University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

VisiBot: automated detection of IoT botnets using heuristic analysis techniques.

Daniel Arthur

April 1, 2021

Abstract

Every abstract follows a similar pattern. Motivate; set aims; describe work; explain results.

"XYZ is bad. This project investigated ABC to determine if it was better. ABC used XXX and YYY to implement ZZZ. This is particularly interesting as XXX and YYY have never been used together. It was found that ABC was 20% better than XYZ, though it caused rabies in half of subjects."

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 1, 2021

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 caused by this attack directly impacted a multitude of services, including Cloudflare, Amazon, Twitter, Netflix, and GitHub. [1] Unlike attacks 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. The Mirai botnet's 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 operate on infected computer systems, such as desktops and laptops, the Mirai botnet exclusively targeted vulnerable IoT devices, including unprotected IP/CCTV cameras and internet 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.

Following a recent surge in popularity for IoT devices, several manufacturers have expanded to compete in a new and highly competitive market to produce IoT devices. Due to a lack of consideration for security and update-ability, IoT devices' rapid production 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] 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] Adversely, the cost of identifying, mitigating, and recovering from botnet DDoS attacks is believed to be in the millions. As DDoS attacks have a detrimental impact on targeted services and external dependants, significant effort is being placed on the identification and alleviation of botnets by being placed on government entities' likes, Internet Service Providers (ISP), commercial businesses, and Cyber Threat Intelligence agencies. Botnets are conventionally controlled using centralised communication strategies, where connected all bots communicate with a central Command & Control (C2) server. As a bot must first receive a command containing the target before participating in an attack, authorities have focused on identifying, pursuing, and shutting down Command & Control servers, rendering corresponding botnets useless. However, botnet developers have sought alternative communication protocols to alleviate any interference with authorities. In particular, IoT botnets have continued to evolve in recent years, implementing DNS and Peer-to-Peer (de-centralised) communication protocols to limit authorities' interference. The constant evolutionary nature of botnets has proven to be a continual challenging for parties concerned with preventing cyber-attacks. Attackers are constantly seeking to exploit new vulnerabilities, target new devices, and employ various communication protocols. As the number of IoT devices continues to grow, so too will the number of vulnerabilities in IoT devices, presenting a bountiful range of devices that IoT botnets can recruit. Additionally, as botnets continue to evolve and utilise complex communication systems such as Peer-to-Peer, detecting the presence of such botnets during their propagation stage will become considerably more difficult.

In consideration of the current challenge of automatically identifying evolving IoT botnets, I propose VisiBot. VisiBot is an automated IoT botnet detection and visualisation tool. The VisiBot Processing System is a framework used by VisiBot, which employs a globally distributed honeypot network to conduct automatic classification, malware extraction, sandboxing and network analysis of all potentially malicious packets. Using an automated Linux Malware Analysis tool known as LiSa, [10] the VisiBot Processing System collects a wide array of static, dynamic, and network-based information through the purposeful execution of malware samples extracted from honeypot intrusion attempts. By analysing the traffic generated by malware samples, botnets can be identified during their propagation stage before attacks. VisiBot explicitly targets IoT-based botnets, using a sandboxing system that supports up to 5 IoT CPU architectures, including i386, ARM, and MIPS. By applying four novel heuristics to the static and network information collected, VisiBot facilitates the automatic detection of candidate IoT botnet Command & Control servers and Peer-to-Peer botnet nodes. A real-time interactive Web Application was also built to visualise the geographic density and distribution of currently propagating IoT botnets. Using a combination of heterogeneous data-sources, a real-time representation of botnet activity is shown through the clustering of all geographic coordinates of all IP addresses classified by the VisiBot Detection System within a 24-hour window.

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. Using four identification heuristics, a total of 1,303 candidate Command & Control servers and 6,876 Peer-to-Peer botnet nodes were identified. 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 Background

2.1 The evolution of Botnets

The earliest of botnets are believed to have originated from back-door concepts first seen in computer viruses discovered in 1999, namely the Sub7 Trojan and the Pretty Park Worm. Both viruses established the notion of using the IRC protocol to send execution commands, disguised as IRC messages, to remotely infected hosts through an IRC server. This discovery ultimately led to the propagation and coordination of the infected computer networks using the IRC protocol as a back-door, sparking the development of a new variant of malware used to cultivate and control large networks infected 'bot' devices, known as botnets. [11]

According to Schiller et al. [12], one of the first IRC botnets is believed to be the Global Threat (GT) botnet. This worm-like malware manipulates the IRC communication protocol as a means of centralised botnet communication. Upon infecting a Windows computer system, the infected host would remotely connect to an IRC server, waiting to receive commands from a central Command & Control (IRC) server whilst disguised as a seemingly legitimate IRC client. This centralised communication method through a central IRC server proved advantageous for botnet owners, as botnet traffic is concealed in plain-sight amongst legitimate IRC traffic. Additionally, through the propagation of IRC-based botnets, owners could use infected hosts to perform out a variety of commands, including port scanning, password sniffing, and DDoS attacks. This technique led to the development of several other notorious botnets, such as the MaXiTE IRC botnet. However, authorities were eventually able to detect such IRC botnets through intrusion detection, binary analysis, and network traffic monitoring. As infected bots had to know the IP address and port of the IRC communication server, authorities could shut down offending IRC servers following identification through inspection of sources such as malware binaries and network traffic.

Given the various shortcomings of IRC-based botnet communication, botnet developers began to employ other protocols and application layer services as a means of communication, such as the now extremely popular HyperText Transfer Protocol (HTTP) and Domain Name Service (DNS) protocols. In particular, botnets, phishing websites, and other malware distributors began to exploit the DNS protocol by using 'Fast Flux' domains. A Fast Flux DNS enables botnets to "hide behind an ever-changing network of compromised hosts, ultimately acting as proxies" [13]. This presents challenges for botnet mitigation as Command & Control servers masked behind such botnet traffic are significantly more challenging to identify. According to Morton and Barth [13], the Storm botnet is one of the first botnets to mask remote servers behind using Fast Flux DNS. Deployed in 2007, the Storm botnet is also known as one of the first de-centralised botnets through the transmission of botnet communication through standard Peer-to-Peer protocols. Early iterations of the Storm botnet have used OVERNET; a Peer-to-Peer distributed hash table (DHT) routing protocol, which bots used to find other bots and receive commands through a Peer-to-Peer network. [14] This transition from centralised to de-centralised communication is said to have marked a significant point in the evolution of botnets, as botnets can be controlled without the concern of deployment and obfuscation factors presented when using a central C2 server.

Similarly, El Defrawy et al. [15] discuss how malicious actors could exploit standard P2P protocols such as BitTorrent to allow for de-centralised botnets and the coordination of large-scale DDoS attacks via peer-to-peer networks. As all bit-torrent clients rely on interacting with trackers, one possible exploit includes disguising a target DDoS victim as a bit-torrent tracker, generating a highly distributed attack using connected P2P nodes. An alternative exploit identified by El Defrawy et al. [15] is that BitTorrent Distributed Hash Tables (DHTs) can be used to coordinate botnet attacks on a victim IP address through botnet owners pinging a DHT with a target IP address, which bots will then target.

Published anonymously due to potentially unethical and potentially illegal experimentation, the *Internet Census 2012* [16] is perhaps one of the first papers to discuss the insecurities of IoT devices within the context of botnet propagation. In this paper, it was found that a significant number of IoT devices were vulnerable to unauthorised remote access by attempting to log in using common factory-set usernames and passwords created by manufacturers. Following this discovery, the author developed Carna - a distributed port-scanning framework used for botnet propagation. By infecting select vulnerable devices with a port-scanning binary, infected devices would scan other vulnerable machines over the telnet protocol using various scanning tools and methods. This activity ultimately led to the identification of 1.3 billion in-use IP addresses. Additionally, the Carna botnet expanded to over 420,000 vulnerable devices, marking a new and highly effective propagation technique for botnet developers to employ.

2.2 An overview of IoT Botnets

Shortly after the anonymous release of the findings presented in the Internet Census 2012, a new variation of botnets began to surface, targeting a diverse selection of operating systems and CPU architectures common to IoT devices. Aidra, also known as Lightaidra, [17] was discovered in January of 2012 by ATMA.ES. [18] Security researchers observed a substantially high number of telnet attacks were identified to be originating from various ARM-based IoT devices. [19] It was later found that despite the botnet initially targeting ARM-based IoT devices, the malware binaries used for propagation could also be compiled to several other IoT architectures, including MIPS, MIPSel, PPC, x86/x86-64, and SuperH. Aidra could propagate similarly to that of the Carna botnet, by first port-scanning for vulnerable IoT devices using the telnet protocol. Once identified, the botnet attempts to infect the device through brute-forcing known login credentials and sending malicious requests containing command-line injection code, which downloads and executes a malware binary. Like previous botnets, the Aidra botnet uses the IRC communication protocol as a back-door, such that infected devices connect to a central IRC C2 server and await new commands from the botnet owner. However, as several IoT devices operate using different CPU architectures, the command-line injection code attempts to download and execute all architecture variations of the Aidra malware binary from a remotely hosted Payload server.

Bashlite, also known as Gafgyt, LizardStresser, Lizkebab, PinkSlip, Qbot and Torlus, [20] was identified in 2014 as a new botnet variant that targets specific IoT devices. The first iteration of the Bashlite botnet malware initially targeted various router devices running outdated versions of the BusyBox software to exploit the GNU Bash vulnerability known as ShellShock. [21] As the versions of GNU Bash affected by this vulnerability inadvertently allow for the processing of trailing strings after function definitions in the values of environment variables, attackers could use this exploit to execute arbitrary code through the likes of command-line injection. [22] All scanning activity is controlled by the C2 server through sending commands to infected bots. Marzano et al. [23] Upon receiving a scan command, a bot will begin to randomly scan SSH and Telnet ports for vulnerable devices and report potential targets to a loader server. The loader server will attempt to login to the device via SSH or Telnet protocols through brute-forcing commonly used passwords. Once the server gains unauthorised access, it will download, execute, and infect the device with a botnet malware binary. The IP address and port of the C2 server are

often hard-coded within the binary so that a handshake between the bot and C2 server can be performed once the device is infected. Using Bashlite's command distribution functionality, a bot owner can perform various attacks, including DDoS, TCP Flood, and SYN Flood attacks. These commands are typically un-encrypted and sent from a C2 server as plain-text. Additional functionality includes distributing binary updates using management commands, collecting statistical information on a botnet using query commands, and stopping on-going attacks using interrupt commands.

First identified in 2016 by MalwareMustDie [24], the Mirai botnet improves upon Bashlite in several respects. [23] The most considerable change proposed by the Mirai botnet was the brute-forcing of a broader range of potential target IoT devices, including DVRs, routers, and IP Cameras. Due to the infection of non-conventional devices, it is proposed that the reason for the lack of detection was due to the difficulty of detecting and extracting malware samples from infected IoT devices. [24]. Honeypot servers could not detect or collect Mirai binaries as particular ports, vulnerabilities, and device architectures common to IoT devices were being used for propagation, allowing for botnet growth to go unnoticed for a considerable amount of time. In a study by [23], several differences between Mirai and Bashlite were observed. Like that of Bashlite, Mirai also employs telnet port-scanning and login credential brute-forcing techniques for botnet propagation. However, unlike Bashlite, password dictionaries comprising of default credentials of a wide range of IoT devices were used.

By targeting a more comprehensive range of IoT devices, the Mirai botnet can achieve better global distribution, as devices such as IP Cameras and DVRs are widely used across various regions. Unlike Bashlite, Mirai C2 servers are limited to the distribution of attack commands. All port-scanning activity is automatically performed by infected bots, emitting the distribution of C2 port-scanning commands. When a device is infected with Mirai, the binary will execute code that blocks access to ports commonly used for infection, such as 23 and 2323, and actively removes competitor botnet malware to prevent future re-infection. Mirai also utilises DNS-based botnet communication, removing the use of hard-coded IP address and ports common to Bashlite variants. DNS allows for quick and easy hot-swapping of C2 server IP addresses using DNS name records. Should a C2 server be taken down by authorities, the botnet owner can change the IP address. As mirai botnets communicate with C2 servers using a domain instead of an IP address, bots can continue to be controlled even if the C2 server is compromised. That is, the IP address associated with the DNS record can be changed on the spot. DNS also supports various means of traffic encryption, enabling botnet owners to communicate via encrypted messages between bots through DNS over TLS or HTTPS, making packet analysis and traffic monitoring significantly more challenging.

