C2Miner: Weaponizing Malware Binaries for Discovering Live C2 Servers

Anonymized Paper # 455

Abstract

How can we identify live Command and Control (C2) servers for a given 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 on purpose, and use sophisticated and proprietary communication protocols. In this work, we propose, C2Miner, a novel approach to weaponize malware binaries to make them reveal their currently live C2 servers. The novelty of our approach is that we fool the malware twice: (a) we convince the malware to activate in a sandbox and start searching for its C2 server, and (b) we perform a Monkey-in-The-Middle (MitM) attack on the C2-bound traffic and use it to search for C2 servers in an IP:port space of our choice. Enabling this idea in practice involves several significant challenges. Even after activating the malware, we need to: (a) disambiguate the C2-bound traffic generated by the malware, (b) determine if a target IP:port is indeed a C2 server as opposed to a benign server, and (c) fingerprint and cluster C2 communications, which is important for large-scale exploration. We develop new approaches to address all these problems effectively, including a grammar-based method to distinguish benign from C2 servers, and group servers of the same malware family. In our work, we use 3M distinct explorations 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 85%. More importantly, we show how our approach can be used in practice and at scale. Using only two six-month old binaries, we scan 18K IP:port pairs daily for a six days and find 6 new live C2 servers.

1 Introduction

Identifying Command and Control (C2) servers is a critical capability in the battle against botnets and IoT botnets in particular. As the 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.

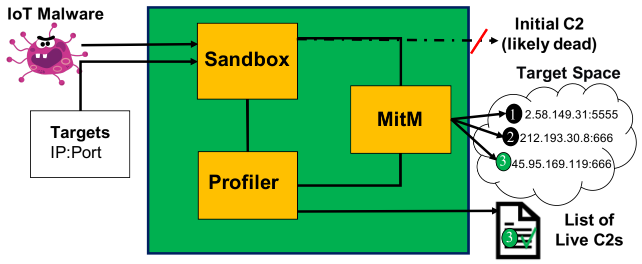

Figure 1: Overview of C2Miner and how it weaponizes malware binaries to reveal live C2 servers by activating the malware, and redirecting the traffic to scan IP spaces of interest.

For example, the defense could include 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. Finally, Internet Service Providers (ISPs) and law enforcement can block and take down these C2 servers to disrupt the botnet [27, 31].

Given their crucial role in the operation of botnets, C2 domains 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 is questionable [8, 29]. The issue for the domain-based feeds is more severe, and has been known for quite some time: for example a study in 2014 found quote “the union of 15 public blacklists includes less than 20% of malware domains” [26].

Finding C2 server addresses that are currently live is a challenging task. First, the current practice in industry for finding the C2 servers is based on manual effort and domain knowledge [43]. Even in academia, researchers rely on the patterns shared by industry experts for C2 address detection [26, 34]. This is because C2 server communications are using proprietary and increasingly sophisticated protocols, therefore automated reverse engineering of their application layer protocols is not straightforward. Second, malware C2 servers are short lived (usually less than three days), and in the case of IoT malware do not include fall-back mechanisms [37]. This means that by the time analysts find malicious binaries and reverse engineer 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 [37].

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. The input is a malware binary, and a target IP:port (hereafter target for short) space of interest. The desired output is: (a) live C2 servers for that binary in the given IP:port space, and (b) traffic "fingerprints" of the C2 communication, which can be used to identify C2 servers in a network trace.

We make the an overarching assumption of nearly *zero a priori knowledge*, which makes the problem harder but more relevant in practice. Specifically, we assume we do not have any a priori knowledge about: (a) the binary (e.g., its family), (b) the targets within the given IP:port space, and (c) actual bot traffic patterns, namely, no access to existing traffic traces. Here, the IP:port space of interest is part of the problem input and it could reflect the space that an ISP or an enterprise owns or is interested in.

State of the art: There has been limited work addressing the problem in the way we have framed here. We can classify related efforts in the following groups. First, several efforts analyze the behavior of IoT malware either at the OS or network level [5,13,14,21,24,30,35,37,39]. These approaches develop behavior models but do not attempt to find live C2 servers, especially servers that are not contacted directly by the malware. Second, several efforts analyze network traces and develop methods to find botnet traffic [16, 22, 23, 28]. Third, several approaches probe the Internet in pursuit of finding live targets [5, 17, 33, 42], but they follow a different approach to ours. They attempt to imitate a bot, by replaying traffic or reverse engineering the protocol, and as a result they cannot handle sophisticated and complex communication protocols, as we explained earlier. Finally, the most relevant related work tries to engage with the malware and reveal its alternative C2 servers [18, 32, 34]. Unfortunately, these methods are not as applicable to IoT botnets, which have a disposable nature and rarely use alternative C2 addresses [37]. We discuss previous work in detail in section 7.

