C2Miner: Tricking IoT Malware into Revealing Live Command & Control Servers

Ali Davanian
UC Riverside | NVIDIA
Riverside, USA
adava003@ucr.edu

Michalis Faloutsos
UC Riverside
Riverside, USA
michalis@cs.ucr.edu

Martina Lindorfer
TU Wien
Vienna, Austria
martina@seclab.wien

ABSTRACT

How can we identify live Command & Control (C2) servers for a given IoT malware binary? An effective solution to this problem constitutes a significant capability towards detecting and containing botnets. This task is not trivial because C2 servers are short-lived, and they use sophisticated and proprietary communication protocols. We propose C2Miner, a novel approach to trick IoT malware binaries into revealing their currently live C2 servers. Our approach weaponizes old disposable IoT malware binaries and uses them to probe active servers. We provide novel solutions to overcome the following challenges: (a) disambiguating the C2-bound traffic generated by the malware and (b) determining if a target IP:port is indeed a C2 server as opposed to a benign server.

In our evaluation, based on 3M distinct exploration attempts over 150K distinct IP addresses, we show that we can identify C2 servers within a given IP:port space with an F1 score of 86%. In addition, we show how our approach can be used in practice and at scale. Conducting a large-scale probing campaign has scalability issues given that the number of probes is proportional to the IP addresses, the number of ports, and the number of binaries from distinct families which we want to explore. To address this challenge, we propose a grammar-based method to fingerprint and cluster C2 communications which, among other applications, allows us to select malware binaries for weaponization efficiently. Additionally, we use spatio-temporal features of C2 servers to narrow down our search in the entire IP space. An optimistic observation from our study is that using only 2 (more than 6 months) old IoT malware binaries, we scan 18K IP:port pairs daily for 6 days and find 6 new live C2 servers.

CCS CONCEPTS

• **Security and privacy → Malware and its mitigation**.

KEYWORDS

IoT, malware, network security, command & control servers

ACM Reference Format:

Ali Davanian, Michalis Faloutsos, and Martina Lindorfer. 2024. C2Miner: Tricking IoT Malware into Revealing Live Command & Control Servers. In *Proceedings of ASIA CCS 2024 (AsiaCCS'24)*. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3634737.3644992

AsiaCCS'24, July 01–05, 2024, Singapore

ACM ISBN 979-8-4007-0482-6/24/07.
https://doi.org/10.1145/3634737.3644992

1 INTRODUCTION

Identifying Command and Control (C2) servers is critical in the battle against botnets, and in particular against botnets targeting Internet of Things (IoT) devices. As their name suggests, C2 servers control their bots and orchestrate malicious activities, such as Denial of Service (DoS) attacks. Once the C2 servers of malware are known, we can mount a defense to contain its proliferation and damage, e.g., by monitoring or blocking traffic to these destination addresses. This is an especially effective defense in the case of IoT devices, because they do not have enough computation power to have sophisticated on-device defenses like anti-virus (AV) software. In addition, we can identify infected devices within a network once we know the C2 addresses they communicate with. Finally, Internet Service Providers (ISPs) and law enforcement can block and take down these C2 servers to disrupt the botnet [35, 41].

Given their crucial role in the operation of botnets, C2 domain names and IPs have long been used as indicators of compromise (IoCs), and are part of different (commercial) threat intelligence feeds. Nevertheless, the effectiveness of these feeds in terms of coverage, accuracy and timeliness has come under scrutiny [9, 38]. Furthermore, the issue of coverage of domain-based feeds has been known for quite some time: a study in 2014 found that *"the union of 15 public blacklists includes less than 20% of malware domains"* [34].

The problem of identifying C2 server is challenging especially if liveliness is of interest. The difficulty lies in the fact that C2 servers (a) use proprietary and increasingly sophisticated protocols [21], (b) are short lived [52], and (c) use domain generation algorithms (DGAs) to dynamically generated addresses [51]. So far the C2 mechanisms of IoT malware have been found to be less sophisticated than malware on other platforms: malware binaries typically hardcode IP addresses [3], and do not include fallback mechanisms [52]. However, this also means that IoT malware does not yet employ C2 discovery mechanisms that could be reverse engineered and used to discover new server locations [42, 45].

The current best practice is based on manual effort and domain knowledge [60], as well as of sharing of information, such as IoCs. Analyzing and activating malware binaries is one method to extract IoCs, including C2 servers, but this effort has its own challenges as we detail later in this paper. Most importantly, by the time analysts find malicious binaries and analyze them, botmasters have most likely moved their servers to new locations — essentially abandoning any existing binaries. This is because reviving an IoT botnet is relatively easy, and botmasters avoid maintaining them [52].

Problem definition: *How can we identify live C2 servers for a given IoT malware binary?* This is the question that lies at the heart of our work. Our input is an IoT malware binary and a target IP:port space of interest (target for short). This target space is provided by the user and it could be an enterprise network or a space of interest for a research study. Our desired output is: (a) live C2 servers for that binary in the given IP:port space, and (b) "fingerprints" of the C2 communication, which can be used to identify C2 servers in a network trace. We assume nearly *zero a priori knowledge*, which makes the problem harder but more relevant in practice. Specifically, we assume that we are not given any a priori knowledge about: (a) the binary (e.g., its family), (b) the target space or reputation information per IP address, and (c) actual bot traffic patterns, i.e., we do not require access to existing traffic traces.

State of the art: There has been limited work addressing the problem in the way we have framed it here. Most related work falls in the following broad categories: (a) dynamic analysis and profiling approaches, (b) network traffic fingerprinting methods, and (c) active probing methods and studies. The most relevant technique to this work is the use of "milkers" to find C2 servers [5]. The idea behind milkers is to reverse engineer the application layer protocol used by the malware and, then, implement an imitation of it to find live C2 servers. There are two main limitations with this approach that may impede its wider adoption, especially as IoT malware becomes more sophisticated in the future. First, any manual effort to reverse engineer a malware communication protocol has scalability issues due to the need for human effort. Second, automated reverse engineering of traffic have its own limitations, especially if the malware deploys sophisticated techniques such as encryption [19, 36].

Contributions: We propose C2Miner, a systematic probing approach for discovering live C2 server for a given malware binary with zero a priori knowledge. The novelty of our approach is that we weaponize malware binaries by turning them into spies. We accomplish this by developing novel algorithmic solutions to (a) disambiguate the C2-bound traffic initiated by the binary, (b) determine if a target IP is indeed a C2 server, and (c) fingerprint and cluster C2 communication, which is necessary for exploration at scale and intrusion detection. Our fingerprinting approach is a significantly more powerful approach to existing approaches, which we discuss in §8: our method (a) defines a formal grammar to model the *entire communication* as a string, and (b) introduces a customizable way of representing a packet as we elaborate in §4. Our approach allows the traffic to be modeled at different levels of granularity and is among the first approaches to create an abstract representation of a network traffic flow using a formal language without the need for payload decryption. As we discuss in §7, our fingerprinting can also generalize to other applications.

In summary, we make the following contributions:

- **We disambiguate C2-bound traffic accurately.** We propose an algorithm that can distinguish C2-bound traffic from other traffic (i.e., other malware activity, such as scanning and exploitation) with 92% precision.
- **We determine C2 servers accurately.** We develop an approach that can identify C2 servers among benign Internet servers with an F1 score of 86%. To achieve this, we introduce a grammar-based approach for characterizing the communication between bots and C2 servers.
- **We show that we can utilize old binaries to discover current live servers**. We find that 84% of pairs of malware in our dataset can interchangeably talk to the same C2 server. We show the promise of our approach in practice with a small-scale study. Using two six-month-old binaries, we identify six live servers during a six day probing campaign targeting 1,536 IP addresses on 12 ports.

Availability: We provide our source code at https://github.com/adava/CnCHunter as an extension of our work presented at Black Hat [17], and release our ground truth and datasets (Appendix F).

Ethical considerations: We discuss the implications of executing malware samples and interacting with live servers and our safeguards in §5.1 and Appendix A.

2 SCOPE, ASSUMPTIONS AND LIMITATIONS

Threat model: The threat is posed by a deployed and active botnet and our goal is to defend against the attacker by detecting the live C2 servers, which will help us mitigate and contain the malware. Conversely, the attacker wants to avoid revealing its servers to our probes. We have a binary of the malware that we activate in order to communicate with a target IP in an effort to see if it is a C2 server. We redirect the C2 traffic from the malware to candidate addresses. For instance, if we were to scan a single port within a subnet of 256 nodes, we need to redirect malware traffic to all these nodes (at the given port) and output which one(s) are a C2 servers. Naturally, the goal of the detection is to balance coverage (finding all the live servers) and accuracy (avoiding misclassification of benign IPs).

We discuss the key assumptions of our problem framing below:

a. Activation. We assume that the malware instance that we try to exploit can be activated in our sandbox environment. This means the malware would unpack itself (if packed) and start the communication with the C2. Thus, we need to provide a sandbox environment that sufficiently mimics an IoT device for the malware to start the execution. In addition, we do not need to activate all the binaries, as long as as we activate instances that uses a sufficiently-similar communication protocol.

b. Encryption. We assume that the malware is not using a layer III/IV encryption like IPSec. This is different from the use of encryption at the application layer which does not affect our solution, e.g., the use of TLS does not break our approach, although it might impact its accuracy. We revisit this issue in §7.

b. Communication type. We assume that the botnet uses a traditional hierarchical server-bot communication and common communication channels without the use of covert channels.

c. Network protocols. We assume that the malware uses the IP protocol at layer III and the TCP protocol at layer four. Extending our techniques to UDP is mostly an engineering effort.

d. Connection persistence. We assume that the malware attempts to connect to a non-responsive C2 server multiple times, which is a reasonable assumption as we discuss in §3.4.

Although we argue that the above assumptions are sufficiently representative of the current IoT malware landscape, they may not be true in the future. In particular, if we can not activate any malware sample of a particular family, our solution will not work. However, this is an issue for any method that relies on dynamic analysis. Naturally, one can expect that malware authors that become aware of C2Miner may actively develop methods to defeat our system. We discuss this and potential countermeasures in §7.

Feasibility assessment. We conducted a small-scale manual study to assess the feasibility of our approach, which we describe in Appendix B. We manually analyzed well-known IoT malware families and we found (a) bad news: communication protocols vary significantly between families; and (b) good news: The protocol tends to remain nearly identical for binary samples of a family. These observations are corroborated by our experiments in §5 and §6.

Target space selection and prioritization. Selecting the target IP for scanning presents two distinct but related questions: (a) what is the initial space of exploration, and (b) how should we prioritize its exploration. We consider these questions to extend beyond the scope of this work. First, we consider the overarching space for exploration as part of the problem input. In practice this could be driven by the goal of the study. For example, a company may want to scan its own space for the existence of C2 servers. Another approach is to identify "Internet Neighborhoods" with bad reputation using historical behavior. Second, prioritizing the exploration of the desired space is a deployment optimization problem, especially if we assume there is a limited "budget" for the exploration. By contrast, this is not a concern in the case of a small target space or abundant resources. We further discuss prioritization in §6.

