed March 24, 2021].

[39] Troy Mursch. Bad packets - what does mirai-like scan mean? https://badpackets.net/mirai-like-botnet-one-year-review-and-a-new-website/, 2021. [Accessed March 24, 2021].

[40] IPinfo. Ipinfo - developers api. https://ipinfo.io/developers, 2013–2021. [Accessed March 24, 2021].

[41] AFRINIC. Afrinic the region internet registry (rir) for africa. https://afrinic.net/, 2021. [Accessed March 24, 2021].

[42] APNIC. Apnic. https://www.apnic.net/, 2021. [Accessed March 24, 2021].

[43] RIPE NCC. Ripe network coordination centre. https://www.ripe.net/, 2021. [Accessed March 24, 2021].

[44] Pivotal Software. Rabbitmq. https://www.rabbitmq.com/, 2007–2020. [Accessed March 24, 2021].

[45] Sergi Alvarez and contributors. Radare2 (r2). https://github.com/radareorg/radare2, 2006–2021. [Accessed March 24, 2021].

[46] Fabrice Bellard. https://www.qemu.org/. https://www.qemu.org/, 2011 –2020. [Accessed March 24, 2021].

[47] Peter Korsgaard and contributors. Buildroot - making embedded linux easy. https://buildroot.org/, 2005–2021.

[48] VirusTotal. Virustotal. https://www.virustotal.com/, 2004–2021. [Accessed March 24, 2021].

[49] Daniel Uhříček and Daniel Arthur. Lisa - linux sandbox. https://github.com/denBot/LiSa, 2020–2021. [Accessed March 24, 2021].

[50] Kevin A. Roundy and Barton P. Miller. Binary-code obfuscations in prevalent packer tools. *ACM Computing Surveys - October 2013*, page 3, 2013. doi: 10.1145/2522968.2522972. URL https://doi.org/10.1145/2522968.2522972.

[51] Free Software Foundation, Inc. strings(1) - linux man page. https://linux.die.net/man/1/strings, 1991–2009. [Accessed March 24, 2021].

[52] Markus Oberhumer, László Molnár, and John F. Reiser. Upx - the ultimate packer for executables. https://upx.github.io/, 1996–2020. [Accessed March 24, 2021].

[53] Armin Ronacher and contributors. Flask documentation (1.1.x). https://flask.palletsprojects.com/en/1.1.x/, 2010–2021. [Accessed March 24, 2021].

[54] Barracuda Networks. Barracuda reputation block list (brbl). https://www.barracudacentral.org/rbl, 2009–2020. [Accessed March 24, 2021].

[55] abuse.ch. Abuse.ch: Fighting malware and botnets. https://abuse.ch/, 2017. [Accessed March 24, 2021].

[56] Spam Rats. Spam rats: Really annoying trouble spots. https://www.spamrats.com/, 2021. [Accessed March 24, 2021].

[57] SORBS. Sorbs (spam and open-relay blocking system. http://www.sorbs.net/, 2002–2019. [Accessed March 24, 2021].

[58] The Spamhaus Project SLU. The spamhaus project: Zen. https://www.spamhaus.org/zen/, 1998–2021. [Accessed March 24, 2021].

[59] Alexandre Chopin, Sebastien Chopin, and Pooya Parsa. Nuxt.js - the intuitive vue framework. https://nuxtjs.org/, 2016–2021. [Accessed March 24, 2021].

[60] OpenJS Foundation. Express - node.js web application framework. https://expressjs.com/, 2010–2021. [Accessed March 24, 2021].

[61] OpenJS Foundation. Node.js. https://nodejs.org/en/, 2009–2021. [Accessed March 24, 2021].

[62] Automaticc, Inc. Mongoose odm. https://mongoosejs.com/, 2021. [Accessed March 24, 2021].

[63] Alex Regan, Jacob Müller, Vitaly Mosin, Pooya Parsa, and Troy Morehouse. Bootstrapvue. https://bootstrap-vue.org/, 2017–2021. [Accessed March 24, 2021].

[64] Vladimir Agafonkin and contributors. Leaflet.js. https://leafletjs.com/, 2010–2021. [Accessed March 24, 2021].

[65] MongoDB Inc. $graphlookup aggregation. https://docs.mongodb.com/manual/reference/operator/aggregation/graphLookup/, 2009–2021. [Accessed March 24, 2021].

[66] Python Software Foundation. unittest - unit testing framework - python 3.9.2 documentation. https://docs.python.org/3/library/unittest.html, 2021. [Accessed March 29, 2021].

[67] Kirill Klenov and contributors. pylama. https://github.com/klen/pylama, 2013–2019. [Accessed March 29, 2021].

[68] Guido van Rossum, Barry Warsaw, and Nick Coghlan. Pep-8 – style guide for python code. https://www.python.org/dev/peps/pep-0008/, 2001–2013. [Accessed March 30, 2021].

[69] OpenJS Foundation. Eslint. https://eslint.org/, 2013–2021. [Accessed March 29, 2021].

[70] Linus Torvalds and contributors. Git. https://git-scm.com/, 2005–2021. [Accessed March 29, 2021].

[71] GitHub, Inc. Github. https://github.com/, 2008–2021. [Accessed March 30, 2021].

[72] Linus Torvalds and contributors. Git hooks. https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks, 2005–2021. [Accessed March 29, 2021].

[73] Travis CI, GmbH. Travis ci. https://travis-ci.com/, 2011–2021. [Accessed March 29, 2021].

[74] Netlab 360. Mozi, another botnet using dht. https://blog.netlab.360.com/mozi-another-botnet-using-dht/, 2019. [Accessed March 24, 2021].

[75] speedguide.net. Port 53 details. https://www.speedguide.net/port.php?port=53, 1999–2021. [Accessed March 24, 2021].

[76] NetworkX Developers and contributors. Networkx documentation. https://networkx.org/, 2014–2020. [Accessed March 24, 2021].

[77] NetworkX Developers and contributors. Networkx kcomponents approximation algorithm. https://networkx.org/documentation/stable//reference/algorithms/generated/networkx.algorithms.approximation.kcomponents.k_components.html, 2014–2020. [Accessed March 24, 2021].

[78] Philip Hane and contributors. Ip asn lookup - ipwhois 1.2.0 documentation. https://ipinfo.io/developers, 2013–2020. [Accessed March 24, 2021].