Contribution: C2Miner is a novel probing approach for discovering live C2 server by weaponizing malware binaries with zero a priori knowledge. The novelty of our approach is that we fool the malware twice: (a) we get the malware to activate in a sandbox and start searching for its C2 server, and (b) we perform a Monkey-in-The-Middle (MitM) attack on the C2-bound traffic, which we redirect to C2 candidates within the target space of our choice. To substantiate this approach, we develop 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 communications, which is necessary for exploration at scale. In particular, we introduce a grammar-based method to model and fingerprint the C2 communication.

Having built a working prototype, we evaluate and showcase the capabilities of our approach using 575 binaries and exploring 3M IP:port combinations. In our evaluation, we establish the ground truth through manual analysis. Our contributions can be summarized as follows:

- **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 web servers with an F1 score of 85%. To achieve this, we introduce a grammar-based approach for characterizing the communication between bots and C2 servers.
- **We how we can weaponize old binaries to discover current live servers**. We provide initial indications of the promise of our approach by weaponizing two six-month-old binaries to identify six live servers within a six day of probing 1536 IP addresses on 12 ports! In addition, we find that 84% of pairs of malware in our dataset can interchangeably talk to the same C2 server.

Open-source and data sharing. We already have our code as a public GitHub repository with an open-source license. We are committed to enabling and collaborating with the security community in the search of powerful tools against cybercrime. In addition, we will make our datasets and ground truth available to facilitate further research.

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

2 Design & Implementation

We start by describing the architecture of C2Miner, which is shown in Figure 1. Recall that C2Miner takes as an input one or more malware binaries and a target IP:port space to explore. For ease of presentation, we discuss the operation of C2Miner in the following four steps: (a) activating the binary, (b) disambiguating C2-bound traffic from other traffic, (c) launching a MitM redirection of the C2-bound traffic to a target address, and (d) determining if the responding target is a C2 server. The *sandbox component* is in charge of the binary activation. The *profiler component* oversees the disambiguation and the determination functions above. The *MitM component* implements the redirection of the traffic to the target IP address. In the following subsections, we describe how each task is accomplished.

Note that, a significant contribution of our work is our grammar-based fingerprinting capability. In our context, fingerprinting refers to the detection of a remote network service based on its network footprint. We discuss this in section 3 in order to streamline the flow of the paper. We use this capability in two ways. First, it is used by the profiler for determining the existence of a C2 server. Second, it is used to cluster binaries based on their type of interaction in order to optimize a large-scale deployment of C2Miner as we discuss in section 5.

2.1 Binary Activation

We need to overcome three challenges in order to activate an IoT malware binary. By activation, we mean driving the malware process execution to a point where it generates network traffic. First, the binary must be executed on the corresponding CPU architecture. We statically analyze the binary in order to find the right architecture for execution. If the binary is packed, we unpack it using well-known packers (like UPX). If we can not determine the CPU architecture after unpacking statically, we iteratively execute the binary on every CPU architecture that we support until the binary is activated, or we exhaust all options.

Second, we need to identify a virtual environment. Executing a malware on actual IoT devices is not feasible: there are many IoT devices and given our zero knowledge assumption, we would not know which device it targets. In contrast, a virtualized environment allows us to instrument the execution, and efficiently analyze the malware. However, failures could happen because sometimes an instruction is not supported by the virtualization engine. Our *sandbox* emulates the malware execution using QEMU [6], and as we report in subsection 4.2, QEMU performs very well.

Third, the execution platform should be configured properly for the malware to activate. By configuration, we mean the operating system and the filesystem. 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 [14], an open source tool, for this purpose, which provides us the appropriate configuration. We report our activation rate in subsection 4.2.

We highlight some implementation details of 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 component stores the logs for analysis by other components. It logs network traffic, and the system call traces. The former provides us the data for C2 communication analysis, while the latter allows us verify the correctness of the emulation. For logging, we use *strace* and the QEMU traffic record functionality. Our emulator executes the malware with the *strace* logging enabled. For network traffic collection, we enable 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.