3 DESIGN AND IMPLEMENTATION

In a nutshell, C2Miner takes as input one or more IoT malware binaries and a target IP:port space to explore. C2Miner probes the given target space by redirecting the C2 traffic of the malware to find live C2 servers. To achieve this, we first need to execute the malware to a point where it generates C2 traffic (*binary activation* §3.1). Then, we need to detect the malware C2 traffic (*traffic disambiguation* §3.2). Next, we need to redirect the C2 bound traffic to the targets within the given IP:port space (*MitM-enabled probing* §3.3). Finally, we need to determine whether a target is indeed a C2 by analyzing the communication (*C2 determination* §3.4).

Figure 1 shows an overview of the architecture of C2Miner. The *sandbox* is in charge of the binary activation. The *profiler* oversees the disambiguation and the C2 determination. The *MitM* module implements the redirection of the traffic to the target IP address.

A significant contribution of our work is our grammar-based fingerprinting capability (see §4). In our context, fingerprinting refers to the detection of a remote network service based on its network footprint. We use this capability in two ways. First, we use it in the profiler for determining the existence of a C2 server. Second, we use it to cluster binaries based on their communication patterns to prioritize a proof-of-concept deployment (see §6).

3.1 Binary Activation

Here, we briefly explain how we activate a malware binary in our sandbox, although the sandbox component of C2Miner is not the focus of this study (see Appendix C for more details on the underlying virtualization environment). On a high level, we overcome three challenges for activating an IoT malware binary, i.e., drive the the execution to a point where it generates network traffic:

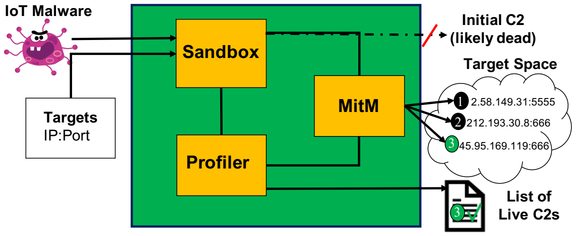

Figure 1: Overview of C2Miner and how it tricks malware binaries into revealing live C2 servers by activating them, and then redirecting the traffic to scan IP spaces of interest.

First, the binary must be executed on the corresponding CPU architecture. To do so, we statically analyze the binary to find the right architecture for execution. If the binary is packed, we unpack it using well-known packers (such as UPX). If we can not determine the CPU architecture after unpacking, we iteratively execute the binary on every CPU architecture that the sandbox supports until the binary is activated, or we exhaust all options.

Second, we need to bootstrap a virtual environment for execution. Executing a malware on actual IoT devices is not feasible in practice: there are many IoT devices and given our zero knowledge assumption, we do not know which device it targets. Furthermore, a virtualized environment allows us to instrument the execution, and efficiently analyze the malware. Our *sandbox* emulates the malware execution using QEMU [6], and as we report in §5.3, QEMU performs very well despite occasional failures due to instructions not being supported by the virtualization engine.

Third, the execution platform, in terms of operating system and the file system, should be configured properly for the malware to activate. For instance, if the malware looks for a specific file, the absence of that file would result in the early termination of the execution. We use RiotMan [15] for this purpose, which enables us to provide the appropriate configuration.

During the binary activation process C2Miner automatically copies the malware executable to the filesystem in the start-up script directory, and starts the input malware emulation. The profiler module stores the logs for later analysis by other modules. It logs network traffic and system call traces. The former provides us with the data for C2 communication analysis, while the latter allow us to verify the correctness of the emulation. Our emulator executes the malware with the `strace` logging enabled. For network traffic collection, we enable the QEMU network bridging functionality, and record the traffic in pcap format. Finally, to communicate with the candidate addresses, the guest (the virtual machine that the malware runs on) needs an active Internet connection, so we activate IP forwarding on the Linux host. We also activate the ARP proxy configuration and NAT the traffic to the outside world.

3.2 Traffic Disambiguation

Traffic disambiguation aims to distinguish between the C2 and non-C2 traffic that the malware generates. The C2 traffic is only a small percentage of all the traffic that the malware generates. IoT malware, like other worm-like malware, tries to infect other devices, and hence, it generates scanning and exploitation traffic [2, 3]. In addition, there might be random traffic, for instance, for checking Internet connectivity on the infected device. Our profiler module finds the C2 address (IP:port or DNS name) that the malware tries to communicate with from a large amount of traffic that the malware generates (and contains all the aforementioned types of traffic).

Algorithm 1 Disambiguate-C2-Traffic

```
                         Input: Packets
    Output: Scores                                          ▷ for IP:ports
 1: $TargetStats \leftarrow \{\}$  ▷ A hashtable tracking the number of connections to each target (ip:port or DNS).
 2: $Ports \leftarrow \{\}$  ▷ A hashtable tracking the number of times a destination port is seen.
 3: $Scores \leftarrow [\,]$                ▷ A list of targets with their C2 likelihood score.
 4: for each $pkt \in Packets$ do
 5:     if $Approved(pkt) == TRUE$ then
 6:         $target \leftarrow Get\_Target(pkt)$
 7:         $Update\_Target(target, TargetStats)$
 8:         $Update\_Ports(target, Ports)$
 9: for each $target \in TargetStats$ do
10:     if $is\_DNS(target)$ and not $White\_list(target)$ then
11:         $Scores[target] \leftarrow Calc\_DNS\_Score(target)$
12:     if $is\_IP(target)$ then
13:         $Scores[target] \leftarrow Calc\_IP\_Score(target, Ports)$
14: $Sort\_Desc(Scores)$
15: return $Scores$
```

We developed the *Disambiguate-C2-Traffic* algorithm illustrated in Algorithm 1 to identify C2 traffic. It analyzes each target in the traffic generated by the IoT malware and assigns a score that shows the likelihood of a target being a C2 server. We use the term "target" to refer to either an IP:port tuple or a DNS address. We take the top N (configurable but $N = 1$ by default) most likely targets as the malware's C2 servers. We implemented Algorithm 1 using Python's pyshark library [31]. The algorithm consists of the following steps:

Part 1: Quantifying activity. As a first step (lines 4-8), we analyze each packet and count the number of times the targets (IP:port or DNS name) are contacted. At line 5, we filter out the traffic from unrelated protocols: ICMP, DHCP, ARP and NTP. We assume that the malware uses TCP for C2 communication, which seems to be the preferred protocol [42]. Note that extending our approach to UDP and other protocols is a matter of engineering. In addition, although ICMP and NTP are sometimes used as covert channels, we did not find such instances in IoT malware.

Next, at line 6 and 7, we quantify the number of times a target is contacted. We make a distinction between the IP and DNS addresses. For IP:port targets, we count the number of packets to those targets. Our insight here is that a binary will exchange a large number of packets with its C2 server. In fact, we anticipate that even if the C2 is not alive, the malware will most likely attempt to connect multiple times. For DNS-based targets, we consider two cases. If the DNS resolution succeeds, then, we focus on the resolved IP:port tuple which we explained above. If DNS resolution fails, we count the number of times that the DNS queries fail. This is based on the observation that a blocked C2 DNS address would not resolve, and hence there will not be a connection to an IP address.

Part 2: Calculating the C2 likelihood score. Having completed the first step, we can perform the second step and calculate the likelihood score for each target (lines 9 to 13). First, for DNS-based addresses, we check the reputation of the domain and filter out the reputable domains from further analysis. To do so, we associate the reputation of a domain to its popularity ranking measured by Alexa and eliminate the top X (configurable but $X = 1,000$ by default) well-known domains based on their Alexa ranking. Note that we check the ranking of the entire DNS address, and not only the second level domain (e.g., Microsoft's reputation and ranking is different from its subdomains). This means that a cloud-based C2 address (e.g., hosted on Microsoft or Amazon) would not have necessarily equally high reputation. If the DNS address passes the reputation check, the score is the number of times it was queried.

Second, for IP:port-based targets, we assign a score that considers the activity for both the target and the port by relying on two insights. The first insight is that a bot will have regular communication with the C2 server, so the higher the number of packets sent to an IP:port pair, the higher the likelihood it is a C2 server. Note that this insight does not refer to the percentage of the C2 traffic compared to the overall traffic, but rather the frequency of communication to a particular target compared to other targets the malware tries to communicate with. The second insight is that a port number used for the C2 server is different from ones used for other bot traffic, such as proliferation (scanning and attacking other devices), in which case the same port will be used across many different IP addresses. The higher the number of IP addresses that are being contacted at a specific port is, the lower is the likelihood that a communication at that port is towards a C2 server. These two insights can be quantified in different ways. We chose the most straightforward way that does not require any additional parameters such as weights: We calculate the score of an IP:port pair as the number of packets to that IP:port divided by the number of times the port was used. This works well in practice as we show in §5.3.

3.3 MitM-enabled Probing

The task of the machine-in-the-middle (MitM) module is to redirect the malware C2 traffic to the input targets. We use the term "candidate" to refer to these targets that may be C2 servers. The traffic redirection results in traffic exchange between the malware and the target that later can be used to determine whether the target is a C2 server. Note that this module assumes that Algorithm 1 has already identified the C2-bound traffic. The MitM module handles IP:port-based and DNS-based targets in the following ways:

a. IP:port-based targets. Here, we want to replace the IP and port of the C2 server that the malware wants to reach with that of the target of interest. Implementing this functionality is non-trivial as using the readily available NAT functions at the network perimeter would not be sufficient. For ethical reasons, we can not let the traffic leave the infected machine. Hence, we move the address manipulation to the guest level. We used the Linux iptables, which we adapted to work in our case, as following. In the NAT mode, iptables allows altering the packets as soon as they come in and/or as they go out from the network proxy if the traffic originates from the proxy itself. Since we rely on the guest's (i.e., the infected machine's) iptables to provide the redirection, we can alter the packets and redirect them before the traffic leaves the guest.

b. DNS-based targets. Operating systems use different DNS resolution methods. Linux starts by looking up the DNS address in a host file that maps DNS names to IP addresses and uses other methods only if this method fails (although the order can be modified). We take advantage of this process, and modify the host file (`/etc/hosts`) of the guest machine that maps DNS addresses to IPs. We add an entry that maps the C2 DNS address to the candidate address, similar to the IP case, which we look up. Doing so, the traffic is redirected to the candidate address.

3.4 C2 Determination

One of the profiler module's tasks is to determine whether a target is a C2 server based on its traffic exchange with the malware. This problem can be solved if we have a priori knowledge about the application protocol used by the malware, however, we assume such knowledge is not available. If traffic samples of communication with the C2 server are available, some knowledge can be drawn through the analysis of the traffic and the problem can be solved differently. Here, we only focus on the transport layer (IV layer of the OSI model) headers and payload. We make no effort to understand the malware's application layer protocol, which often is encrypted.

We propose two methods: (a) a SYN-DATA-aware (see below), and (b) a Fingerprinting-aware method (see §4). The latter can have wider applications and we compare the two methods in §5.4.