Following the public release of the Mirai botnet in 2016, Antonakakis et al. [25] propose that the availability of the Mirai botnet source-code has sparked a new generation of Mirai-based botnet variants, marking a new point in the evolutionary scale of botnet history. Hajime, a recent variant of Mirai, is evidence of such evolution. Unlike Mirai, the Hajime botnet has adopted a de-centralised strategy to botnet communication through current Peer-to-Peer technologies. This strategy eliminates the need for a centralised C2 server, as peer-to-peer botnets can receive commands and files from each other. As there is no central point of communication, the 'serverless nature of P2P botnets makes identification and mitigation significantly more challenging than centralised botnet infrastructures. Herwig et al. [26] found that the Hajime botnet employs P2P botnet communication through a Distributed Hash Table (DHT) overlay network available via the BitTorrent P2P protocol. Using a DHT overlay network allows for the distribution of software updates and attack commands across all connected nodes within a given P2P botnet network. Once a device is infected with the Hajime payload, it will join a DHT through a known peer and download content or receive commands from connected peer-nodes marked as 'seeders'. As the BitTorrent DHT network is publicly available and actively used by legitimate peers, bots often interact with legitimate peers to introduce noise during any potential network analysis.

One significant characteristic of the de-centralised Hajime botnet is that infected bots are used to self-propagate by individually downloading, hosting, and attacking other IoT devices using Hajime malware binaries exclusive to the architecture of the bot.

Despite the feature-set offered by the Hajime Peer-to-Peer IoT botnet, experts have observed a recent influx in de-centralised botnet traffic coming from a new Mirai botnet variant known as Mozi. [27] In the same way as Hajime, Mozi utilises a de-centralised Peer-to-Peer structure using Distributed Hash Tables. However, the Mozi botnet uses it's own DHT protocol for propagation, utilising various public nodes from popular P2P services as bootstraps for guiding infected bots to the botnet DHT. These nodes are specified in a config file downloaded following successful infection through telnet brute-forcing and command-line injection. Similarly to Hajime, the Mozi botnet also employs a unique strategy for botnet propagation, as infected devices actively self-host malware binaries. Once infected, the bot will download Mozi malware binaries and DHT configuration files and set up an HTTP server assigned to a randomly selected port. The bot will attempt to infect other devices using the malware hosted on its own server, alleviating the need for payload or loader servers commonly used by Mirai botnets. The port number will change over time, ensuring binaries are no-longer retrievable from previously used ports after a given time-frame. Netlab 360 [27] observed that the Mozi botnet uses a specific identifier starting with `1:v4:` which is used to distinguish Mozi bots from legitimate P2P traffic. This identifier is followed by a flag consisting of 4 bytes. The first byte is randomly generated, the second is hard-coded in the Mozi config file, and the 3rd and 4th bytes are calculated using an algorithm found in Mozi's source code.

2.3 Defence against IoT Botnets

As the rise of IoT vulnerabilities continues to increase, Bertino and Islam [28] argue that IoT botnet research must remain a high priority amongst cyber-security specialists. The propagation and utilisation of such botnets have proven to pose drastic effects on even the most secured internet services. Bertino and Islam [28] state that there are eight categories of vulnerabilities common to IoT devices that must be resolved to mitigate attacks coordinated by malicious actors using IoT botnet malware. This list includes several vulnerabilities explicitly targeted by the Mirai botnet: the use of insecure web interfaces and control panels, known default login credentials, vulnerable web services, and insecure operating systems or firmware. The authors argue that monitor both the network traffic and device behaviour of IoT devices should remain a central concern for mitigating botnet activity, as even secured devices can still be infected with malware. Through analysing network traffic for anomalies using Network Behavioural Analysis (NBA), potential botnet activity can be investigated and countered accordingly. It is suggested that various metrics can be used to identify such anomalies, including monitoring network bandwidth and the usage of network protocols.

In the past, real-time botnet traffic analysis was challenging to coordinate. Thus, systems often relied on post-attack data such as network logs for identifying botnets. This reactive approach for botnet identification was not ideal as attacks need to take place first. Lividas et al. [29] proposed a solution for preemptively identifying botnets without relying on data collected from attacks that previously took place. Using machine learning classifiers, Lividas et al. [29] suggested that botnet traffic could be identified through IRC network traffic flow analysis. However, as both technology and the internet has advanced, the Peer-to-Peer and Hypertext Transfer Protocol (HTTP) protocols are now some of the most common means of communication for botnet traffic, gradually rendering the use of IRC-based botnets obsolete. Moon et al. [30] propose botnet identification through purposeful execution and dynamic/network analysis of malware binaries using an advanced honeypot system. Integrated with the back-end infrastructure of KORNET, the largest ISP service in Korea, this system showed a prediction accuracy of approximately 40% for future blacklisted botnet traffic. Despite the valuable information obtained through static

and dynamic analysis of malware binaries, this system's resulting low accuracy was likely due to un-optimal malware sample extraction from trivial data-sources.

Despite the use of conventional honeypots showing some success in the findings presented by Moon et al. [30], it is suggested by [24] that typical honeypot infrastructures are not able to detect IoT botnet activity effectively. Botnets such as Mirai target particular devices based on CPU architecture and exposed ports. Antonakakis et al. [25] propose that IoT botnet activity can be captured by configuring honeypots to mimic vulnerable IoT devices' characteristics by configuring them to use BusyBox and respond to requests with IoT-like headers. This method proved somewhat effective, logging connection attempts from over 80,000 foreign IP addresses over approximately five months and collecting a total of 151 unique malware binaries. Combining this honeypot information with various other sources, Antonakakis et al. [25] identified different trends common to Mirai botnets, including several infection strategies, commonly targeted ports, and an analysis of high profile attacks. The study also investigates geographic patterns using the MaxMind GeoIP 2 database for collecting IP address geolocation information. This information provided an insight into which countries are hot-spots for Mirai device infection and how detected botnets are globally distributed.

Similarly, Pa Pa et al. [31] also noted that current honeypot systems proved insignificant for extracting IoT malware information, and thus presented *IoTPOT*, an improved IoT honeypot system with sandbox integration. By imitating various IoT devices, the proposed honeypot system offers a novel approach to collecting IoT-specific malware samples from malicious actors by utilising sandbox environments running embedded software. This proved considerably more effective non-IoT honeypot systems, such as Honeyd [32], as the virtual sandboxes could be expanded to support various IoT architectures, including ARM, MIPS, and PPC. Additionally, IoTPOT logs all telnet-based intrusion attempts made by visiting IP addresses, allowing for the extraction and collection of malware samples. However, this implementation yielded a potential risk of honeypot instances becoming infected following successful intrusion attempts, in which case the sandbox environments would have to be completely reset. Some preventive measures were put in-place to attenuate potential side-effects of infected hosts: outgoing Denial of Service (DoS) traffic was actively blocked, and all other traffic was rate-limited by the host network. Following successful infection, IoTPot actively redirects all port-scanning activity to a dummy server, enabling the monitoring and observation of botnet propagation activity.

Bastos et al. [33] build upon the concept of preemptive botnet detection through purposeful execution of malware binaries by collecting samples via 47 low-interactivity honeypot servers distributed across 15 Brazilian states. Similarly to Moon et al. [30], this implementation infers candidate C2 servers of Bashlite and Mirai botnets using four heuristics based on a combination of static and dynamic analysis techniques. Such heuristics include analysing hard-coded IP addresses in binary strings, contacted DNS names, payload server URLs, and port-specific connections uncommon to the scanning process. Once identified, candidate C2 servers are validated by executing variant-specific handshake procedures between botnets and infected hosts within the IoT sandboxing environment *Detux Sandbox* [34]. This environment supports binaries compiled to run on common IoT device architectures, including ARM, MIPS and x86. Binaries were executed for only 90 seconds, and common scanning ports such as the telnet ports 23 and 2323 were blocked to prevent significantly malicious activity from occurring during analysis. This system successfully identified and validated C2 servers for 62% of binaries. Ceron et al. [35] builds upon the framework suggested by Bastos et al. [33] by focusing on improvements towards the dynamic and network analysis procedures previously suggested. It is noted that blocking specific ports could potentially lead to valuable network traffic from being blocked, assuming that some C2 servers communicate over commonly used ports such as 23 or 2323. Alternatively, Ceron et al. [35] suggest using an adaptive network layer, similar to that of the strategies employed by Pa Pa et al. [31], that monitors, redirects and drops malicious packets. Redirecting packets through a Software Defined Network (SDN) ensures that no traffic data is lost due to the blocking of

ports. Additionally, all outgoing traffic following a successful handshake with a C2 server can be redirected to an impersonator C2 server for improved behavioural analysis of botnet activity.

Alternatively, Dwyer et al. [36] propose the identification of IoT botnets through DNS-based analysis using machine learning. The data-set used for evaluation was collected using a distributed honeypot collection system capable of detecting Mirai-like TCP sequences. Mirai-like activity detection was implemented by comparing the TCP sequence fingerprint of an incoming packet with the IP address that was being targeted. If the sequence matches the IP address, the packet is deemed to be 'Mirai-like'. By distributing honeypots across the United States, Russia, and Brazil, the honeypot system collected a total of over 800,000 Mirai-like probes over approximately two years, 25% of which were successfully reverse-queried to obtain the DNS names of all logged IP addresses. The DNS records collective were tested against a set of legitimate DNS records collected from the Majestic Million Rankings list. A total of 6 features were extracted from each DNS record, including the Time To Live of the DNS record, the diversity of Autonomous Systems across all IP addresses associated with a DNS record, the Spatial distribution of IP addresses, the vowel density the domain name, and the Shannon entropy (randomness) of the domain name. Dwyer et al. found that the vowel density and Sharron entropy features gave the best results, yielding a detection accuracy rate of over 99% when using the Random Forest classification model.

To identify de-centralised, Peer-to-Peer IoT botnets, Wang and Chen [37] propose a detection model for identifying malicious data-streams amongst legitimate P2P traffic. Similar to previous propositions, the detection system actively identifies peer-to-peer botnets within the botnet propagation stage, such that botnet activity can be identified and monitored before attacks occur. The procedure involves first filtering out legitimate P2P traffic using characteristics such as packet size and monitoring the flow of packets between nodes of a peer-to-peer network in real-time. Li and Xue [38] builds upon the concept of data-flow analysis for P2P botnet detection by proposing a detection system using distributed intelligence sharing. Potentially malicious P2P traffic is collected from distributed communal clusters, which have raised potential threat alerts with the central hub. Upon receiving the network traffic flows of possible bot devices from community-based clusters, these flows are analysed using an algorithm based on the hidden Markov model.

Alternatively, Herwig et al. [26] propose that a similar honeypot detection system similar to that of [33, 35] can be applied to Hajime botnets. Hajime botnet propagation and attack activity can be actively monitored through performing handshakes with Hajime botnet nodes through performing handshakes with infected P2P nodes. Once an authenticated handshake is established, Herwig et al. [26] could gain access to and actively monitor the Distributed Hash Tables of several Hajime botnets, logging a total of over 10 million public keys. This is accomplished through computing the same identifier keys as identified bots during malware analysis and exhaustively looking up the keys in the Distributed Hash Tables commonly used by Hajime. However, as this method focuses on Distributed Hash Table patterns and handshake procedures unique to the Hajime botnet, the proposed strategy cannot be directly applied to other P2P IoT botnets, such as Mozi. As identified by Netlab 360 [27], the Mozi botnet uses it's own Distributed Hash Table protocol and has an extensive signature validation process strictly enforced between all botnet nodes. If other nodes cannot validate a Mozi node's signature, it will be excluded entirely from the botnet. The use of a valid signature may allow access to the Mozi DHT, though, Mozi botnet activity can still be identified from within a host system by analysing system processes and monitoring HTTP and DHT network connections.

3 Problem Analysis

The concept of preemptive botnet detection proposed by Moon et al. [30] proved significant in detecting botnets, as the purposeful execution and analysis of malware enabled the collection and analysis of information crucial to the identification and mitigation of botnets during propagation periods. This system allows for the automatic collection and analysis of binaries using two dedicated processing modules. However, several issues are presented within this proposed botnet detection framework. A primary concern is that the data collection methods employed by Moon et al. [30] 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.