The SYN-DATA-aware method. The solution that we present here is independent of the application layer protocol used by the malware and should work even in the case of encryption at the application layer. Our solution is based on the insight that if a target is a C2 server, it should engage with the malware in a "meaningful" way. A meaningful or successful connection could imply different behaviors for different application layer protocols. Assessing success based on the network-level features derived from the transportation layer satisfies the assumption that no a priori knowledge about the application layer protocol is available.

Determining meaningful communication at the transport layer is challenging. On the TCP transport layer, a successful connection completes the handshake. However, a successful handshake does not always indicate a successful connection (at the application level). After the handshake, endpoints might immediately close the connection by sending RST/FIN flags or silently not respond back. Alternatively, an endpoint might respond with a packet informing the target about an error at the application level. For instance, in response to an invalid HTTP request, the server might respond with a "400 Bad Request" error code. It should be noted that these are all possible scenarios since we redirect a malware sample's C2 traffic to a candidate that might or might not be a C2 server.

We reduce the problem of determining whether the target is a C2 server to whether the communication succeeds or fails by assessing transport layer features. If the communication fails, regardless of the reason for the failure, we find that malware insists on retrying to connect to the server. To illustrate this, consider an analogy to the human communication. If two individuals do not understand each other's language, they restart the conversation and repeat themselves. Similarly, if a candidate address listens to our contacted port, but it does not speak our malware application layer protocol, it will most likely restart the connection. We can identify this behavior by looking at the number of times the SYN flag is set for a particular candidate. This approach is error prone in case malware constantly closes the connection to a port and re-opens it. However, such cases are rare, as we discuss in §5.4.

If the communication is successful based on the number of SYN flags, we check for exchange of data to further assess the communication success. In general, after completing the handshake, malware sends data to the C2 server, which is a payload at the TCP layer. This could be a simple "PING" command. In response, the server typically acknowledges the packet and might choose to respond with a TCP payload. The TCP segment from the server contains control flags and zero or more bytes of data. We find that factoring in the exchange of data increases the precision, because false positives are rare (see §5.4). In summary, for C2 determination, we check whether: (a) the number of times the SYN flag has been set is lower than a threshold t (configurable but $t = 1$ by default), and (b) if there is an exchange of data packets between the malware and the target.

4 GRAMMAR-BASED FINGERPRINTING

How can we profile the bot-to-C2 server interaction at the network level? This is the question that we are trying to answer in this section. We want to find communication patterns that are uniquely indicative of C2 communications, and at the same time robust to variations and noise. By comparing these fingerprints with captured traffic, we can identify the presence of a C2 server. To do so, we propose to model the network flow as a dialogue and model it using a grammar. At the network level, if we look at the dialogue, we can see the following features that depend on the applications (vs. the operating system):

a. Number of send/receive pairs. As in a dialogue, the communication consists of request/response pairs. For instance, just like "Hi" answers to a "Hi," a "PING" or a "PONG" is a response to a "PING."

b. Order within the send/receive pairs. Is it the client who initiated the exchanged pair or the server? Please note that this is independent of who first initiated the flow. The order within each pair can change at any time. A dialogue can go quiet, and each of the participants can start a new send/receive pair.

c. Termination control flags (if any). This helps us understand how the connection terminated, for example a network failure.

At a high level, we summarize the flow by capturing: (a) its beginning and end, (b) its adherence to TCP flow rules, and (c) select properties of its packets. We design a grammar G that models the raw traffic to a *"phrase in this language"* based on the above observations. Formally, we define $G = (N, \Sigma, P, Flow)$, where

$$N = \{Flow, FlowBody, DataPacket, ControlPacket, Flags, Sender, Attributes\}$$

are the non-terminals symbols, and

$$\Sigma = \{handshake, client, server, ACK, SYN, FIN, RST, attr^*\}$$

are the terminals symbols (we explain $attr^*$ below). Note that $Flow$ is the start symbol. We list the production rules set P in Algorithm 2. Please note that for readability (and to save space), we use the symbol "|" (corresponding to logical or) to present two rules in one line in some cases.

a. Summarizing a flow. Every flow starts with the start symbol $Flow$ and expects to "see" a TCP handshake captured by our terminal symbol $handshake$. This symbol corresponds to the three TCP packets: the client sends a packet with the SYN flag, the server responds with SYN and ACK flags and finally client sending an ACK. The rest of the flow consists of control or data packets from the endpoints denoted by the $Sender$ non-terminal and the $client$ and $server$ terminals. In our malware analysis, $client$ is the malware and the $server$ is the C2 server.

b. Summarizing each packet. Our grammar provides the ability to profile each packet in the flow using $attr*$. Defining what are

Algorithm 2 The Production Rules P

```
$Flow \longrightarrow handshake\ FlowBody$   (1)
$FlowBody \longrightarrow DataPacket\ FlowBody$   (2)
$FlowBody \longrightarrow ControlPacket\ FlowBody$   (3)
$FlowBody \longrightarrow \epsilon$   (4)
$DataPacket \longrightarrow Sender"\_"Attributes$   (5)
$ControlPacket \longrightarrow Sender"\_"Flags$   (6)
$Sender \longrightarrow client|server$   (7)
$Flags \longrightarrow Flag\ Flags|Flag$   (8)
$Flag \longrightarrow ACK|SYN|RST|FIN$   (9)
$Flag \longrightarrow \epsilon$   (10)
$Attributes \longrightarrow attr*$   (11)
```

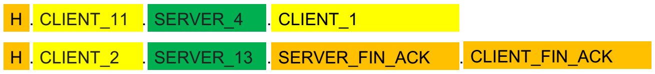

Figure 2: Two *real* short phrases in our language: H represents the TCP handshake abbreviated, followed by the yellow CLIENT_X and green SERVER_X packets of size X.

the right properties of packets can adapt to the sophistication of the malware. For example, the attributes can be a vector of values that could include packet size, byte entropy, or the appearance of a string in the payload. The latter requires deep packet inspection.

Our selection of attributes was driven by two considerations, we wanted to: (a) avoid deep packet inspection to increase its practicality, and (b) show the promise of the method. The former consideration suggests that only network headers are available. From a deployment point of view, this allows network devices to quickly process the packets, and also allows data sharing because of alleviated privacy issues. Another advantage is that packet headers are not affected by encryption at the application layer.

Given the above considerations, we define $attr*$ in rule (11) to be simply the length of the packet. In §5, we show that this attribute works sufficiently well with our dataset and the current sophistication level (or lack thereof) of the malware. Naturally, we will consider additional attributes in the future.

The best way to understand the grammar is through the examples illustrated in Figure 2. We use orange for control packets, yellow for data packets from the client and green for the server data packets. In these examples, "H" stands for the TCP handshake. The first example shows that after the handshake, the client (the malware) sends 11 bytes of data to which the server responds with 4 bytes. At the end, the communication terminates with 1 byte of data from the client. In the second example, the sever sends a packet with the FIN flag set at the end, and the client acknowledges this and also sends a packet with the FIN flag set.

c. Transforming flows into a "string" of the grammar. We take as input a traffic pcap file, and our goal is to transform each network flow between the malware and a destination into a string in our grammar. Note that the malware is always one of the communicating entities.

The process still hides several subtleties that we discuss here briefly. First, the traffic does not need to be restricted to a single flow, and it might contain multiple flows to the C2 server. This is because the traffic might be captured as the result of executing the malware several times, or because the malware opens and closes its connection to the C2 periodically. Second, we remove erroneous packets from our trace, i.e., packets that are not "part of the dialogue." In practice, these are packets that are not acknowledged by the other endpoint or re-transmissions of the same packet. This could be in the handshake process, or after the connection establishment. Finally, based on the features we mentioned earlier, we generate a string in our grammar from the remaining packets. At the end of our transformation phase, we have one string per flow between the malware and each destination of interest.

d. Comparing strings of the grammar. Given two strings in our grammar, we can quantify how similar they are using a distance function. We use clusters of strings to increase the robustness of our approach, which we discuss below. A cluster of similar strings based on a distance function indicates recurring patterns.

e. Clustering communication patterns. Clustering the transformed traffic in form of strings in our language can help us find common patterns, which we refer to as $fingerprints$. These fingerprints can be used in several applications beyond the scope of this work. Here we use it to (a) determine if a target is a C2 server, and (b) optimize the deployment of C2Miner, as discussed in §6. Our goal is to select a representative subset of binaries for probing a target space, such that we discover the same C2 servers as if we had used all the binaries. In essence, we can find malware clusters based on the similarity of their communication patterns.

For a given set of binaries N, we want to compare the similarity of their flows to their respective C2 servers. Formally, the input to the clustering algorithm is a set $S = \{S_1, S_2, S_3, ..., S_N\}$ where each S_i is the set of network flows for binary i to its C2 server represented by strings f_i in our grammar, $S_i = \{f_1, f_2, f_3, ..., f_P\}$.

We use hierarchical clustering which gives us the ability to study the clusters at different levels of granularity. Here, we use the *hierarchical k-means* algorithm, which seems to work well for our applications. For the distance function, we use the Jaccard distance based on the output of the Longest Common Sequence (LCS) algorithm of two flows f_i and f_j. We use $LCS(f_i, f_j)$ to calculate the intersection length. In order to calculate the union size, we define $Length(f_i)$ as the total number of sender chunks in our grammar. Since a sample's communication with the C2 could extend to multiple flows, the formula below accounts for these cases:

$$I(S_1, S_2) = \sum_{i=1}^{|S_1|} \sum_{j=1}^{|S_2|} LCS(f_i, f_j) \tag{12}$$

$$L(S_r) = \sum_{i=1}^{|S_r|} Length(f_i), \quad r = 1, ..., N \tag{13}$$

$$Jaccard(S_1, S_2) = \frac{I(S_1, S_2)}{L(S_1) + L(S_2) - I(S_1, S_2)} \tag{14}$$

Usage and practical considerations. We use our grammar-based method to fingerprint the communication between the malware and likely C2 servers. We report a fingerprint match under the condition that LCS returns a string that contains exchange of data from both endpoints. In §5.5, we evaluate this approach and discuss the choice of parameter k and the seeds for the k-means algorithm.

5 EVALUATION

We evaluate our approach by answering the following questions:

Q1: How accurately can we disambiguate C2-bound traffic from other malware traffic? (see §5.3)

Q2: How accurately can we determine that a probed target is indeed an active C2 server? (see §5.4)

Q3: Can our grammar-based fingerprinting help distinguish different families of malware? (see §5.5)

Q4: Do binaries exhibit "cross-talk"? Namely, if it is likely that two different binaries talk to the same C2 server? (see §5.6)

This area of research is notorious for having limited to non-existent benchmarks and ground truth despite several studies in this direction [52, 55]. Thus, we needed to create our own ground truth, which we provide to the community as detailed in Appendix F.

Implementation and experimental setup. We implemented C2Miner in roughly 3,500 lines of code (LOC) in Python and Linux shell scripts. For the evaluation, we use an x86-64 virtual server. Our machine has 4 Intel(R) Xeon(R) CPU E5-2686 v4 at 2.30GHz, and 16GB of RAM. It runs an Ubuntu bionic 18.04 AMD64 OS.