Similarly, 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 [32], have proven ineffective in identifying and monetising from IoT botnet traffic. The honeypot systems proposed by [31] and [25] can identify such 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 distribution of honeypots. The implementation proposed by [31] 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, this system required frequent resets due to successful intrusions and only focused on monitoring telnet-based honeypot intrusions. By exclusively monitoring telnet traffic, this system overlooked important information provided by analysing other commonly used botnet communication protocols, including SSH, HTTP, P2P, and IRC traffic. As botnets frequently operate across various communication protocols, the exclusion of such protocols may lose important information valuable towards botnet identification.

Bastos et al. [33] and Ceron et al. [35] further build upon the concept of IoT honeypots and the execution and examination of malware within sandbox environments using a combination of static, dynamic, and heuristic analysis techniques. Both papers propose the separation of botnet data collection and analysis through extracting malware binaries and evaluating in/outgoing SSH and Telnet traffic within secure sandbox environments via the Detux [34] Linux sandbox. Despite supporting the emulation of up to 5 CPU architectures, the Detux sandbox is limited in functionality and only allows for basic static and dynamic analysis. Detux also presents limitations for real-time malware analysis. It cannot be easily scaled or parallelised like that of competitor malware sandboxes, hindering the possibility of real-time static and dynamic malware analysis of honeypot traffic. Additionally, 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.

Dwyer et al. [36] employs a solution that allows for accurate botnet detection without the requirement of malware execution and analysis of malware samples. Globally distributed across three continents, a honeypot system collected the IP addresses of all detected traffic emitting Mirai-like scanning activity. The host-names and DNS records of a sample of IP addresses were further collected and analysed by applying machine-learning classification models to various extracted features. This approach gave high botnet detection accuracy for Mirai botnets; however, as with most other centralised botnet detection systems, this solution cannot be applied to that of 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. The adopting Peer-to-Peer ifrastructure mitigates the use of centralised Command & Control servers altogether, thereby lessening the incentive to employ DNS-based communication tactics common to centralised botnets.

The findings presented by Herwig et al. [26] illustrate that new de-centralised botnets are coordinated through interaction with de-centralised Peer-to-Peer Distributed Hash Tables (DHT). The various forms of shell-injection exploits commonly seen originating from propagating Hajime botnets are discussed, highlighting how shell commands such as `wget` and `tftp` are used to download and execute binaries. However, little detail is given regarding how payloads, obfuscated through command-line arguments and other methods, are extracted before sample analysis. As many malware samples are likely obfuscated using command-line tool arguments, such samples must be extracted to ensure botnet identification is maximised. It is proposed that the public keys used by Hajime botnets can be used as unique identifiers when monitoring botnet propagation, as IP addresses frequently change ownership. However, the context of IP address ownership may be relevant to botnet identification, as patterns can be identified to indicate prominent hosting services and Autonomous Systems that are affiliated with botnet activity.

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.

Given the limitations of previous botnet identification strategies, I propose VisiBot, 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. Through the synchronous processing of malicious traffic streams detected by a widely distributed IoT honeypot system and the combined use of various heterogeneous data-sources, VisiBot allows for real-time analysis of botnet propagation, geographic distribution, and density. By applying four heuristics based on the combined findings of Bastos et al. [33] and Herwig et al. [26], the automated processing system employed by VisiBot constantly analyses and extracts knowledge from incoming botnet traffic, collected through a combination of static, dynamic and network-based analysis techniques. The resulting data collected during this process permits the identification and visualisation of presently propagating centralised and Peer-to-Peer IoT botnets.

4 Design Overview

This chapter describes a high-level design overview of the VisiBot Processing System and Interactive Web Application, highlighting the various design patterns, principles, and methods applied.

4.1 The VisiBot Processing System

4.1.1 Honeypot Integration

The VisiBoT Processing System employs a wide 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 central honeypot processing server analyses and classifies packets based on characteristics observed across various Common Vulnerability Exploits (CVE). 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.

4.1.2 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" [39], 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 completely 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 main 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 source IP of the packet, 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. The IP corresponding to the extracted URL is therefore 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 the sniffing of 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. For example, the interaction between a Report server and Payload server is stored as a directional graph where each IP address is represented as a node.

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:

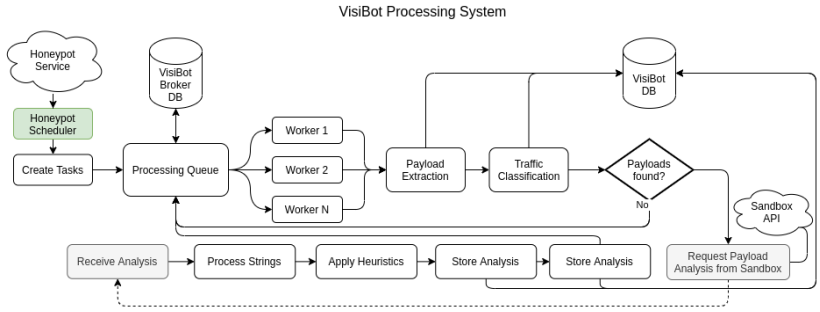


Figure 4.1: *High-level flow-diagram of the core VisiBot Processing System.*

4.1.3 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, and an analysis task will be 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 generate 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.

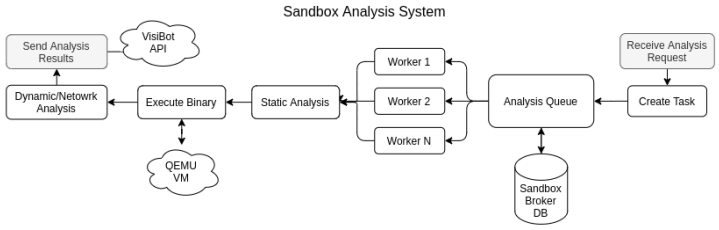


Figure 4.2: *High-level flow-diagram of malware sandbox analysis system.*

4.1.4 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. The first heuristic is concerned with identifying Peer-to-Peer botnet activity through analysis of 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 hard-coded IP addresses extracted during static analysis are later connected to during network analysis, the IP address is associated with a 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 full high-level overview of the VisiBot Processing System, including remote sandbox integration, see Appendix A.1.

4.2 Interactive Web Application

To allow for real-time visualisation of the botnets detected by the VisiBot Processing System, 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:

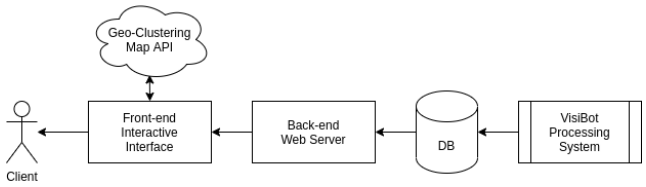


Figure 4.3: *High-level overview of the VisiBot Web Interface.*

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:

```
foreach 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 4.1: *Pseudocode for Greedy clustering algorithm based on an example by Agafonkin [40]*

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.

5 Implementation

This chapter discusses the various implementation stages involved during 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.

5.1 Distributed Real-time Processing System

5.1.1 Message-based task queues with Celery and Redis

Written in Python 3.9 [41], 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 [42], 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 [43], 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 can be configured automatically based on the queue's size and the work-load. Celery workers also support the execution of tasks in parallel via multi-processing, enabling the swift and concurrent performance of time-consuming tasks that might otherwise bottle-neck synchronous applications. The VisiBot Processing System utilises Celery workers to analyse, identify, and extract legitimate botnet traffic from honeypot sources and sandboxes. 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.

To monitor the VisiBot task queue, Flower, a python-based Celery monitoring tool by Movsisyan and contributors [44], was used. Accessible remotely through a web-application, Flower allows for 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:

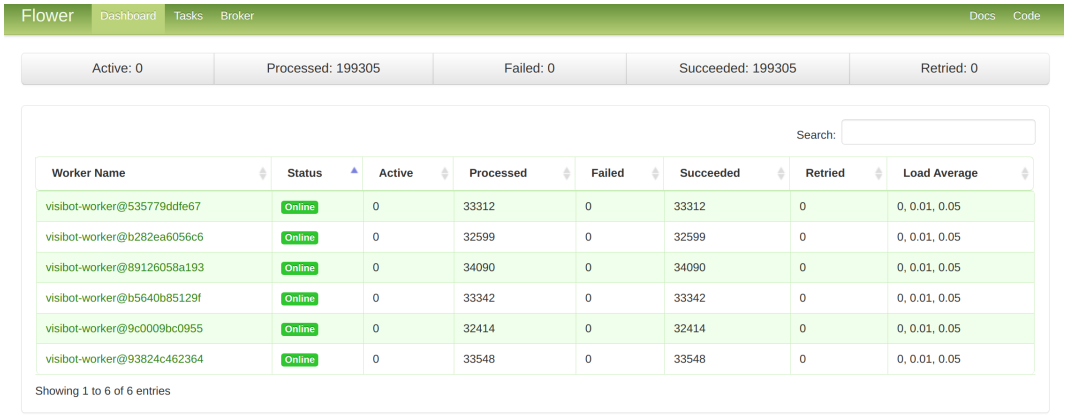


Figure 5.1: *Flower Dashboard. Celery workers are represented in a table as rows.*

5.1.2 Docker

As the VisiBot task queuing system and analysis process utilises various applications, celery workers, servers, and external services, the setup and deployment procedure of such a system can introduce a multitude of complexities and anomalies due to varying environmental factors such as operating system, package managers, system architecture, etc. Thus, to streamline the setup and deployment process, all applications, services, and workers utilised by the system have been encapsulated within a multi-container Docker application using the command-line tool Docker Compose [45].

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 the docker application. 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 Application, MaxMind GeoIP2 Updater, and a remotely accessible instance of Flower. Docker enables shared access storage between 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 various aspects of the processing system, such as API keys and paths, without modifying any code directly.

As VisiBot workers use shared MaxMind GeoIP2 databases [46] 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` [47] 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:

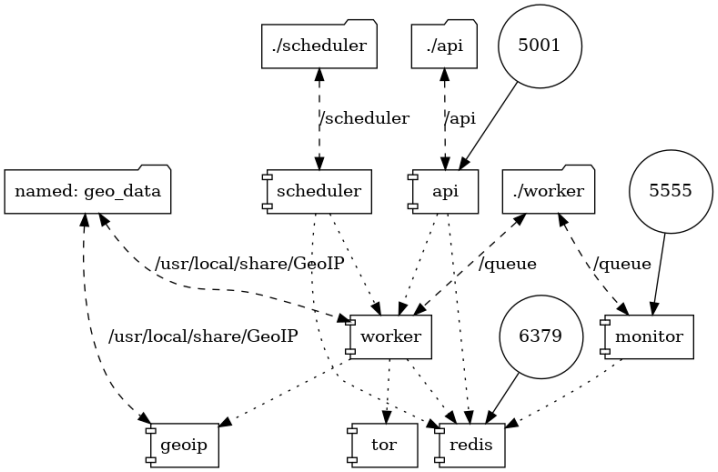


Figure 5.2: *Visualisation of VisiBot Docker-Compose Containers. Reverse Engineered using docker-compose-viz. Containers are represented as rectangles and exposed container ports as circles.*

5.2 VisiBot Processing System

5.2.1 Database Record Storage using MongoDB

Developed by MongoDB Inc. [48], 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, large, and nested data objects through MongoEngine, an open-source python object data mapper for MongoDB [49]. As Documents are represented in a standard JSON format, such documents 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, certain relationships such as One-to-One and One-to-Many can be represented by 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 full Entity Relationship diagram of all of the VisiBot Processing database relationships can be seen 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 [50].

5.2.2 Honeypot Data Collection

All honeypot information processed by VisiBot is provided by Bad Packets LLC [51], 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 [52], 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. For an example of a Bad Packets honeypot result, see 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. The scheduler begins to create analysis tasks 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 5.1: *Example output of VisiBot Processing Scheduler.*

Upon receiving tasks from the Celery broker, workers will begin to perform analysis on 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.

5.2.3 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 will likely attempt 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 a packet. By extracting the URL of the botnet malware payload, the binary can later be retrieved and analysed.

Noise removal and de-obfuscation The first stage of malware payload extraction involves the removal of noise and mitigation of the various obfuscation techniques employed by bad actors, such as botnet owners. 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. 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 5.2: *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, as the `${IFS}` variable is used as an input field separator in-place of space characters. The second payload contains some light obfuscation employed to prevent regular expression patterns from matching URLs in the payload string. The first payload URL downloaded using `wget` has an escaped forward slash such that the pattern `http://` cannot be matched. Should the infiltrated system not have the `wget` command, the injected code also tries to download the binary using the `tftp` client. However, the malware payload URL is obscured using various command-line arguments, as the host of the URL is specified using the `-g` flag, the path using the `-r` flag, and the output using `-l`. These URLs can be re-built, but the presence of noise such as `${IFS}`-based spacing can make this process difficult. Through the manual inspection of several payload strings and post_data contents, a short-list of noisy characters and sub-strings was created. This list is used during the cleaning process, such that significant noise is removed 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 the limitations of Regular Expression matching, the extraction process also utilises other methods of URL extraction. The python library `urlextract` [53] 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. Domain Name Systems (DNS) are frequently used by bad actors such that IP addresses can be quickly changed without impacting communication between botnets and C2 servers. This method allows for the extraction and validation of domain names used for botnet propagation

by bad actors, such as botnet owners, as partial domain-based URL strings are identified and re-formatted using the standard HTTP URL format. As these strings do not indicate an HTTP schema, the `http://` schema is used by default.

The practice of obfuscating payload URLs through executable arguments was quite common across inspected honeypot payload command-line injection strings, as shown in the example sub-strings above. However, the VisiBot Processing System also 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 executed, parsing the arguments following the initial command name 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 the base URL extraction process, 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, the VisiBot Processing System's current limitation is that bash scripts cannot be executed from within the chosen sandbox environment. To combat this constraint, the worker will parse the contents of any bash script URLs and extract and store any nested URLs, which lead to binary executables using the same process as the payload string URL extraction stage. As bash scripts often include links malware for various types of architectures, this can lead to the extraction of up to 6 (or more) binaries. 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 [54]. 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` [55, 56], 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 builtin python `socket` library. [57] 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, it's 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.

5.2.4 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 was 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, used as a download host when attempting to infect other machines.

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, suppose no URLs are extracted from the honeypot packet payload or post_data strings. In that case, 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. [58]

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

Figure 5.3: *Mirai TCP sequence source-code. Snippet provided by Bad Packets LLC [51]*

Using the additional tag information provided by Bad Packets, VisiBot can infer if a packet is associated with botnet scanning activity. A honeypot packet that does not contain any payload URLs and indicates either Mirai-like or otherwise general port scanning activity is classified as a "Bot".

5.2.5 Honeypot Packet Processing

Following the identification and classification of the source IP and payload IP addresses of a given honeypot packet, each IP address is parsed for additional meta-data required for botnet analysis. The geographic information for each classified IP address is looked up Using various MaxMind GeoIP2 databases [46] and is stored within the VisiBot database. Such geographic metadata 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 identified using the IPInfo Privacy Detection API [59], 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 for each IP address using the `ipwhois` python module, an IP WHOIS lookup tool that sources ASN information directly from Internet Service Registries such as AFRINIC, APNIC, and RIPE NCC [60, 61, 62].