5.1 Ethical Considerations

We adhere to an ethical code of conduct and best practices for executing malware and interacting with live servers [49]. First, we do not violate anyone's privacy, as we never deal with any personally identifiable information. We only measure properties of devices (publicly accessible servers) and the services that run on them. Second, our measurement study is light-weight and it does not increase the load on servers or the traffic on the network significantly. Third, we take all possible measures to limit any potential harm. We only let C2 traffic communicate with real servers, and we filter out traffic that goes to any destination other than the C2 server with the exception of our limited controlled attempts for experiments we explain in §5.4; for those exceptions, we monitored the communication as it was happening. In addition, C2 traffic initiated by the bot towards the server is focused on establishing a communication, so it is not harmful. Fourth, even as the malware running in our sandbox temporarily joins a botnet, we are vigilant and have filters in place to recognize (a) C2 commands that could instigate an attack, and (b) outgoing filters to block any suspicious or potentially harmful traffic. In particular, through reverse engineering, we identify all the exploits the malware may run and have a filter in place to block them. We also limit the amount of ongoing traffic in terms of bandwidth and number of connections to mitigate DDoS attacks. We use SNORT IDS and iptables to detect and prevent malicious traffic from leaving our network. Furthermore, we have additional techniques in place to contain malicious traffic in each of our experiments as further detailed in Appendix A.

5.2 Malware Dataset

We collect malware binaries from MalwareBazaar [1] and VirusTotal [56] on a daily basis. We focus on the MIPS architecture, since it is a common platform for IoT devices and a less explored architecture by the community [15]. MalwareBazaar tags binaries as MIPS, while for VirusTotal, we query *elf* samples and filter for "mips" keyword. Later, in our static analysis stage, we only analyze MIPS 32B Big Endian samples that have C2-based communications, and not peer-to-peer (P2P) malware. To further validate that the binary is malware, we expect at least 5 engines to identify the sample as malicious (using the query `fs:1d+ type:elf positives:5+ mips`), which is aligned with established best practices [62].

Dataset	Description	Section
DAll	1,447 binaries collected in total	Base
Ground1	241 malware binaries used as ground truth	§5.3
Ground2	1,083 binaries and their IP:port C2 address cross verified by VirusTotal	§5.3
Ground3	202 binaries of Ground1 with a live C2 used as ground truth	§5.4
Trace-1	317MB traffic of 80 binaries from Ground3 redirected to 34 C2 servers and 39 benign servers	§5.4
DFinger	202 traffic fingerprints in our formal grammar of the 202 binaries from Ground3	§5.5
Trace-2	230MB traffic of 49 binaries from Ground3 redirected to "talk" to 32 C2 servers in January 2022	§5.6

Table 1: Our datasets of a total of 1,447 MIPS 32BE IoT malware samples from MalwareBazaar and VirusTotal.

To improve coverage, we collected binaries during four phases over the span of one year: (P1) March 29, 2021 to April 5, 2021; (P2) June 6, 2021 to August 11, 2021; (P3) October 25, 2021 to November 1, 2021; and (P4) November 10, 2021 to March 16, 2022. On average, we collect roughly 4 new unique malicious MIPS binaries a day. We refer to the union of all the samples as DAll. A summary of all our datasets is listed in Table 1. We initiated 3M exploration requests to 150K IP:port addresses over the course of our 24 weeks study period to augment and build these datasets. We identify 20K live services (IP:port) and trick 149 malware binaries into connecting to these addresses to prepare our ground truth. We manually checked the safety of our probes, i.e., that they are only "Call-Home" requests and do not modify the server state.

Activation of live malware-specified C2 servers. Due to the short-lived nature of C2 servers, we conduct the dynamic analysis on the same day we obtain the binary to maximize our chances of finding the malware-specified C2 servers. C2Miner successfully activates 90% of the binaries, which is on par with earlier studies [15]. We focus on activated binaries in the remainder of this work. The failures to activate were mainly due to "illegal instruction error" within QEMU, and in one instance, the malware self-terminated once it sensed the emulation environment.

Obtaining C2 traffic for ground truth. Activating the binary is just the beginning: we still need to collect its C2 traffic, which is not straightforward. First, we need a live C2 server, but they are rare and ephemeral. In fact, the malware-specified C2 servers (which the malware attempts to contact initially on its own) are usually inactive: only 49% of the binaries had a live C2 server by the time they appear in our two sources. Making things worse, many of the live servers became inactive before we finished our experiments. In total, we managed to connect 202 binaries with live C2 servers, which we refer to as Ground3. These binaries generate 230MB of traffic that includes C2 communication and scanning activity.

Observation: *C2 traffic is a small percentage of the overall malware-generated traffic.* In our ground-truth dataset, we find that C2 traffic is only 14.8KB or 0.06% of the total traffic. This might initially appear counter intuitive and in contrast to our initial insight that a higher frequency of packets to a particular IP:port indicates C2 activity; however, our initial insight correctly applies here because the other non-C2 traffic (99.94%) are to a variety of different targets but mostly with single (or a few) packets to each one. As a reminder, our insight compares the number of all packets to a particular target compared to the same number for all other targets in the traffic.

A note on malware family coverage: Across our datasets, we find binaries of 11 different malware families (see Table 2). Classifying a binary according to family accurately is non-trivial. We used AVClass2 [50] for labeling, but we found the AV engine labels for MIPS samples to be highly inaccurate. For example, all the instances of the Mozi family, a peer-to-peer (P2P) malware for which we observed the corresponding traffic in our analysis, are wrongly classified as Mirai. Thus, we used crowd-sourced YARA rules (provided by VirusTotal results) in addition to AVClass2 to identify the malware family labels. In particular, we used the YARA rules for P2P, XORed (variants of Mirai that use encryption, such as Fbot, Apep and Sora), Daddyl33t and VPNFilter. We do not analyze P2P binaries (mainly Mozi) further in this work, as we focus on C2 communication, which are typically absent in P2P malware.

5.3 Q1: Traffic Disambiguation Precision

In order to answer Q1, we evaluate the precision of our traffic disambiguation, namely our ability to differentiate between C2-bound to other traffic generated by the malware. Among all the traffic generated by the binary, only a subset of it is sent to its C2 server. Other traffic could be towards, say a benign victim IP, as part of its proliferation attempt of the malware. In fact, the malware can even contact a random IP address to mislead detection attempts. For example, some Mirai binaries contact 65.222.202.53 when the appropriate activation key is not provided at run time. Interestingly, the IP address obtained "notoriety," as it was arguably owned by the NSA [25], and featured in an xkcd comic (https://xkcd.com/1247).

Ground truth: In the absence of a benchmark, we created the `Ground1` dataset that has IoT malware samples paired with their C2 server address (IP or DNS name) and port. We created this dataset by manually analyzing IoT malware samples and the traffic they generate to find their C2 servers. We followed a three step process. First, we create network-level signatures based on the source code of IoT malware. Second, we find those signatures via manual reverse engineering in the `Ground1` binaries (see Appendix D). Third, we analyze the destination receiving the signature communication, and we consider target addresses (either IP:port or a DNS name) as a C2 server. We find that only 13% of the binaries in the `Ground1` dataset use DNS-based C2 addresses, and the rest are IP:port-based.

Result: We disambiguate C2-bound traffic with 90% precision. We evaluate our algorithm using a fine-grained and a coarse-grained method. In our fine-grained evaluation, for every binary in the `Ground1` dataset, the algorithm returns the target that is most likely the C2 server. We compare this result with the manually found C2s (explained above) and found that C2Miner has a precision of 90% in correctly finding the C2-bound traffic on the `Ground1` dataset. In our coarse-grained evaluation, we find the C2s of binaries in `Ground2` using C2Miner and then query VirusTotal. If VirusTotal reports the C2 address as malicious, we count the finding as accurate. It turns out that the precision based on this method is the same as the one with the fine-grained method in our dataset.

Family	Not Packed	UPX	Modified UPX	Total
Mirai	449 (59.63%)	273 (36.25%)	31 (4.12%)	753
Gafgyt	227 (82.55%)	22 (8.00%)	26 (9.45%)	275
Xored	224 (98.25%)	4 (1.75%)	0 (0.00%)	228
P2P	0 (0.00%)	70 (100.00%)	0 (0.00%)	70
Bash	3 (100.00%)	0 (0.00%)	0 (0.00%)	3
Dakkatoni	0 (0.00%)	28 (100.00%)	0 (0.00%)	28
Tsunami	9 (75.00%)	0 (0.00%)	3 (25.00%)	12
Lightaidra	53 (100.00%)	0 (0.00%)	0 (0.00%)	53
Daddyl33t	10 (100.00%)	0 (0.00%)	0 (0.00%)	10
VPNFilter	2 (100.00%)	0 (0.00%)	0 (0.00%)	2
Hajime	13 (100.00%)	0 (0.00%)	0 (0.00%)	13
Total	990 (68.42%)	397 (27.44%)	60 (4.15%)	1,447

Table 2: The 11 malware families and the number of binaries with different packing techniques in our `DAll` dataset.

We looked into why our traffic disambiguation failed for some samples and identified two reasons: (a) some samples employ evasion techniques that require a command line argument for activation, and in absence of the command, they mislead us by regularly communicating with a benign address, and (b) some samples do not start C2 communication within our 3 minutes analysis timespan.

Would static analysis of the binary reveal the C2 server? We answer this question to gauge the value of our dynamic analysis. It turns out that static analysis would provide the C2 server IP address only for 30% of the binaries in `Ground1`. Note that we first unpack the binaries using UPX and then we use Balbuzard [20] for string analysis as it cracks obfuscation such as XOR and ROL.

Bonus: Some of the malware-specified C2 servers were not known. C2Miner is able to find malware-specified C2 servers that were not known by threat intelligence platforms. VirusTotal is a widely-used aggregator of threat intelligence combining the intelligence feeds of 91 vendors. For IP:port-based C2 targets, out of the live C2 servers that C2Miner finds, 17% are reported as not malicious by VirusTotal. Intrigued, we kept querying VirusTotal and we found that this number drops to 3% after 4 weeks. This drop shows that we identified C2 servers (a) correctly, and (b) earlier than the threat intelligence feeds provided by VirusTotal.

5.4 Q2: Server Determination Accuracy

Here, we evaluate our ability to determine correctly whether a probed target is an active C2 server. Note that targets are an input to the problem and they do not have to be C2 servers.

Ground truth: The required ground truth here is uncommon. We need a data-trace of malware binary interactions with (a) real C2 servers, and (b) benign web and application servers. The goal of the algorithm is to tell the two apart.

As far as we know, such ground truth does not exist, therefore, we created our own `Trace-1` dataset. First, we started from samples with live C2 server from the `Ground3` dataset. These samples are from 8 of the 11 families listed on Table 2. Second, we selected /23 subnets from the C2 server of these samples; this was to simulate the real application of C2Miner in practice. Third, we performed a lightweight SYN stealth scanning to find the live services within those subnets. Fourth, we redirected the C2 traffic (only safe "Call-Home" requests) to all live services in those subnets. Finally, we manually analyzed the contents of the traffic and decide what service the targets host. We communicated with 200 live services and receive responses containing a data payload from roughly 37% of these live services. The live services mainly hosted HTTP servers (Apache, Nginx, OrgaMon), but we also found OpenSSH, MySQL, ESMTP and IMAP servers. The rest were communications with live C2 servers. In total, we collected malware-redirected traffic to 34 C2 and 39 benign servers. The C2 servers and their paired binaries are from the Gafgyt, Mirai and Xor families. Although the diversity may seem low, as we mentioned earlier, the types of C2 servers that are live at the time of experimentation is out of our control.

We determine C2 servers with 86% F1 score. We evaluate the effectiveness based on the precision, recall and the F1 score. We evaluate two C2 determination approaches: (a) *SYN-DATA-aware* (§3.4), and (b) *Fingerprinting-aware* (§4). As a reference, we use a simple baseline, which assumes that a target is a C2 server, if it responds with any data packet. Figure 3 summarizes our results.

a. Not everything that is "alive" is a C2 server. The baseline approach is overly "aggressive" but it shows an interesting phenomenon. It finds all C2 server, but at the cost of poor precision (44%): it also identifies benign servers as C2 servers. The 100% recall is expected because C2 servers respond with data packets to malware requests.

b. Achieving 100% precision with moderately reduced recall. Our SYN-DATA-aware method has 100% precision but only 62% recall. The fingerprinting method, that is assessing the target based on an expected communicated pattern, has the same precision with an improved recall of 79%. Overall, our C2 determination based on fingerprinting has a 86% F1 score.

Troubleshooting: We analyze the errors (FNs and FPs) of the two approaches. With SYN-DATA-aware, there are cases where the application does not close the socket even in the case of error leading to a false positive. With fingerprinting, there are cases where a legitimate service communication looks similar to a malware communication. An example is a benign IRC server, as this protocol is also used by some malware families (as shown in Appendix B).

5.5 Q3: Clustering and Fingerprinting

Evaluating the quality of our proposed clustering approach, and the clusters, ultimately depends on the application scenario. Here, we answer Q3 and evaluate the clustering quality based on how well we can distinguish between different malware families. The idea is to first cluster the samples based on their communication patterns, and then see if the clusters correspond to malware families.

Ground truth: We require a traffic dataset of labeled malware samples' communications with their C2 servers. As we stated before, this dataset does not exist, and hence, we created DFinger. Note that an already existing labeled malware dataset (with family labels) would not be useful because the malware should have a live C2 server so we can collect the communicated traffic. The challenge is obtaining traffic with C2 server communications for IoT malware.

We created the DFinger dataset containing 202 binaries and their fingerprints starting from the binaries in Ground3; binaries in Ground3 have a C2 live server and we were able to engage with their malware-specified C2 during the data collection. In the DFinger dataset, each binary is associated with (a) the family label, as we explain below, and (b) a series of our grammar-based fingerprints of the communication with its C2 server that we captured by collecting samples on a daily basis, and storing the raw traffic that contains communication with live C2 servers. We then transformed the traffic to strings in our grammar as shown in §4. To determine the malware family, we combined AVClass2 [50] and YARA, as discussed in §5.2 and summarized in Table 2.

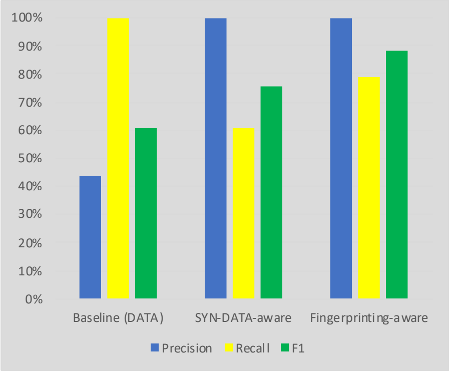

Figure 3: C2 determination accuracy on the Trace-1 dataset.

	Mirai	Gafgyt	Daddyl33t	Xored	Light/ra	Hajime	Tsunami
#1	25%	**68%**	20%	34%	0%	50%	0%
#2	**46%**	0%	40%	**66%**	14%	0%	0%
#3	18%	26%	40%	0%	**86%**	0%	0%
#4	11%	6%	0%	0%	0%	50%	**100%**

Table 3: Clustering effectiveness: the clusters are reasonably aligned with malware family labels in the DFinger dataset.

Result: We cluster binaries into malware families based on their communication patterns. We hierarchically cluster the C2 traffic based on the method that we explain in §4. We choose the seeds randomly, and we observe that the choice of k (in the k-means algorithm) has minimal effect for $k \geq 5$; they always converge to the same four clusters; thus we select $k = 5$. The malware family labels of the four clusters is shown in Table 3. The clustering seems to capture the behavior of malware families reasonably well, but not perfectly. For example, the majority of Gafgyt, XORed, Lightaidra and Tsunami fall into clusters #1–#4, respectively. At the same time, Mirai is spread among all four clusters. Roughly half of the Mirai samples fall into #2 that is also the dominant cluster for XORed. This makes sense as XORed consists of variants of Mirai with encryption. Our clustering result has an important implication which is: *using a weaponized sample of one malware family, we can probably find the C2 server of multiple malware families.*

What do the results suggest? The majority of the samples of the same malware family use the same communication protocol although there are exceptions, which we reviewed in more detail:

The three patterns of Daddyl33t: We observe three distinct communication patterns (corresponding to clusters #1, #2 and #3) when studying their fingerprints. Interestingly, we find one C2 server that responds to patterns 1 and 3, which further indicates that these are communication patterns of the same malware.

Pattern in cluster #1: The malware connects to the server and sends 18 bytes in plaintext with the message "VER:3:unknown." The server responds with a single byte followed by another 4 bytes. The binary closes the connection, and repeats the same thing again and again.

Pattern in cluster #2: The malware connects to the server and send in plaintext "`VER:0:/malware`." The server responds with 516 bytes of data. The binary does not close the connection.

Pattern in cluster #3: The malware connects to the server and sends information about the compromised device (e.g., architecture and endianness). The binary then keeps the connection open.

The two patterns of Hajime: We observe two distinct communication patterns (corresponding to clusters #1 and #4). Note that Hajime is a P2P malware that also seems to have C2 communication, which is not uncommon: new variants of Mozi appear to have dedicated C2 servers for DDoS commands in addition to the P2P infrastructure [54]. We identify two patterns of C2 communication in Hajime. In the first pattern, the client connects to the server, and receives 1314 bytes of data and then closes the connection with a FIN flag. These 1314 bytes are the same for different sessions and different malware samples. In the second pattern, the malware connects to the server and receives 1448 bytes of data.

Summary: Our hierarchical clustering results in 215 clusters. One interesting observation is the common patterns that appear in communications: Most Mirai samples share the `CLIENT_4.CLIENT_1.CLIENT_2.SERVER_2` pattern, while most Gafgyt samples share `SERVER_4.SERVER_1.SERVER_4.SERVER_1`. With just two patterns, we can detect 68% of C2 communication.

5.6 Q4: Quantifying Cross-Talk

Here, we want to quantify the cross-talk, namely the ability of a binary to talk to a plurality of servers. Our initial analysis of publicly available IoT malware source code suggests this is true (see Appendix B), nevertheless, we empirically assess this hypothesis. Our assessment has two goals. First, we want to see whether the MitM module functions in practice. For example, reasons for failure could involve encryption at the IP layer, or active efforts to bypass the kernel-level networking. Second, we want to see whether old binaries can talk to current C2 servers.

Ground truth: For this evaluation, we need a dataset of live C2 servers and the samples with which they can communicate. In absence of such a dataset, we created the `Trace-2` dataset from our collected binaries as follows. In more detail, we started with the list of malware with live C2 addresses in `Ground3` as they are collected on a daily basis. We first find the malware-specified C2 target of the malware. Second, we want to find other binaries that can communicate with these new servers. To do this, we find binaries with similar C2 communication behavior using our fingerprinting which we described earlier. Third, we checked if the binaries of the previous step communicate with the server successfully, which we validated with manual inspection. Our manual evaluation method was similar to what we described in §5.3 for the creation of ground truth. The number of binaries in `Trace-2` was limited since many C2 servers were short-lived, and it was not always possible to find compatible binaries to engage with them in time.

Result: Our MitM capability works for 84% of sample pairs. To measure the success our MitM approach in practice we count the number of sample pairs that can successfully talk to the designated candidate live C2 server for the malware cluster. A high rate of success indicates that one sample from a cluster is good enough to find most C2 servers for the entire cluster. We use the fingerprinting-based C2 determination. We find that 84% of sample pairs can successfully cross-communicate with the C2 of the cluster.

6 CASE STUDY: FINDING LIVE C2 SERVERS

How can we use C2Miner in practice to find live C2 servers? This is the what we answer in this section. Finding C2 servers in the entire IP:port space using malware weaponization even with a fully working solution is challenging. This problem is two fold. First, the IP port space is 2^{48} bits large, and we can not send probes to the entire space. Second, we do not know what malware binary sample should be chosen for a given space. Bruteforcing over the space and malware binaries requires massive computation power and may disrupt the Internet. Instead, we showcase the value of C2Miner with a proof-of-concept deployment that addresses the problems described above by carefully selecting binaries and IP:port space.

We efficiently select malware for probing based on our fingerprinting approach. We select two old malware binaries: (a) one from the Mirai family, and (b) one from the Gafgyt family as shown in Table 4. We chose these two samples because their communication fingerprint, based on our grammar-based clustering (see §5.5), is "close" to 68% of the samples. These samples are at least six months old at the time of our experiments. We would like to point out that adding more samples will linearly increase the complexity of deployment. Due to the computation resource constraints and ethical reasons, we intentionally keep the case study size small.

We efficiently select IP port space for probing. We made the following choices for the target IP:port space. We identified six /24 subnets and 12 "popular" ports as we explain in Appendix E based on the increased likelihood of observing malicious activity from past C2 hosts [11, 16]. We conduct roughly 331K probes across 18K IP:port combinations for 1,536 IP addresses and 12 ports with three probes per day for six days. For efficiency, we perform our probing in two phases. First, we use a light-weight SYN stealth scan using MASSCAN [26] to establish liveness. Once this is established, we deploy the more resource-consuming C2Miner probing.

We find active C2 servers on a daily basis. We employ C2Miner's probing mode in practice for six days. The results are very promising: we find a total of six active C2 servers for our binaries (see last row of Table 4): five using Gafgyt and one using Mirai. We can not say much about the identity features (including their malware family) of the found C2 servers because we do not have access to those servers, and we only have the communicated traffic between them and our weaponized malware binaries. We can only say that their C2 communication protocols resemble that of Gafgyt and Mirai. This was expected given the similarity of communications across different malware families that we show in §5.5.

We plot the number of active servers each day and the new distinct servers discovered that day In Figure 4. We argue that the performance to cost ratio is high. Despite the limited scope of our study, we identified one live server a day. We consider this fairly remarkable given that we used only two malware binaries, and the heuristic approach in finding IP space with likely C2 servers.