Following the IP address meta-data storage, a new event log entry is created for each IP address, indicating 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, specifying the source and destination IP addresses of IP address relationships and the time at which the connection/relationship between IP addresses was logged. Lastly, the packet JSON data sourced from the Bad Packets honeypot service is stored directly within the VisiBot database. Each record contains valuable information for botnet analysis, such as the targeted port, request user-agent, Bad Packets tags, and event count. The entity-relationship structure of all documents stored within the VisiBot database is shown in Appendix A.2.

Once complete, the VisiBot worker will iterate over the list of extracted payloads URLs and sequentially send HTTP requests to a remotely hosted malware sandbox server for analysis, along with a specified execution time in seconds. In this case, the worker specifies that the binary must be executed for a total of 30 seconds before termination. Following the creation of a malware analysis task, the sandbox server will respond with a unique `task_id` identifier, which is stored within the VisiBot database along-side the identifier of the payload being analysed. Upon malware analysis completion, the sandboxing server will send an analysis report back to the VisiBot Processing System through a REST API. The received report is then analysed for candidate botnet Command & Control server and Peer-to-Peer node identification. For a full low-level overview of the VisiBot Processing System, please refer to the flow diagram shown in Appendix A.5.

5.2.6 LiSa Sandbox Integration

Developed by Daniel Uhříček, the LiSa Sandbox [10] is an open-source, scalable Linux Sandbox platform for executing and analysing malware binaries of various architectures, including CPU architectures such as MIPS, ARM, x86_64, i386, and aarch64. Like the VisiBot Processing System, LiSa is a multi-container docker application that invokes the message-broker design pattern using Celery and RabbitMQ as message broker [42, 63]. Accessible through both a web dashboard and REST API, the LiSa sandbox allows for remote static and dynamic analysis of binary executables. 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 a scaled and easily distributable system for performing remote malware analysis, enabling multiple binaries to be processed at any given time.

Static analysis information is directly extracted from uploaded binaries using Radare2 [64], 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 using an embedded Linux image that matches the binary CPU architecture. As LiSa supports five different architectures, an embedded Linux image was created for each architecture using Buildroot [65]. 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 [66]. As the VirusTotal scan reports of identified malware binaries are actively used throughout VisiBot analysis, this service was enabled 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 of the LiSa Sandbox [67] to mitigate such issues.

The first issue encountered was that binaries had to be provided as file attachments when creating analysis requests through the LiSa REST API. The LiSa sandbox API was modified to accept an additional `url` parameter the URL of a binary to be provided instead of the file itself, as downloading and storing malware binaries onto the VisiBot server may result in potential security issues.

Following the manual inspection of several LiSa analysis reports, the malware reports indicated that many malware binaries were obfuscated using packing software to prevent the extraction of strings. Any attempt to extract strings from a packed malware binary would result in un-readable and heavily obfuscated output. Commonly incorporated by malware authors as a means of obfuscation, a packed binary contains a payload of compressed or encrypted code unpacked into the program's address space at run-time. [68] This process ensures that analytical software tools, such as the `strings` command-line tool [69] used by LiSa, output obfuscated information. Modifications were made to the LiSa sandbox to allow for the detection and unpacking of malware packed using the UPX packing tool [70]. UPX a popular tool used by botnet perpetrators 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 query the LiSa API for task status updates repeatedly, yet this proved highly inefficient and ultimately introduced several reliability issues. Thus, the VisiBot Processing System hosts a minimal REST API written in Flask, [71] 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 endpoints */api/lisa-analysis/success/<task_id>* and */api/lisa-analysis/failure/<task_id>* respectively. Additional modifications were made to the LiSa sandbox source-code to send analysis or failure information 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) can be removed from the MongoDB database.

5.2.7 Malware Analysis Information Extraction

Before applying identification heuristics, the worker will first attempt to parse the top-most recurring malware keyword from a list of anti-virus scan results obtained from VirusTotal. However, as many anti-virus services do not follow the same naming convention when labelling the type of malware detected, additional steps were taken to standardise the keyword extraction process by removing punctuation, case sensitivity, and generic keywords such as `Trojan`, `Malware`, `Worm`, etc. What is left is a list of less common keywords for each malware analysis result. These lists are combined, and the most commonly recurring string is used as the keyword

identifier for the analysed malware. For an example of this process, please refer to Appendix A.6.

Following this, the worker will parse each of the binary strings collected during the static analysis stage of the LiSa sandbox analysis, such that all strings of significance are extracted. Such strings of significance may match regular expression patterns for IPv4 and IPv6 addresses, URLs, and domains. This step was necessary, as a full list of strings extracted by LiSa is often too large to store within a database. Additionally, the extraction of such features allows for ease-of-use in future research.

5.2.8 Application of heuristics

After processing, the next stage of the LiSa analysis worker task is to identify all candidate Command & Control Servers and Peer-to-Peer botnet nodes associated with a given binary by applying four identification heuristics on the provided network analysis information logged during malware execution. The four heuristics used by VisiBot 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 between the infected host and hard-coded IP address in malware binary.
iv Interaction between the infected host and blacklisted Command & Control Server IP address.

Heuristic (i) The first heuristic is primarily used to differentiate between centralised and de-centralised Peer-to-Peer botnets. By analysing the DNS queries made by a given malware sample during sandbox analysis, we can infer that the malware communicates through Peer-to-Peer networks if any of the given domains fall within a list of known bit-torrent services. As de-centralised botnets often use distributed hash-tables as a means of communicating with other bots and the botnet owner from within a peer-to-peer network, a device infected with peer-to-peer botnet malware may attempt to look up a distributed hash table using a common bit-torrent service. For example, a botnet may query the domain **dht**.bittorrent.com as part of the distributed hash-table lookup process. Thus, the VisiBot Processing system identifies if this heuristic is satisfied by checking if any of the DNS queries invoked during malware analysis point to a short-list of domain names for known Peer-to-Peer services. The VisiBot Processing System matches all DNS queries collected from the network traffic a given binary with the below string-patterns:

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

If any of the above domain patterns are matched, we deduce that the heuristic *i)* is satisfied. However, as this heuristic only serves as an indication of 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)*, such that botnet activity can be identified.

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 significant network traffic, as IP addresses that send and receive information to/from an infected host will satisfy this heuristic. If both heuristics *i)* and *ii)* are satisfied, it is deduced 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 LiSa analysis process's previous stage. 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 logged during the network analysis is blacklisted as a known Command & Control server, the heuristic *iv)* is satisfied. The VisiBot Processing system checks the blacklist status of each IP address endpoint logged during malware analysis using various blacklist services, including those provided by Barracuda Central, abuse.ch, SpamRats, Sorbs DNSBL, and Spamhaus. [72, 73, 74, 75, 76] If any IP address has previously been blacklisted as a C2, the IP address is classified as a candidate Command & Control server.

Following the above application of heuristics, each IP address endpoint logged during network analysis will be classified either as a candidate C2, a Peer-to-Peer Node, or as benign traffic, which is ignored. A new C2 or P2P entry is added to the VisiBot database depending on which heuristics are satisfied. Followed this, a variety of additional meta-data is collected from the given IP address and stored within the VisiBot database, including geographic information, ASN history, IPinfo Privacy information, and a new event log entry. Following meta-data collection, connection records are created between the analysed payload's IP address and all malicious C2/P2P IP addresses identified during analysis. Lastly, the LiSa analysis report is refined and stored within the VisiBot database and all significant strings and the identified VirusTotal malware keyword. Once the analysis is complete, the worker will consume another LiSa analysis task from the VisiBot broker and repeat the above steps.