Intrigued by the seemingly high temporal variability of the C2 servers, we further investigate the temporal characteristics of C2

Parameter	Values
Subnets	136.144.41/24, 195.133.40/24, 2.58.149/24, 212.193.30/24, 107.173.176/24, 45.95.169/24
Ports	1312, 666, 1791, 9506, 606, 6738, 5555, 1014, 3074, 6969, 42516, 81
Sample(s)	Gafgyt 46501d723f368c22e5401f7c95d928ab
Sample(s)	Mirai 800af659256f0232a27f955a4430aed0
Live C2 Servers	2._._.34:5555, 212._._.91:666, 45._._.119:666, 136._._.240:666, 212._._.123:5555, 107._._.144:42516

Table 4: Parameters of our probing study conducted in January 2022: (a) 6 /24 subnets, (b) 12 ports in order of "popularity," and (c) 2 malware samples. The last row lists the 6 live C2 IP:ports (obfuscated for privacy reasons) we found.

servers during our probing. More specifically, we focus on the daily liveliness of the found C2 servers, which we show in Figure 5. Our key observation here is that that live C2 servers response rate is low even for bots of their family. Our results suggest that even a live server will not respond consistently to a bot request. We count the percentage of successful responses under two different assumptions. First, the success rate of a probe is 15% if we assume that the servers are alive for all six days. Second, the success rate is 31%, if we assume that a server is alive only during the days of its first and last response to our probes.

In summary, we found C2 servers that without a MitM approach would not have been discovered even if one had activated these two malware in a sandbox, as the malware-specified C2 servers (i.e., the ones embedded in these particular binaries without any fallback mechanism) are no longer active.

7 DISCUSSION

a. Will the use of encryption neutralize C2Miner? We consider two cases: encryption at the application layer (VII), and encryption at the III/IV layer(s). First, the use of commonly used TLS protocol does not break our solution because the server does not authenticate the clients (the malware binaries). Nevertheless, there are ways to defend against C2Miner if the malware authors knowingly try to do so. A more sophisticated attacker can extend the two-way authentication or mutual authentication schemes, such as mTLS, to only allow a one-to-one communication which would create problems for our approach. However, such an extreme measure will increase the complexity of implementation for malware authors, because each malware binary will only work with one C2 server and vice versa. Second, if the malware uses encryption at the TCP/IP layer, our solution will not work. However, the use of encryption at this layer is computationally expensive and because of that it is not commonly used. In fact, many related security solutions will not work in the presence of such an encryption.

b. Will the use of obfuscation neutralize C2Miner? We discuss two potential ways of obfuscation that malware authors can consider. First, malware authors can try to detect the MitM at the application layer. A possible solution is the inclusion of the original C2 address (before it is changed by us) in the payload so that the C2 server can detect the layer III/IV addresses are modified. However, such a defense also affects the use of DNS addresses by the malware and limits the C2 server mobility. In addition, any countermeasure like this which is only applied to the malware binary is prone to yet another counter-countermeasure by us because we have full access to the binary and the sandbox. Second, there is a risk that malware authors knowingly try to defeat our C2 determination technique. For instance, they might try to connect (at the III/IV layer) to the C2 server only once. Or, they might try to chunk the traffic and send it to multiple addresses in separate connections. We argue that any attempt to disguise the current behaviors we rely on, such as repeated trials for re-connection, will lead to an anomaly which is different from how benign servers behave. Henceforth, a counter solution is likely to exist as long as the malware behavior is different from the benign servers, which is true because malware authors can not change the behavior of benign servers. Overall, security is an arms race, and a security approach is successful, if it forces attackers to modify their behavior, especially, if the modification is non-trivial, as is the case here.

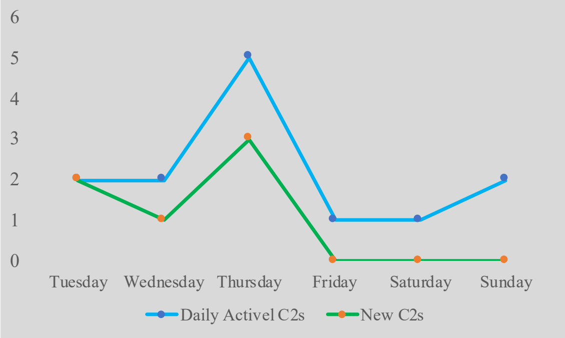

Figure 4: Number of responsive C2 servers per day and number of distinct new C2 servers during our six day probing over 1,536 IP addresses across six subnets and 12 ports.

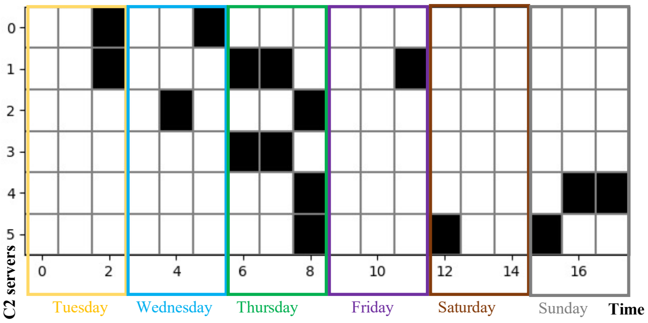

Figure 5: The responsiveness of the six responsive C2 servers over time to our three daily probes which is 15% per probe assuming they are alive for the full duration (six days).

c. Can we generalize our fingerprinting approach to other applications? We believe we use generic properties of network systems to design our SYN-DATA aware and grammar-based fingerprinting. Specifically, our SYN-DATA aware solution relies on the low level implementation of the transport protocol rather than the application-level peculiarities of the C2 servers. In addition, our grammar-based solution can provide a valuable embedding of the network traffic between any client and server. Again, our grammar does not depend on C2 implementations at the application layer.

d. Can the C2Miner output be used in Intrusion Detection Systems? The fingerprinting strings that represent C2 clusters can also be used in Intrusion Detection Systems (IDS) like Snort and Surricata. For converting these strings into IDS rules, the *flowbits* functionality of these systems can be used to tag individual packets following our grammar rules. Then, the *isset* operator can be used to compare fingerprints of flows. The Jaccard distance function can be used to identify C2 traffic.

e. Are the evaluation results representative? We believe our evaluation reflects the general performance of our approach, and the intrinsic nature of the problem. The size of our evaluation datasets is constrained by (a) the scarcity of live C2 servers in our collection, (b) the lack of benchmark datasets, and (c) the manual effort required for establishing the ground truth. That said, our datasets are comparable or larger than the datasets used for similar studies [23, 42]. We continue to collect data and we will share the results with the community (see Appendix F).

f. What if the malware does not activate? This is a concern for any work that relies on dynamic analysis [39]. Our intention is to use the latest sandboxing technology, as this is not the focus of our work. Recent studies [15] suggests that IoT malware is not yet as sophisticated as malware targeting other platforms. Although this may change, we hope that sandbox techniques will also evolve. In addition, note that we do not need to activate all binaries, as long as we activate enough binaries that "talk" to most types of C2 servers.

8 RELATED WORK

So far, no related work has focused on the problem as framed here: finding live C2 servers for an unknown binary and within a given target IP:port space. To the best of our knowledge, we are the first to redirect the communication of a binary using a MitM approach to "reconnect" old samples with currently live C2 servers.

a. IoT malware analysis. Studying IoT malware has become a hot topic both for academia and industry. First, many efforts focus on characterizing the behavior of a single malware family [5, 27, 30]. These works characterize infected IoT devices, infection vectors, and the life-cycle of the malware. While these studies provide a deep insight into a single malware family, they are less likely to lead to generalizable methods for detecting C2 servers dynamically. Another line of related work characterizes the behaviors of several malware families at the same time [2, 3, 13–15, 18, 40]. Some recent studies summarize the landscape of DDoS activity and highlight the contributions of IoT-based attacks and provide countermeasures [4, 24, 32], while honeypot efforts collect attack data [33]. Finally, several studies analyze C2 server communication from a networking point of view [3, 52, 55]. These studies focus on understanding the infrastructure that supports its operation, and profiling aspects of malware behavior, but do not engage in active probing.

b. Active probing of malware. The most relevant studies to our work focus on active probing. Such efforts require an understanding of malware communication protocols and encryption algorithms, if any. Assuming that this can be accomplished effectively, searching for C2 servers of a single malware family becomes easier [5, 22]. Prior work also took first steps in automating the active probing for a more widespread group of malware families [43, 59]. The main problem with these approaches is that they cannot work well in the presence of encryption. By contrast, we sidestep this issue as we let the activated binary "speak" to the server directly.

c. Engaging with malware. Prior work has attempted to engage with malware to understand different aspects of its behavior and mitigate its impact [8, 23, 42, 45]. GZA [42] and Squeeze [45] try to reveal desktop malware's alternative methods of communicating with C2 servers. In contrast, IoT botnets are disposable and hence alternative addresses are not a broad phenomena [52]. Botacin et al. [8] find the domains contacted by the malware by analyzing the sample in a sandbox using a technique that we will consider in our future work. C3PO [23] develops techniques to detect and spoof bot-to-C2 communications, but is only applicable to malware with over-permissioned protocols and does not include active probing.

d. Network traffic modeling and analysis. Modeling network traffic enables the identification of particular protocols or activities. First, several works try to infer the application layer protocol [10, 12, 57]. However, this is not effective in the presence of encryption, computational intensive, and error prone due to the complexity of the task. Second, other efforts [19, 28, 29, 37] study captured traffic in order to extract the behavioral patterns. Thus, they need a network trace, and they can only find server behaviors whose flows are in that trace, which is a different problem formulation.

e. Signature Generation and Fingerprinting. This group of work is closely related to our proposed fingerprinting approach. Several related efforts focus on malware network traffic [7, 21, 46–48, 53, 58, 61] and employ clustering by using features that include IP, port, timestamps, number of bytes, connection duration, direction of the connection etc., which are more difficult to generalize. Some of these approaches also use application layer protocol features [46, 47]. For most of these approaches, the payload content plays an import role and the presence of strong encryption weakens the effectiveness of these solutions. ProVeX [48] tries to probabilistically compile the fingerprint, instead of using strings or regular expressions, but it also requires access to the decrypted training traffic. The most relevant work to ours is CoCoSpot [21] where for, arguably, the first time, the communication is treated as a "dialogue." The approach uses a fixed vector of the first eight messages with features. In contrast, we explained the novelty our method in §1 and §3.

9 CONCLUSION

We propose C2Miner, a novel approach to trick arbitrary malware binaries to reveal their currently live C2 servers. The novelty of our approach is that we use the malware as a spy to reveal its whole family live C2 servers within a selected IP:port space. To substantiate this vision, we develop techniques to: (a) disambiguate the C2-bound traffic with 92% precision, (b) determine if a target IP:port is indeed a C2 server with an F1 score of 86%, and (c) fingerprint and cluster C2 communications effectively. We showcase a proof-of-concept deployment of this approach to show its promise.