5.3 Interactive Web Visualisation

A significant component of the VisiBot Processing System is that the information collected can be easily visualised. A map-based web application was 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. The development of the front-end presentation layer and back-end data interaction layer are separate, allowing for a separation of concerns. The model used is the same MongoDB database [48] 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 [77] JavaScript framework. Express.js [78] is a flexible web application framework written in Node.js [79], often used as a controller within the MVC design pattern.

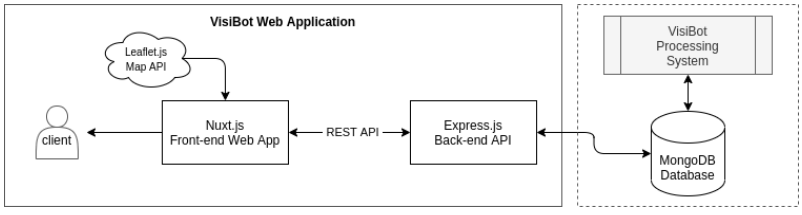


Figure 5.4: *Server Architecture Diagram: VisiBot Web Application*

The back-end Exprss.js REST API communicates with the VisiBot MongoDB database using Mongoose, an Object Data Modelling (ODM) library for MongoDB. [80] 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 BootstrapVue [81] library was used with Nuxt.js to develop a modern, flexible, and intuitive user interface using the Bootstrap CSS framework.

5.3.1 Geographic Clustering and Networks

An interactive, cluster-based map was developed using [82] 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 5.5: *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 [83] 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.

5.4 Software Development Practices

Issue Tracking To manage the many deadlines, priorities, tasks, and to-do-lists involved throughout the development of VisiBot, Trello, a Kanban-style, web-based, ticket tracking application [84] was used. Throughout the development of VisiBot, various issues were created and managed using Trello. Using Trello, I sorted all tickets related to the VisiBot development process between three lists based on development status: 'Backlog', 'In Progress', and 'Complete'. All tickets awaiting development are stored in a ticket backlog, with some tickets also being organised based on feature priority. The use of a board-based ticket management system ensures that all priorities and features are displayed coherently. More-so, Trello also proved highly beneficial for tracking and managing various project-related deadlines and to-do-lists. Each ticket can be assigned a due-date timestamp, encapsulate lists of sub-tasks using interactive checklist boxes.

Quality Assurance Throughout development, linting tools and unit testing tools were used as a means of quality assurance. Several unit tests were written using the unittest [85] python testing framework, ensuring reliable and robust code. Pylama, an auditing/linting tool for Python, [86] 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 best practices followed by Python programming professionals, including official style guides, such as the PEP-8 style guide written by van Rossum et al. [87]. Additionally, ESLint, [88] a static analysis linting tool for JavaScript, was also used for actively identifying problems in both Nuxt.js and Express.js applications, identifying issues 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 arise from inconsistent or poorly written code.

Source Code Management Git [89] was used throughout development for managing and versioning the VisiBot source code throughout the project development life-cycle. Git is a Source Code Management (SCM) tool that allows for versioning control amongst various other beneficial features. All source-code managed using the Git SCM was also remotely hosted on a private repository using the GitHub [90] code-hosting and collaboration platform. Throughout development, I 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 only one developer carried out the development of VisiBot, branching strategies such as feature branching were deemed unnecessary, as the likelihood of encountering merge conflicts within a single-developer work environment was minimal. Before pushing any local changes to the remote repository, all code was automatically tested locally using Git pre-push hooks. [91] By hooking a shell script onto the `git push` command, the contents of the script is executed and evaluated before the push request being executed. 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 [92] Continuous Integration service. Integratable with GitHub, Travis CI was used throughout development to automate the linting and unittesting quality assurance measurements whenever code is pushed to a monitored GitHub branch, such as the master branch. If the pipeline fails, all maintainers of the branch 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.

6 Results and Evaluation

6.1 Data Collection

The VisiBot Processing System was deployed over five weeks from 11/02/2021 to 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 several of these honeypot servers mimic characteristics of commonly targeted IoT devices, this ensures that several types of IoT-based botnet interactions will be caught. The LiSa Sandbox was hosted on a separate server from the VisiBot system to ensure a separation of concerns. All objects generated during malware analysis, such as malware samples, program output logs, and packet capture (pcap) files, are stored on a block-storage partition. Separation ensures that if the sandbox were to become compromised, all the virtual private server could be reset accordingly without the loss of valuable information. All binaries captured by VisiBot were executed within a LiSa sandbox for a total of 30 seconds.

The distributed processing and analysis of 61,293 honeypot packets lead to the detection 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 the analysis of all extracted binaries through VirusTotal, a total of 62,203 out of 97,659 anti-virus vendor scans were detected positive for malware. Through the combination of heuristic-based malware analysis, payload extraction, and the detection of Mirai-like characteristics of honeypot packets, the VisiBot Processing System was successfully able to classify botnet traffic into six categories, as shown below:

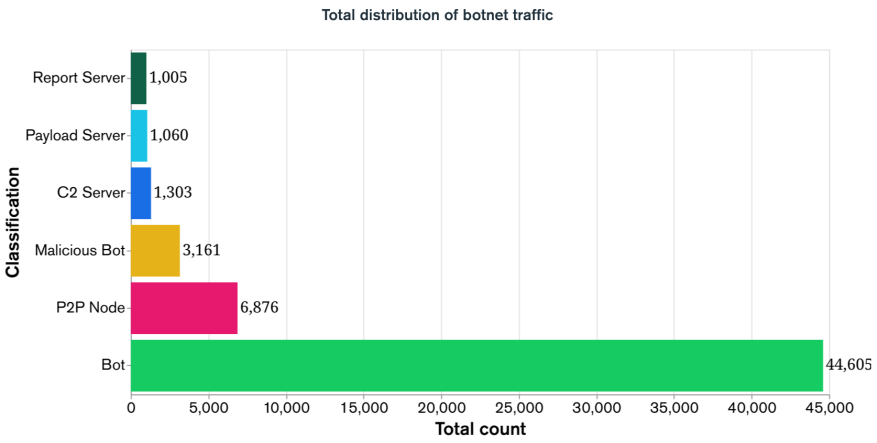


Figure 6.1: *distribution of classified botnet traffic.*

The above histogram illustrates that a significant proportion of botnet traffic is that of Mirai-like port-scanning activity. However, a large proportion of botnet traffic also originates from several Peer-to-Peer networks logged during malware analysis of collected binaries. Notably, considerably more malware-hosting Malicious Bots have been observed than that of Report Servers and Payload Servers, which are common to centralised Mirai botnets. The higher number of observed malicious bots may allude to a potential change in contemporary botnet propagation strategies, as botnet developers often employ new de-centralised tactics to mitigate centralised communication between entities such as Report, Payload Command & Control servers.

6.2 Geographic Botnet Distribution

The 5-week period of automated botnet traffic collection through strategically distributed honeypots yielded a wide collection of geographic information, including the logging of cities, countries, autonomous systems, and geographic coordinates of all IP addresses detected by the processing system. VisiBot successfully processed and classified a magnitude of honeypot information, encountering IP activity spanning across several continents, including Europe, North America, South America, and Asia. Such geographic information can be analysed in a manner of ways to gain a better understanding of how IoT botnets propagate, which countries are specifically being targeted, and how botnets are globally distributed. 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:

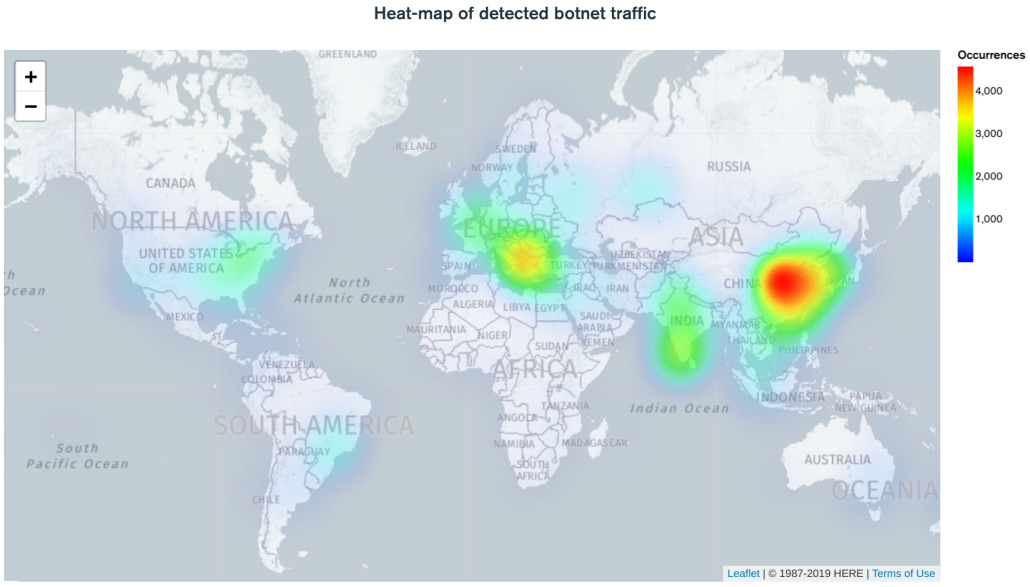
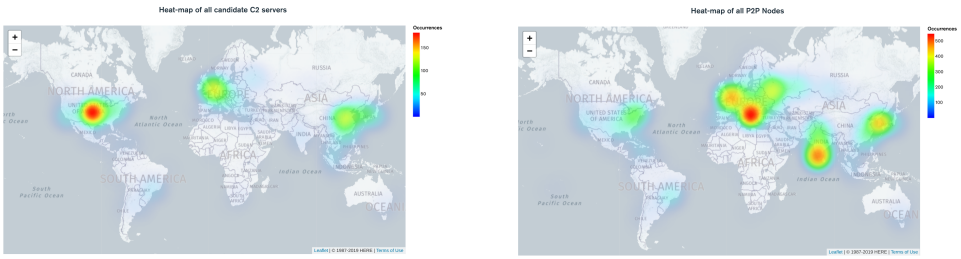


Figure 6.2: *Geographic distribution of all botnet traffic detected by the VisiBot Processing System.*

As shown above, there is a significant traffic density originating from Asia. More specifically, a lot of the Mirai-like port-scanning activity noted in Figure 6.1 appears to be originating from China, one of the leading manufacturers of IoT devices. Due to the high-scale production and low-quality control of IoT devices originating from China, such devices may still be prone to known vulnerabilities and exploits. Similar but less dense geographic patterns are also observed across India, Europe, North America, and Brazil. However, a significant proportion of all

botnet traffic logged by VisiBot is originating from China. When mapping the geographic distribution of identified Command & Control candidates and Peer-to-Peer nodes, the traffic density differs significantly from overall traffic distribution. Consider the below side-by-side heat-map visualisations of identified candidate C2 servers and P2P nodes identified by VisiBot:



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

Figure 6.3: *Malware sample architecture and endianness represented side by side.*

The above heat-map visualisation of candidate C2 server distribution shows a high density of C2 activity across North America, with some occurrences spanning across Europe and China. There is a clear contrast in traffic distribution when comparing the areas of high geographic C2 density with all botnet traffic distribution. 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:

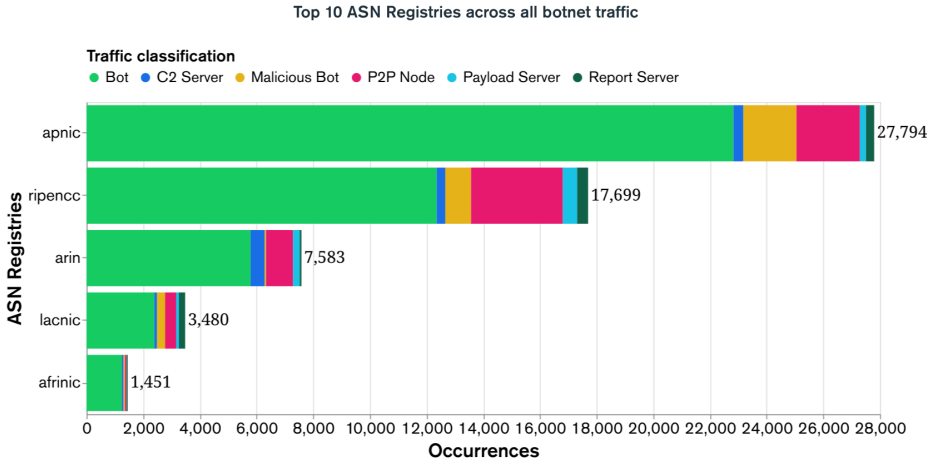


Figure 6.4: *distribution of classified botnet traffic across ASN registries.*

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:

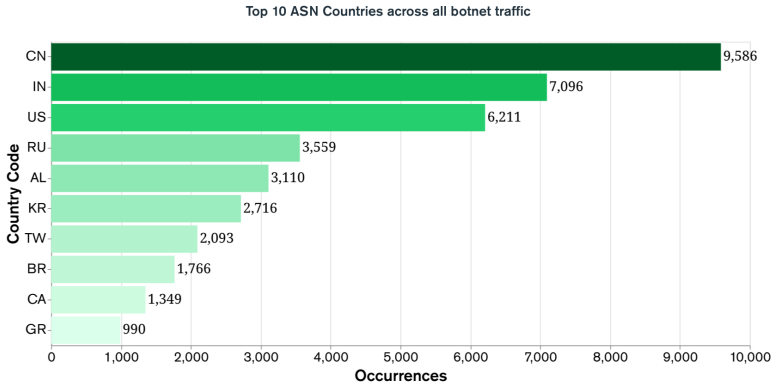
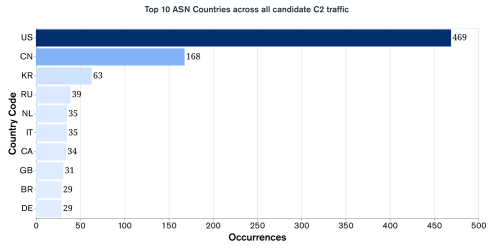
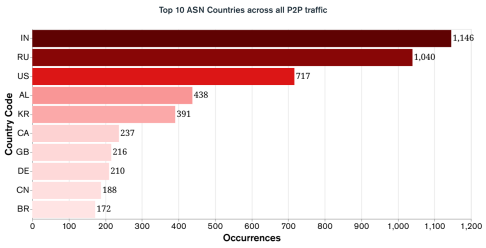


Figure 6.5: *Top 10 ASN country codes for overall botnet traffic.*

As inferred in Figure 6.2, 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 6.6: *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.

6.3 Traffic Analysis

Despite a large proportion of benign botnet activity shown in Figure 6.1, 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. [51] The below table represents the top 10 most commonly observed botnet packet categories and CVEs across all traffic classified by the VisiBot Processing System:

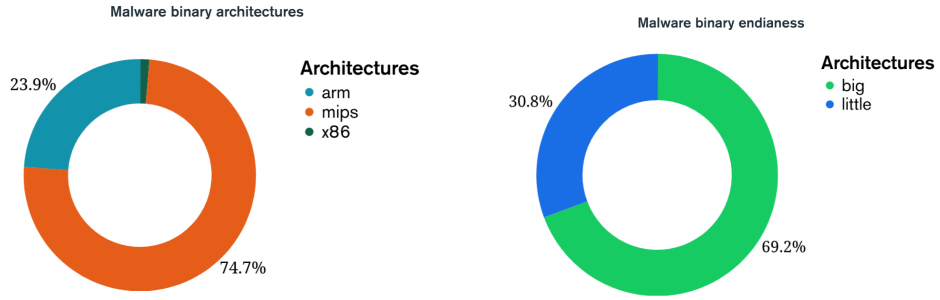
Table 6.1: *Top 10 CVEs and packet categories across all botnet traffic*

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 default login credential brute-force 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, it was 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. This ensures that botnets can propagate through various CPU architectures, such as those listed in the table above.

6.4 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 6.7, 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 6.7: *Malware sample architecture and endianess represented side by side.*

Given these findings, it is likely that the types of botnets identified by the VisiBot Processing System 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 can be extracted. The below table shows the most commonly occurring VirusTotal keywords across all analysed malware binaries:

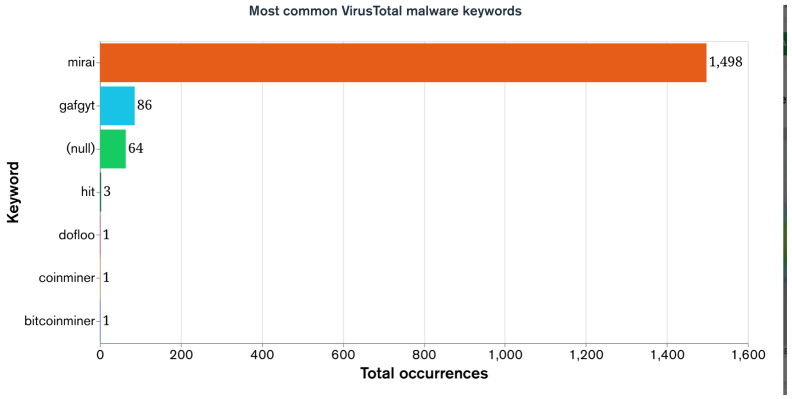


Figure 6.8: *distribution of malware samples based on VirusTotal keywords*

Although the vast majority of analysed binaries were detected as Mirai malware variants by the VirusTotal anti-virus vendors, some were also identified as Bashlite (gafgyt) and coin-miners. The "hit" keyword associated with three binaries was extracted from the anti-virus result string `Trojan.Linux.Agent.HIT`, revealing that despite testing positive, most VirusTotal anti-virus vendors were unable to detect the exact malware families of the binaries. Binaries that were not detected by the VirusTotal vendor scanning service were allocated the `(null)` placeholder keyword.

Netowrk 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 6.2: *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. [36] 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 [27]. 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 mostly 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. [93] Whereas, ports 67 and 1900 are typically used by DHCP servers and UPnP devices, as highlighted in Table 6.1. 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 web servers hosted at randomly selected UDP ports. The files hosted on the temporary web-server 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 web-server 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, but only a small percentage of binaries were actually 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 b 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, and a higher number of binaries can therefore be extracted.

P2P Payload Connections Analysis Generated using NetworkX [94] and the K-Components approximation algorithm [95], 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:

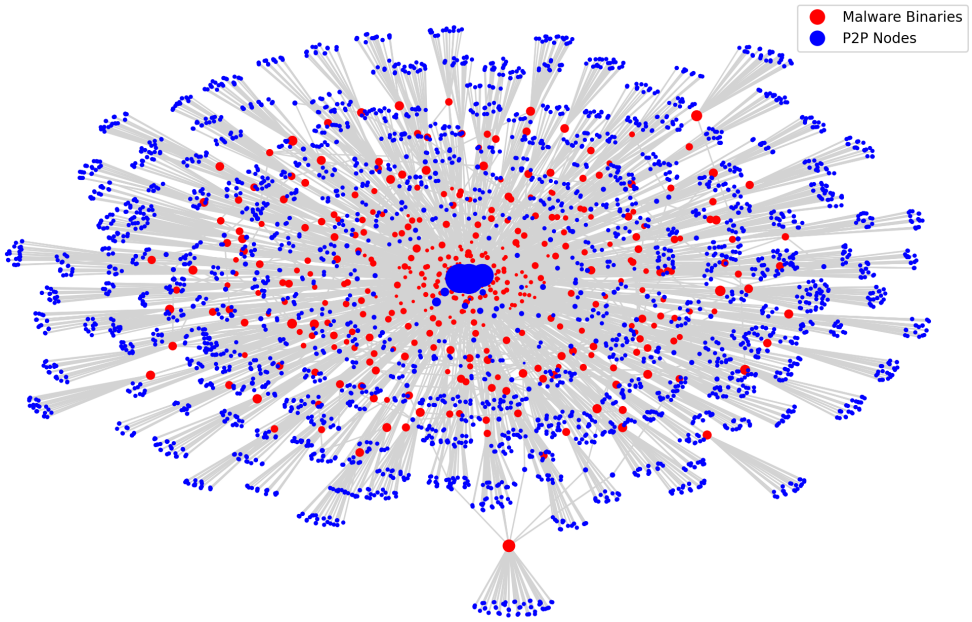


Figure 6.9: *Edge-connectivity graph of P2P Node and malware binary interactions recorded during network analysis. Edge connections approximated using K-Components algorithm.*

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. [26] A summary of the number of nodes associated with malware payloads is shown below:

Table 6.3: *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 can also be 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 [96], the below network graph illustrates the historical interactions, such as ownership or temporary holdings, between Autonomous Systems and identified candidate C2 servers:

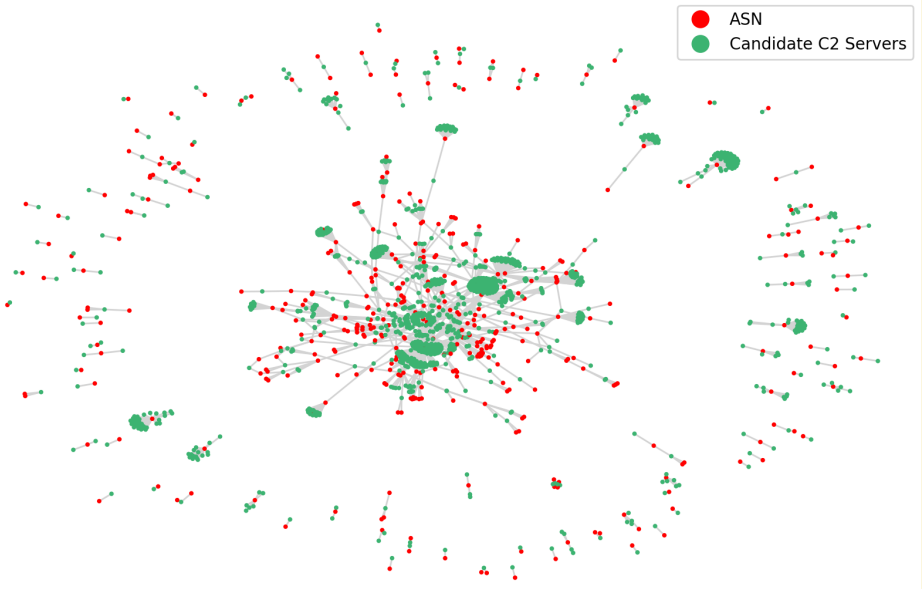


Figure 6.10: *Edge-connectivity graph of P2P Node and malware binary interactions recorded during network analysis. Edge connections approximated using K-Components algorithm.*

Table 6.4: *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 indicates that several ASNs are holding/exchanging ownership of IP addresses associated with Command & Control activity detected by VisiBot. Addition-

ally, 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.

6.5 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 a 35-day period. In comparison, the IoTPoT honeypot system proposed by Pa Pa et al. [31] 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 [51] 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, as a large number of malicious botnets has been observed throughout the experimentation period, a considerable proportion of the extracted malware URLs were no longer active by the malware analysis stage. This resulted in significantly fewer malware binaries being collected for analysis.

However, out of the 9,923 URLs identified, only 1654 binaries could be collected due to limitations presented by the honeypot system used by VisiBot. During the 35-day experimentation period, VisiBot encountered a significant number of malicious bots originating from the Mozi botnet variant, which actively perform remote code execution attacks using self-hosted malware samples. However, as the corresponding port of the web-server used for hosting malware changes to a new randomly selected port on a frequent basis, 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 for packets to be analysed within a short time-frame of being detected by a given honeypot. By doing so, a larger proportion of self-hosted payload URLs can be collected.

The implementation of 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. [33] 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 different as the system proposed by Bastos et al. [33] is exclusively concerned with the identification of Centralised Command & Control servers from the conventional IoT botnets, Mirai and Bashlite, and does not consider approaches for identifying de-centralised botnets. Whereas the VisiBot Processing System is primarily concerned with the identification of 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. As many de-centralised botnets communicate with legitimate public Peer-to-Peer networks, and centralised botnets are known to communicate with fake C2 servers, all entities identified through heuristic analysis must be further validated through the invocation of botnet handshake procedures. Due to time constraints, this functionality is not yet present in the VisiBot Processing System, yet, the malware analysis process can be conveniently extended 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 are effective in differentiating between benign and malicious botnet traffic. However, the lower detection rate indicates when compared to other systems indicates that some of the heuristics used were less effective than anticipated. During the data collection period, a high number of packed malware binaries were observed, mostly through the UPX packing tool [70]. Despite making modifications to the LiSa sandboxing server to enable the unpacking of UPX binaries, a large number of samples could not be successfully unpacked. This resulted in the collection of obfuscated binary strings during static analysis, which proved too noisy for the extraction of hard-coded IP addresses. Following this, the heuristic-based on 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 entirety of 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 have proven useful 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 are being employed by botnet developers. Graph-based visualisations of C2 and Autonomous System interactions also provide insight into the interplay between Autonomous Systems and botnet IP addresses. Similarly, the visualisation of Peer-to-Peer connections observed between malware payloads 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 the context of Cyber Threat Intelligence.

7 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.

When applied to static, dynamic and network analysis of collected malware binaries, the proposed heuristics permit the identification of centralised and de-centralised botnet traffic using automated and distributed techniques. Through the profiling of commonly observed characteristics amongst both centralised and de-centralised botnets, heuristic analysis allows for a computationally inexpensive solution for detecting potential Command & Control server and Peer-to-Peer botnet activity. The extraction capabilities of VisiBot have also shown to be effective in extracting payload URLs obfuscated within remote code execution attempts. However, as botnets evolve, so too will the obfuscation techniques employed. In its current configuration, the VisiBot Processing System proved capable of identifying and analysing traffic generated by various centralised and de-centralised botnet variants, including a recent IoT variant, the Mozi Peer-to-Peer botnet.

In summary, the VisiBot Processing System presents an effective solution for automated identification of centralised and de-centralised IoT botnets through real-time parallelised extraction, execution, and heuristic analysis of honeypot malware samples. In doing so, the VisiBot Processing System successfully classified over 58,010 unique IP addresses, allowing for the identification, examination, and visualisation of 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.

7.1 Future Work

Current capabilities of VisiBot allow for real-time candidate C2 server and Peer-to-Peer node detection. However, the proposed system does not validate C2 servers and P2P Nodes as existing VisiBot heuristics only identify characteristics that resemble C2 or Peer-to-Peer botnet traffic. Such candidates need to be validated to determine if they are genuinely malicious. Heuristic analysis carried out by VisiBot can be updated to include validation steps that perform various botnet handshakes on detected candidate C2 servers and Peer-to-Peer nodes to identify if a given endpoint is malicious. As described by Bastos et al. [33], IoT Command & Control servers for variants such as Mirai and Bashlite can be validated through reverse engineering a malware sample, analysing its handshaking process, and performing and interpreting the handshake with a candidate C2 server. This process is equally necessary for Peer-to-Peer botnet detection, as botnets such as Mozi purposefully attempt to obfuscate Peer-to-Peer botnet activity through