Our approach is a fundamental step towards identifying live C2 servers on demand given a binary. This capability is even more critical for IoT malware that is an emerging battleground for cybercrime. A large-scale deployment of our approach using a large number of binaries and scanning substantial swaths of the Internet can arguably become a game-changing capability in identifying C2 servers and containing the scourge of botnets.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their valuable comments and input for improving the paper. The work was supported partly by the NSF SaTC 2132642 grant, the Vienna Science and Technology Fund (WWTF) and the City of Vienna [10.47379/ICT19056], and SBA Research (SBA-K1), a COMET Centre within the framework of COMET – Competence Centers for Excellent Technologies Programme and funded by BMK, BMDW, and the federal state of Vienna. The COMET Programme is managed by FFG.

REFERENCES

[1] Abuse.ch. 2024. MalwareBazaar. https://bazaar.abuse.ch/.

[2] Arwa Abdulkarim Al Alsadi, Kaichi Sameshima, Jakob Bleier, Katsunari Yoshioka, Martina Lindorfer, Michel van Eeten, and Carlos H Gañán. 2022. No Spring Chicken: Quantifying the Lifespan of Exploits in IoT Malware Using Static and Dynamic Analysis. In *Proceedings of the ACM ASIA Conference on Computer and Communications Security (ASIACCS)*.

[3] Omar Alrawi, Charles Lever, Kevin Valakuzhy, Ryan Court, Kevin Snow, Fabian Monrose, and Manos Antonakakis. 2021. The Circle Of Life: A Large-Scale Study of The IoT Malware Lifecycle. In *Proceedings of the USENIX Security Symposium*.

[4] Radu Anghel, Swaathi Vetrivel, Elsa Turcios Rodriguez, Kaichi Sameshima, Daisuke Makita, Katsunari Yoshioka, Carlos H. Gañán, and Yury Zhauniarovich. 2023. Peering into the Darkness: The Use of UTRS in Combating DDoS Attacks. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*.

[5] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. 2017. Understanding the Mirai Botnet. In *Proceedings of the USENIX Security Symposium*.

[6] Fabrice Bellard. 2005. QEMU, A Fast and Portable Dynamic Translator. In *Proceedings of the USENIX Annual Technical Conference (ATC, FREENIX Track)*.

[7] Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. 2012. Disclosure: Detecting Botnet Command and Control Servers Through Large-Scale NetFlow Analysis. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*.

[8] Marcus Botacin, Paulo de Geus, and André Grégio. 2020. An Empirical Study on the Blocking of HTTP and DNS Requests at Providers Level to Counter In-The-Wild Malware Infections. In *Anais do XX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*.

[9] Xander Bouwman, Harm Griffioen, Jelle Egbers, Christian Doerr, Bram Klievink, and Michel van Eeten. 2020. A Different Cup of TI? The Added Value of Commercial Threat Intelligence. In *Proceedings of the USENIX Security Symposium*.

[10] Juan Caballero, Pongsin Poosankam, Christian Kreibich, and Dawn Song. 2009. Dispatcher: Enabling Active Botnet Infiltration using Automatic Protocol Reverse-Engineering. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*.

[11] M. Patrick Collins, Timothy J. Shimeall, Sidney Faber, Jeff Janies, Rhiannon Weaver, Markus De Shon, and Joseph Kadane. 2007. Using Uncleanliness to Predict Future Botnet Addresses. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement (IMC)*.

[12] Paolo Milani Comparetti, Gilbert Wondracek, Christopher Kruegel, and Engin Kirda. 2009. Prospex: Protocol Specification Extraction. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*.

[13] Emanuele Cozzi, Mariano Graziano, Yanick Fratantonio, and Davide Balzarotti. 2018. Understanding Linux Malware. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*.

[14] Emanuele Cozzi, Pierre-Antoine Vervier, Matteo Dell'Amico, Yun Shen, Leyla Bilge, and Davide Balzarotti. 2020. The Tangled Genealogy of IoT Malware. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*.

[15] Ahmad Darki and Michalis Faloutsos. 2020. RIoTMAN: A Systematic Analysis of IoT Malware Behavior. In *Proceedings of International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*.

[16] Ali Davanian. 2017. *Effective Granularity in Internet Badhood Detection: Detection Rate, Precision and Implementation Performance.* Master's thesis. University of Twente.

[17] Ali Davanian, Ahmad Darki, and Michalis Faloutsos. 2021. CnCHunter: An Mitm-Approach To Identify Live CnC Servers. In *Black Hat USA*.

[18] Ali Davanian and Michalis Faloutsos. 2022. MalNet: A Binary-Centric Network-Level Profiling of IoT Malware. In *Proceedings of the ACM Internet Measurement Conference (IMC)*.

[19] Lorenzo De Carli, Ruben Torres, Gaspar Modelo-Howard, Alok Tongaonkar, and Somesh Jha. 2017. Botnet Protocol Inference in the Presence of Encrypted Traffic. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*.

[20] decalage2. 2019. Balbuzard - Malware analysis tools to extract patterns of interest and crack obfuscation such as XOR. https://github.com/decalage2/balbuzard.

[21] Christian J. Dietrich, Christian Rossow, and Norbert Pohlmann. 2013. CoCoSpot: Clustering and Recognizing Botnet Command and Control Channels using Traffic Analysis. *Computer Networks* 57, 2 (2013).

[22] Brown Farinholt, Mohammad Rezaeirad, Paul Pearce, Hitesh Dharmdasani, Haikuo Yin, Stevens Le Blond, Damon McCoy, and Kirill Levchenko. 2017. To Catch a Ratter: Monitoring the Behavior of Amateur DarkComet RAT Operators in the Wild. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*.

[23] Jonathan Fuller, Ranjita Pai Kasturi, Amit Sikder, Haichuan Xu, Berat Arik, Vivek Verma, Ehsan Asdar, and Brendan Saltaformaggio. 2021. C3PO: Large-Scale Study Of Covert Monitoring of C&C Servers via Over-Permissioned Protocol Infiltration. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[24] Carlos H. Gañán. 2023. Unravelling the Changing Landscape of DDoS Attacks: The Role of IoT Botnets. In *ICANN DNS Symposium*.

[25] Sean Ghallager. 2013. In face of scrutiny, researchers back off NSA "Torsploit" claim. https://arstechnica.com/tech-policy/2013/08/in-face-of-scrutiny-researchers-back-off-nsa-torsploit-claim/.

[26] Robert David Graham. 2021. MASSCAN: Mass IP port scanner. https://github.com/robertdavidgraham/masscan.

[27] Harm Griffioen and Christian Doerr. 2020. Examining Mirai's Battle over the Internet of Things. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[28] Guofei Gu, Vinod Yegneswaran, Phillip Porras, Jennifer Stoll, and Wenke Lee. 2009. Active Botnet Probing to Identify Obscure Command and Control Channels. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*.

[29] Guofei Gu, Junjie Zhang, and Wenke Lee. 2008. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*.

[30] Stephen Herwig, Katura Harvey, George Hughey, Richard Roberts, and Dave Levin. 2019. Measurement and Analysis of Hajime, a Peer-to-peer IoT Botnet. In *Proceedings of the Network and Distributed Systems Security Symposium (NDSS)*.

[31] KimiNewt. 2023. pyshark - Python wrapper for tshark. https://github.com/KimiNewt/pyshark/.

[32] Mizuki Kondo, Rui Tanabe, Natsuo Shintani, Daisuke Makita, Katsunari Yoshioka, and Tsutomu Matsumoto. 2022. Amplification Chamber: Dissecting the Attack Infrastructure of Memcached DRDoS Attacks. In *Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*.

[33] Lukas Krämer, Johannes Krupp, Daisuke Makita, Tomomi Nishizoe, Takashi Koide, Katsunari Yoshioka, and Christian Rossow. 2015. AmpPot: Monitoring and Defending Against Amplification DDoS Attacks. In *Proceedings of the International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*.

[34] Marc Kührer, Christian Rossow, and Thorsten Holz. 2014. Paint it Black: Evaluating the Effectiveness of Malware Blacklists. In *Proceedings of the International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*.

[35] Victor Le Pochat, Tim Van hamme, Sourena Maroofi, Tom Van Goethem, Davy Preuveneers, Andrzej Duda, Wouter Joosen, and Maciej Korczyński. 2020. A Practical Approach for Taking Down Avalanche Botnets Under Real-World Constraints. In *Proceedings of the Annual Network and Distributed System Security Symposium (NDSS)*.

[36] Chao Lei, Zhibin Zhang, and Cecilia Hu. 2023. Mirai IZ1H9 - Mirai Variant Targets Multiple IoT Devices. https://unit42.paloaltonetworks.com/mirai-variant-iz1h9/.

[37] Chaz Lever, Platon Kotzias, Davide Balzarotti, Juan Caballero, and Manos Antonakakis. 2017. A Lustrum of Malware Network Communication: Evolution and Insights. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*.

[38] Vector Guo Li, Matthew Dunn, Paul Pearce, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. 2019. Reading the Tea Leaves: A Comparative Analysis of Threat Intelligence. In *Proceedings of the USENIX Security Symposium*.

[39] Martina Lindorfer, Clemens Kolbitsch, and Paolo Milani Comparetti. 2011. Detecting Environment-Sensitive Malware. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID)*.

[40] Tongbo Luo, Zhaoyan Xu, Xing Jin, Yanhui Jia, and Xin Ouyang. 2017. IoTCandyJar: Towards an Intelligent-Interaction Honeypot for IoT Devices. In *Black Hat USA*.

[41] Yacin Nadji, Manos Antonakakis, Roberto Perdisci, David Dagon, and Wenke Lee. 2013. Beheading Hydras: Performing Effective Botnet Takedowns. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[42] Yacin Nadji, Manos Antonakakis, Roberto Perdisci, and Wenke Lee. 2011. Understanding the Prevalence and Use of Alternative Plans in Malware with Network Games. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*.

[43] Antonio Nappa, Zhaoyan Xu, M. Zubair Rafique, Juan Caballero, and Guofei Gu. 2014. CyberProbe: Towards Internet-Scale Active Detection of Malicious Servers. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*.

[44] National Security Agency. 2023. Ghidra - Software Reverse Engineering Framework. https://www.nsa.gov/resources/everyone/ghidra/.

[45] Matthias Neugschwandtner, Paolo Milani Comparetti, and Christian Platzer. 2011. Detecting Malware's Failover C&C Strategies with Squeeze. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*.

[46] Roberto Perdisci, Wenke Lee, and Nick Feamster. 2010. Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[47] M. Zubair Rafique and Juan Caballero. 2013. Firma: Malware Clustering and Network Signature Generation with Mixed Network Behaviors. In *Proceedings of the International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*.

[48] Christian Rossow and Christian J. Dietrich. 2013. ProVeX: Detecting Botnets with Encrypted Command and Control Channels. In *Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*.

[49] Christian Rossow, Christian J. Dietrich, Chris Grier, Christian Kreibich, Vern Paxson, Norbert Pohlmann, Herbert Bos, and Maarten van Steen. 2012. Prudent Practices for Designing Malware Experiments: Status Quo and Outlook. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*.