interacting with legitimate peers via publicly hosted Distributed Hash Tables. [27] As proposed by Herwig et al. [26], DHT protocols, used by the likes of Mozi or Hajime, can be monitored, and corresponding peer nodes validated through the successful invocation of a handshake.

Despite the proposed identification heuristics proving effective in candidate C2 and P2P detection, some heuristics proved less applicable than others throughout the evaluation period. Namely, one of the heuristics based on hard-coded IP addresses of malware binaries proved challenging to utilise. The VisiBot Processing System had difficulties unpacking several malware binaries packed with tools such as UPX [70], leading to the extraction of obfuscated strings during static analysis. Through implementing a robust malware unpacking procedure that supports various packing methods, crucial strings, such as hard-coded IP addresses used in the heuristic analysis, can be successfully extracted. By doing so, heuristics based on static analysis features, such as strings, will not be hindered by common obfuscation techniques. The application of additional identification heuristics may also allow for increased C2/P2P detection-rate and accuracy, using different data sources collected during static, dynamic and network analysis, such as captured packet files (pcaps) and machine logs.

Lastly, a considerable limitation of the VisiBot Processing System is that it is limited to only retrieving honeypot results from the Bad Packets [51] honeypot service on an hourly basis. The hourly period between queries significantly limits the number of malware samples extracted from remote code execution attempts conducted by malicious bots from botnets such as Mozi. As such bots self-propagate using randomly expiring payload URLs, there is little time to collect the binary before the web-server's network port hosting the binary changes. A possible solution may be to employ a stream-based honeypot collection system that allows packets to be processed as soon as the honeypot network detects them. Processing such packets in truly real-time will allow for the immediate extraction of expiring payload URLs, which, as observed, are becoming increasingly more common.

A Appendices

A.1 High-level Overview of VisiBot Processing System

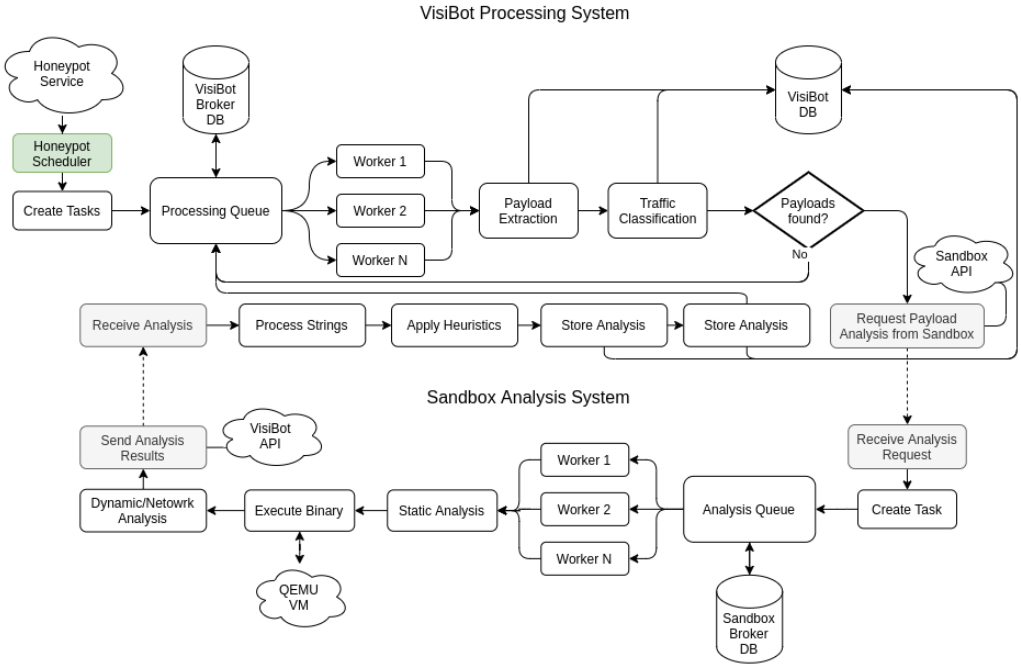


Figure A.1: *High-level flow-diagram overview of the VisiBot Processing System with sandbox integration.*

A.2 VisiBot MongoDB ER-Diagram

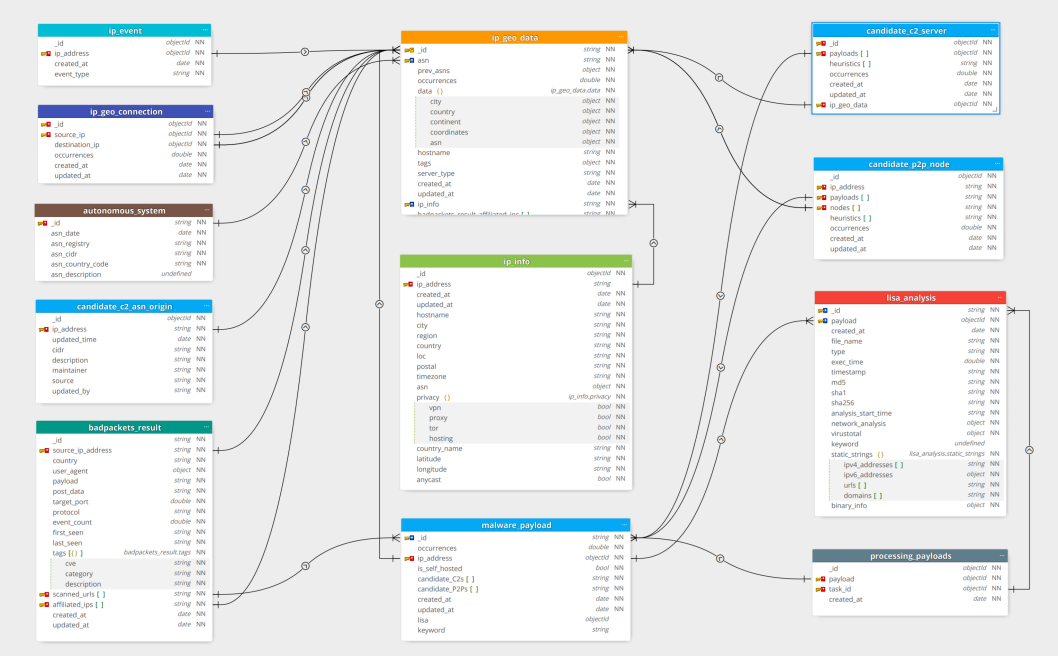


Figure A.2: *Reverse Engineered ER Diagram: VisiBot MongoDB Database.*

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

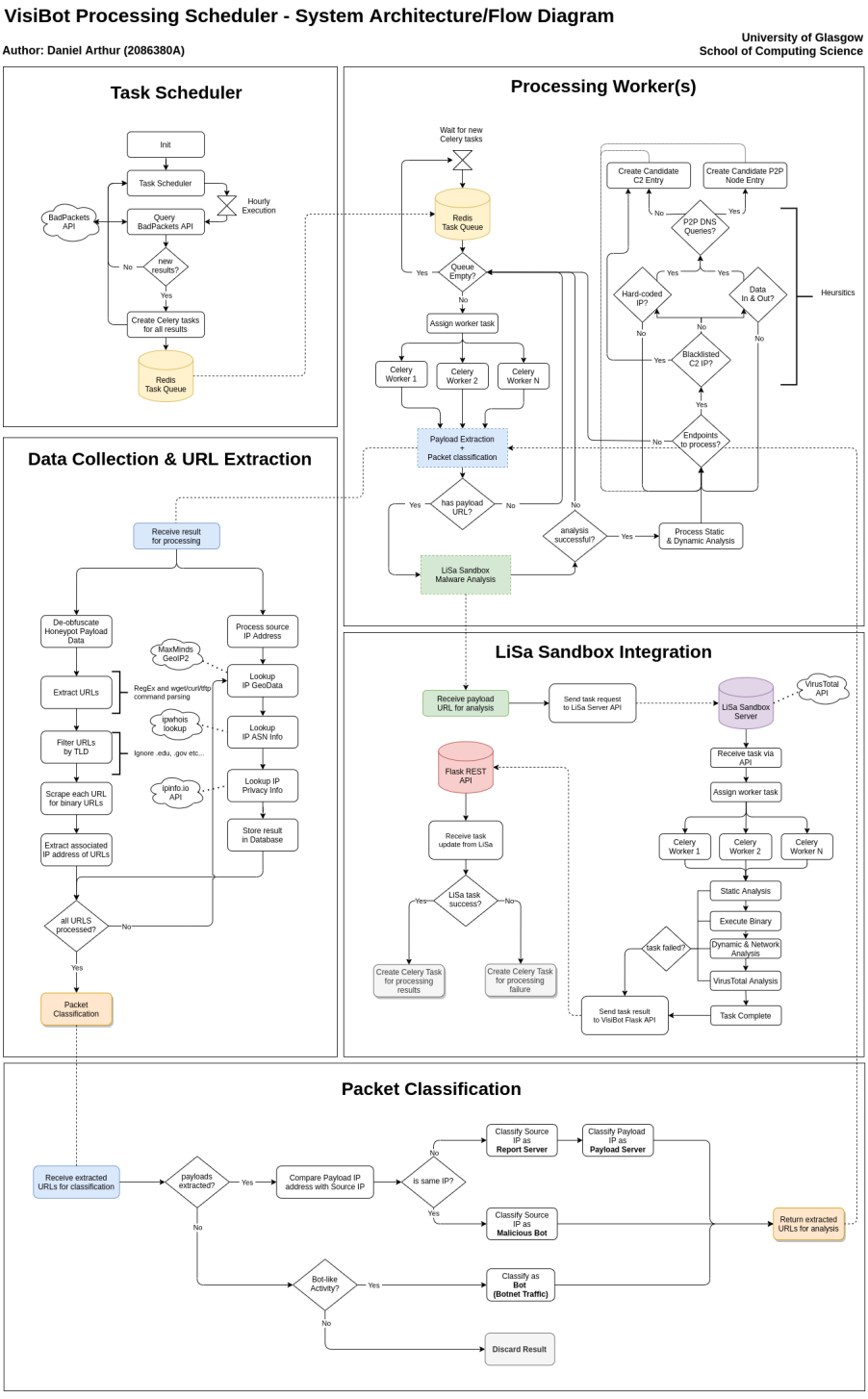


Figure A.3: *Overview flowchart diagram of the VisiBot Processing System.*

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

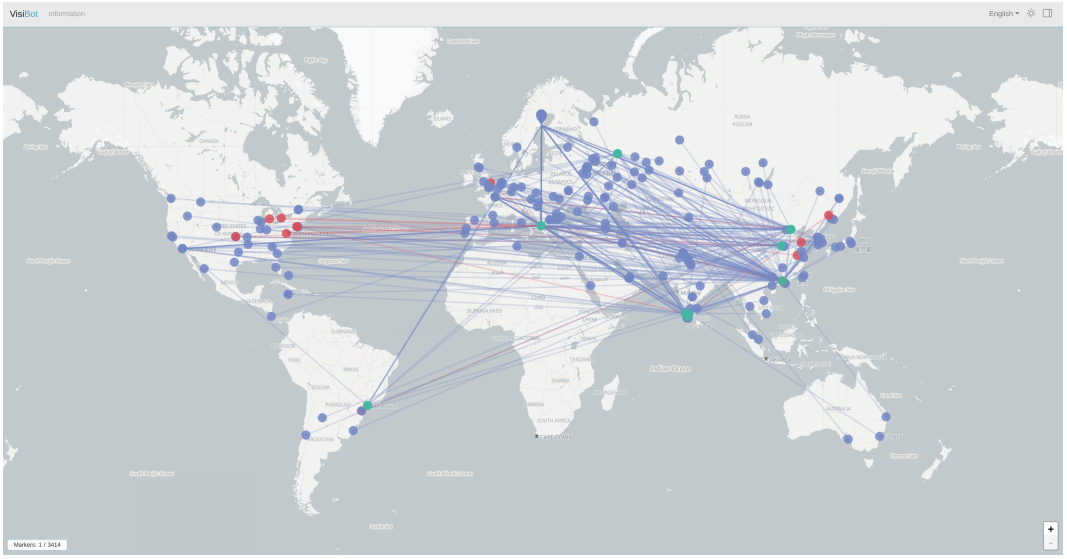


Figure A.4: *Screenshot of Peer-to-Peer connections shown within the VisiBot Interactive Map.*

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

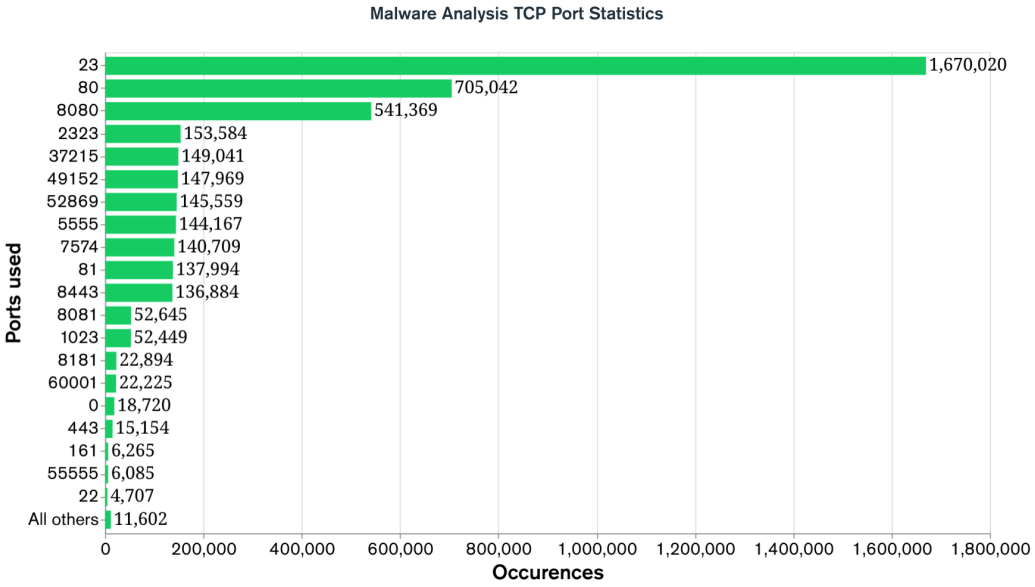


Figure A.5: Malware analysis TCP port usage statistics

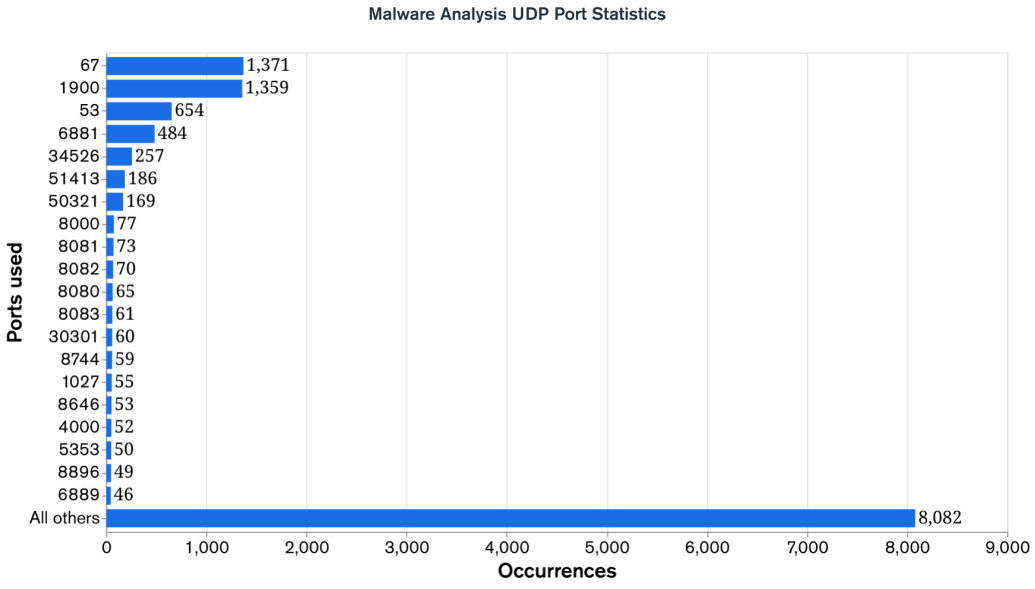


Figure A.6: Malware analysis UDP port usage statistics

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.

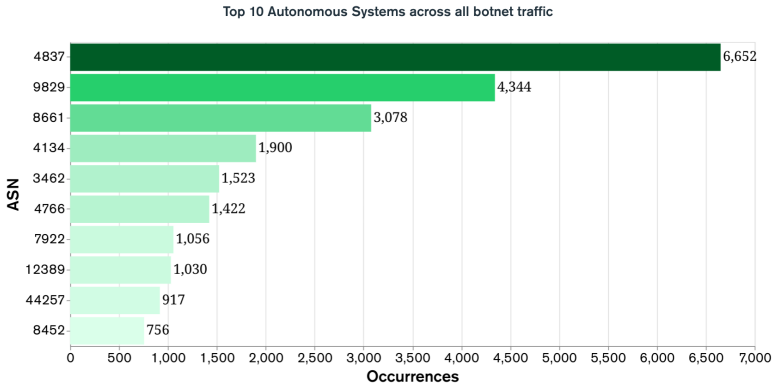


Figure A.8: *Top 10 Autonomous Systems based across all traffic.*

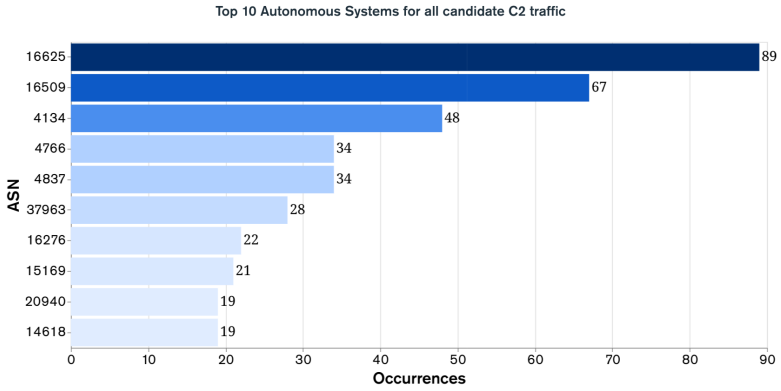


Figure A.9: *Top 10 Autonomous Systems based across all candidate C2 traffic.*

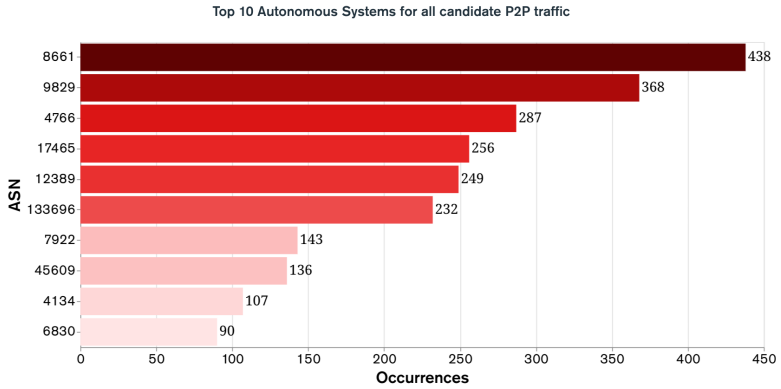


Figure A.10: *Top 10 Autonomous Systems across all P2P traffic.*

7 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 1, 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] Daniel Uhříček. Lisa - linux sandbox. https://github.com/danieluhricek/LiSa, 2019–2020. [Accessed March 24, 2021].

[11] Rik Ferguson. The botnet chronicles: A journey to infamy, 2010. URL https://www.trendmicro.co.uk/media/wp/botnet-chronicles-whitepaper-en.pdf. [Accessed March 24, 2021].

[12] Craig A Schiller, Jim Binkley, David Harley, Gadi Evron, Tony Bradley, Carsten Willems, and Michael Cross. Chapter 8 - irc and botnets. *Botnets: The Killer Web App*, pages 285–311, 2007. doi: 10.1016/B978-159749135-8/50005-6. URL https://doi.org/10.1016/B978-159749135-8/50005-6.

[13] Rob Morton and Tom Barth. Fast flux botnets still wreaking havoc on the internet according to akamai research. https://www.akamai.com/uk/en/about/news/press/2017-press/fast-flux-botnets-still-wreaking-havoc-on-internet-according-to-akamai-research.jsp, 2017. [Accessed March 24, 2021].

[14] Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst Biersack, and Felix Freiling. Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm. *LEET'08: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 1–9, 2008. doi: 10.5555/1387709.1387718. URL https://dl.acm.org/doi/10.5555/1387709.1387718.

[15] Karim El Defrawy, Minas Gjoka, and Athina Markopoulo. Bottorrent: Misusing bittorrent to launch ddos attacks. *3rd Workshop on Steps to Reducing Unwanted Traffic on the Internet*, 2007. URL https://www.usenix.org/biblio-8505.

[16] Anon. Port scanning /0 using insecure embedded devices: Carna botnet. http://census2012.sourceforge.net/paper.html, 2012. [Accessed March 24, 2021].

[17] Federico Fazzi and Kevin Kirsche. Lightaidra. https://github.com/eurialo/lightaidra, 2016. [Accessed March 24, 2021].

[18] Anon. Atma - darknet - securite internet - deepweb - darkweb. http://atma.es/, 2012. [Accessed March 24, 2021].

[19] NJCCIC. Aidra botnet. https://www.cyber.nj.gov/threat-center/threat-profiles/botnet-variants/aidra-botnet, 2016. [Accessed March 24, 2021].

[20] NHS Digital. Mmd-0056-2016 - linux/mirai, how an old elf malcode is recycled.. https://blog.malwaremustdie.org/2016/08/mmd-0056-2016-linuxmirai-just.html, 2016. [Accessed March 24, 2021].

[21] Eduard Kovacs. Bashlite malware uses shellshock to hijack devices running busybox. https://www.securityweek.com/bashlite-malware-uses-shellshock-hijack-devices-running-busybox, 2014. [Accessed March 24, 2021].

[22] The MITRE Corporation. Cve-2014-6271. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271, 2014. [Accessed March 24, 2021].

[23] Artur Marzano, David Alexander, Osvaldo Fonseca, Elverton C Fazzion, Klaus Steding-Jessen, Marcio Jose Chaves, Italo Cunha, Dorgival Olavo Guedes, and Wagner Meira Jr. Understanding the mirai botnet. *2018 IEEE Symposium on Computers and Communications (ISCC)*, pages 1093–1100, 2018. doi: 10.1109/ISCC.2018.8538636. URL https://doi.org/10.1109/ISCC.2018.8538636.

[24] MalwareMustDie. Bashlite linux botnet. https://blog.malwaremustdie.org/2016/08/mmd-0056-2016-linuxmirai-just.html, 2016. [Accessed March 24, 2021].

[25] 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.

[26] 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.

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

[28] Elisa Bertino and Nayeem Islam. Botnets and internet of things security. *Computer*, 50: 76–79, 2017. doi: 10.1109/MC.2017.62. URL https://doi.org/10.1109/MC.2017.62.

[29] Carl. Lividas, Robert. Walsh, David. Lapsley, and W. Timothy. Strayer. Proceedings. 2006 31st ieee conference on local computer networks. *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, pages 1094–1101, 2006. doi: 10.1109/LCN.2006.322210. URL https://doi.org/10.1109/LCN.2006.322210.

[30] 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.

[31] 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.

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

[33] 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.

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

[35] 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.

[36] 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.

[37] Jiajia Wang and Yu Chen. P2p botnet detection method based on data flow. *2017 2nd International Symposium on Advances in Electrical, Electronics and Computer Engineering (ISAEECE 2017)*, 2017. doi: 10.2991/isaeece-17.2017.44. URL https://doi.org/10.2991/isaeece-17.2017.44.

[38] Jiabin Li and Zhi Xue. Distributed threat intelligence sharing system: A new sight of p2p botnet detection. *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*, 2019. doi: 10.1109/CAIS.2019.8769511. URL https://doi.org/10.1109/CAIS.2019.8769511.

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

[40] 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].

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[58] 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].

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

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

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

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

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

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

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

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

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

[68] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[84] Atlassian. Trello. https://trello.com/, 2011–2021. [Accessed March 29, 2021].

[85] 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].

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

[87] 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].

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

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

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

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

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

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

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

[95] 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].

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