[50] Silvia Sebastián and Juan Caballero. 2020. AVClass2: Massive Malware Tag Extraction from AV Labels. In *Procedings of the Annual Computer Security Applications Conference (ACSAC)*.

[51] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. 2009. Your Botnet is My Botnet: Analysis of a Botnet Takeover. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*.

[52] Rui Tanabe, Tatsuya Tamai, Akira Fujita, Ryoichi Isawa, Katsunari Yoshioka, Tsutomu Matsumoto, Carlos Gañán, and Michel Van Eeten. 2020. Disposable Botnets: Examining the Anatomy of IoT Botnet Infrastructure. In *Proceedings of the International Conference on Availability, Reliability and Security (ARES)*.

[53] Florian Tegeler, Xiaoming Fu, Giovanni Vigna, and Christopher Kruegel. 2012. BotFinder: Finding Bots in Network Traffic Without Deep Packet Inspection. In *Proceedings of International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*.

[54] Alex Turing, Hui Wang, and Genshen Ye. 2021. The Mostly Dead Mozi and Its' Lingering Bots. https://blog.netlab.360.com/the-mostly-dead-mozi-and-its-lingering-bots/.

[55] Pierre-Antoine Vervier and Yun Shen. 2018. Before Toasters Rise Up: A View into the Emerging IoT Threat Landscape. In *Proceedings of the International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*.

[56] VirusTotal. 2024. VirusTotal. https://www.virustotal.com.

[57] Zhi Wang, Xuxian Jiang, Weidong Cui, Xinyuan Wang, and Mike Grace. 2009. ReFormat: Automatic Reverse Engineering of Encrypted Messages. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*.

[58] Peter Wurzinger, Leyla Bilge, Thorsten Holz, Jan Goebel, Christopher Kruegel, and Engin Kirda. 2009. Automatically Generating Models for Botnet Detection. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*.

[59] Zhaoyan Xu, Antonio Nappa, Robert Baykov, Guangliang Yang, Juan Caballero, and Guofei Gu. 2014. Autoprobe: Towards Automatic Active Malicious Server Probing Using Dynamic Binary Analysis. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[60] Miuyin Yong Wong, Matthew Landen, Manos Antonakakis, Douglas M. Blough, Elissa M. Redmiles, and Mustaque Ahamad. 2021. An Inside Look into the Practice of Malware Analysis. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[61] Ali Zand, Giovanni Vigna, Xifeng Yan, and Christopher Kruegel. 2014. Extracting Probable Command and Control Signatures for Detecting Botnets. In *Proceedings of the Annual ACM Symposium on Applied Computing (SAC)*.

[62] Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. 2020. Measuring and Modeling the Label Dynamics of Online Anti-Malware Engines. In *Proceedings of the USENIX Security Symposium*.

A ADDITIONAL ETHICAL CONSIDERATIONS

In addition to the steps discussed in §5.1, we employ the following additional measures to contain the malware in our experiments:

a. Detection of C2s: Algorithm 1 (used to detect C2s) is done in isolation. Thus, it does not interact with the Internet, as we "fake" connections to the malware in the sandbox. For sophisticated binaries that check for Internet connectivity, we deploy InetSim to simulate services like DNS and HTTP.

b. Observing C2 traffic: Having identified the signatures of C2 communication in step (a), we filter out any communication of the malware with the outside except connections to the C2 server. We record the C2 traffic, reverse engineer it, and do not allow malware to talk to any other target (except the C2).

c. Probing IP subnets: We only allow harmless C2 communications like "Call-Home" requests to interact with potential C2 servers. We manually analyzed sample traffic traces and we have not found any cases of non-C2 communications. Our target subnets were small /24 subnets with a history of malicious activity. We do not send additional probes if the host does not listen on a port. On live ports, we filter out hosts that present a well-known banner (such as Apache or Nginx).

B IOT MALWARE C2 PROTOCOLS

We conducted a small-scale manual study to motivate our work and gain a basic understanding of IoT malware communication protocols, i.e., the interaction between a bot and its C2 server. In this exploratory study, we analyzed the communication protocols of well-known IoT malware families based on their source code found on GitHub. Table 5 shows a summary of the application layer protocols for each family.

a. The bad news. Communication protocols vary significantly between families. A binary of one family will most certainly fail to interact with a C2 server of another family.

b. The good news. The protocol tends to remain nearly identical for binary samples of the malware family. This implies that old malware samples could potentially be used to scan the Internet for new live C2 servers of the same family, which we we also corroborated in §6.

These insights suggest that C2Miner could work reasonably well in practice as long as we have access to arbitrary binaries of a specific family of interest, even if these binaries are not the latest versions. Our clustering efforts also support this (see §5.5).

Malware	Communication	Details
Gafgyt	Custom	PONG command is communicated via IRC, others are text commands.
Mirai	Custom	All C2 commands are custom binary based.
Lightaidra	IRC	All C2 commands are wrapped inside IRC PRIVMSG (private) messages.
Remaiten	IRC	Similar to Lightaidra but commands are different.
Lizkebab	Custom	Similar to Gafgyt but commands are different.
LuaBot	Encrypted Payload	Uses MatrixSSL lib for encryption.
Tsunami	IRC	All C2 commands are wrapped inside IRC NOTICE messages.
BASHLIFE	Custom	Similar to Gafgyt but commands are different.

Table 5: Application layer communication protocol of eight well-known IoT malware families based on their source code.

C VIRTUALIZATION ENVIRONMENT

As discussed in §3.1, for the malware execution we rely on QEMU [6] and RiotMan [15] for which we provide details below. Table 6 provides the breakdown of the additional engineering effort to implement C2Miner on top of QEMU and RiotMan.

Type	Breakdown	Lines of Code (LOC)
Programming Language	Shell	636
	Python	2,897
C2Miner Module	Sandbox	1,239
	MitM/Probing	553
	Profiler	1,231
	Other	510

Table 6: LOC breakdown of the C2Miner implementation.

QEMU emulates the execution of an input executable on a host machine. The emulated executable can be a user program or an entire virtual machine image. In the latter case, the running virtual machine on the host is called the guest VM. We emulate the execution of a guest virtual image (playing the role of an infected victim) on QEMU. QEMU emulates the execution of a guest VM in an architecture agnostic way. This means that the guest might need a different CPU architecture from what the host is running on e.g. a MIPS 32 guest running on a host x86 CPU. In our case, we run a customized virtual image based on Busybox prepared by RiotMan on a x86 host.

RiotMan finds the right virtual environment configuration for an IoT malware executable. To do so, it performs iterative learning in order to configure the environment for the malware execution. It starts with a clean Linux operating system (OS) image that will host the IoT malware. RiotMan executes the malware multiple times until it succeeds without any error during the execution. If there is an error, RiotMan looks up the last system call and finds the file name (if any) that resulted in the error. The file is looked up in a database of common IoT firmware files, and missing files (requested by the malware) are copied to the OS. If the malware needs a file and it is not found in this database, RiotMn creates a file with the expected name in the path that the system call requested. RiotMan iteratively fixes the errors until the malware can successfully run. We use an image prepared by RiotMan for our IoT malware activation.

D MANUAL REVERSE ENGINEERING

For building the ground truth to evaluate the precision of our traffic disambiguation in §5.3 we manually reverse engineered IoT malware binary samples to extract their C2 servers. We started by analyzing the source code of the IoT malware listed on Table 5. We found C2 communication pattern signatures by analyzing the source code. Next, we extracted traffic signatures from reverse engineered malware binaries using Ghidra [44]. If the samples were packed (we only found UPX packing), we unpacked them first. Then, we examined the reversed source code of the malware (when Ghidra successfully processed them) or the disassembly code of the malware to find the communications malware would do with the outside. Then, we compared the malware sample binaries signatures (found through reverse engineering) with those found during source code analysis. The matches are the signatures of communications with the C2 server. Finally, we found the destination addresses the malware tries to send to those signatures. These destinations are the C2 server addresses.

E SELECTION OF PROBING TARGETS

For our case study in §6 we utilize the spatial properties of C2 servers to identify our own promising target space for probing to showcase the potential of our approach. We study the IP-space spatial properties of 367 malware-specified C2 addresses cross-verified by VirusTotal in order to identify locality patterns. These are a subset of `Ground1` that we identified before January 2022.

a. Selecting ports: First, we observe that 12 ports are used by half of the C2 servers. Figure 6 shows the distribution of ports. Using this insight, we focus our probing to these frequently-used ports.

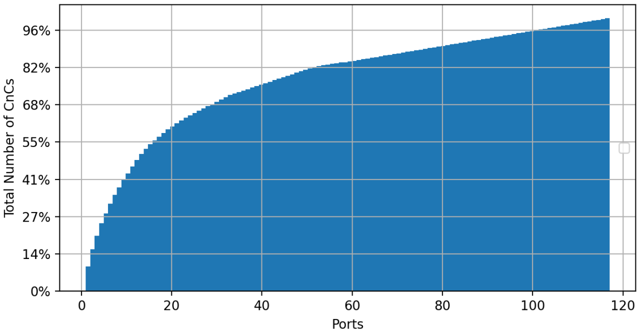

Figure 6: Cumulative distribution of C2 servers across port numbers ranked by popularity.

b. Selecting IP spaces: Second, to identify IP spaces with a higher likelihood of C2 hosting. We plot the distribution of the C2 servers across ASes in Figure 7. We find that the seven most "popular" ASes host roughly 65% of the malware-specified C2 servers.

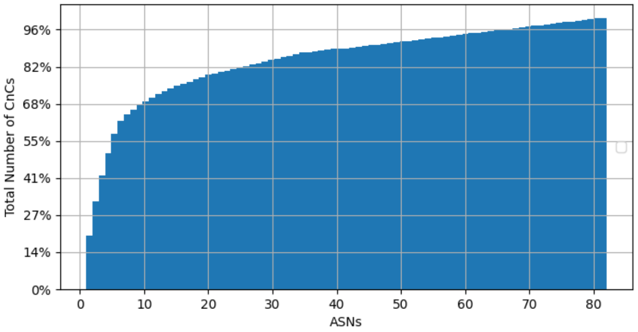

Figure 7: Cumulative distribution of C2 servers across Autonomous System Numbers (ASNs) ranked by popularity.

Based on these two observations, we create our target list of IP:port tuples as follows. From these seven ASes servers, we select subnets with more than ten reported C2s, and the top 12 popular ports as reported in Table 4 in §6.

F RELEASED DATASETS AS GROUND TRUTH

To facilitate further research in the area of IoT malware's C2 communication and to enable comparisons against C2Miner we are committed to releasing the ground truth we prepared as part of this work. We publicly release the following datasets:

- `Ground1` as described in Table 1 in §5.2 that includes the SHA56 of the binaries, and their C2 address. We also release the related info that we queried from VirusTotal whenever they existed. The related information includes the ASN, Country and timestamp.
- `DFinger` as described in Table 1 in §5.2 that includes all the fingerprints that we extracted.

We do not publicly release the other datasets due to ethical concerns. This includes the malware binaries and the traffic traces. We do provide these datasets upon request for research purposes only.