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 $Scores$
```

2.2 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. Worm-like malware like IoT malware tries to infect other devices, and hence it generates scanning and exploitation traffic. In addition, there might be random traffic, for instance, for checking Internet connectivity on the infected device. The profiler of C2Miner finds the C2 address (IP:port or DNS) that the analyzed malware tries to communicate with among all the traffic that it generates.

To identify C2 traffic, we develop the *Disambiguate-C2-Traffic* algorithm illustrated in Algorithm 1 (and implemented using the Python pyshark library [25]) that analyzes the generated traffic from the guest. The algorithm analyzes every target 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. The algorithm consists of two main parts: (a) quantifying the communication activity, and (b) assigning a likelihood score to the targets.

Part 1. Quantifying activity. As a first step (lines 4-8), we analyze each packet and count the number of times targets (IP:PORT or DNS) and ports are contacted. We assume the malware uses TCP for C2 communication, which seems to be the preferred protocol [32], although we note that extending our approach to UDP is a matter of engineering effort. We filter out the traffic from unrelated protocols: ICMP, DHCP, ARP and NTP at line 5. Although there are records of using ICMP and NTP in covert channels, we do not find such instances in IoT malware and we ignore it for now.

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. 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. If it fails, we count the number of times 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 part, we can calculate the likelihood score for each target (lines 9 to 13).

First, for DNS-based addresses, we check the reputation of the domain. To reduce false positives, we eliminate well-known domains by using the top X (X=1,000 in our case, but is configurable) number of domains based on the Alexa ranking. Note that we check the reputation 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. The second insight is that a port number used for the C2 server is different from that used for other bot traffic, such as proliferation (scanning and attacking other devices). In this case, the same port will be used across many different IP addresses. So the higher the number of IP addresses that are being contacted at a specific port, the lower the likelihood that a communication at that port is towards a C2 server. These two insights can be quantified in many different ways. Here, we opted to start with 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 formula works well in practice as we show in subsection 4.2.

2.3 MitM-enabled Probing

The task of the MitM component is to redirect the malware C2 traffic to candidate targets which are given as inputs. 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 component assumes that Algorithm 1 has already identified the C2-bound traffic. We discuss below how the MitM component handles IP:port-based and DNS-based targets.

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 candidate server. In our case, implementing this functionality is non-trivial: as using the readily available NAT functions at the network perimeter would not be sufficient, we moved 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. In neither of these cases, redirecting traffic to another IP address is allowed on iptables installed at network perimeters. That said, iptables can change a destination IP and port address if the traffic originates from the proxy itself. For this reason, we rely on the guest's (infected machine) iptables to provide the redirection. Interested readers can refer to our GitHub repository for further technical details.

b. DNS-based targets. For DNS based addresses, we take a different approach. 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 would map 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.

2.4 C2 Determination

One of the profiler component'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 is available, some knowledge can be drawn through the analysis of the traffic and the problem can be solved differently.

We propose two methods: (a) SYN-DATA-aware, and (b) Fingerprinting-aware. In the next section, we present a fingerprinting method that, among other applications, can determine whether the target is a C2 server under the assumptions of having C2 traffic traces. We compare the two methods' accuracy in subsection 4.3.

The SYN-DATA-aware method. The solution that we present here is independent of the application layer protocol used by the malware and works 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. Establishing what communication is meaningful is a key task of the profiler component. 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 a challenging task. 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 even by silently not responding 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 restarts 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 seem rather rare, as we discuss in subsection 4.3.

If the communication is successful based on the number of SYN flags, we check for exchange of data between the malware and the candidate to further assess the communication success. In general, the malware does more than completing the handshake; it will send data to the C2 server. This could be a simple "PING" command, but still it is a payload for the TCP layer. In response, the server also does more than acknowledging the packet; again, this might be only a few signature bytes, although in some cases the C2 server might not respond with data. Regardless, we find that looking for exchange of data increases the precision because false positives are rare (see subsection 4.3).

In summary, for C2 determination, we check whether the number of times the SYN flag is set is lower than a threshold (=1 by default), and if there is exchange of data packets between the malware and the C2 server. Interested readers can see our GitHub repository for further technical details (withheld for anonymity).

3 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 robust to variations and noise. 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):

- *The number of send/receive pairs between the endpoints.* Similar to a dialogue, the communication is in the form of request/response pairs. For instance, just like "Hi" answer to a "Hi," a "PING" or a "PONG" is a response to a "PING."
- *The 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.
- *The termination control flags (if any).* This helps us better understand how the connection terminated. Was it because the connection was interrupted by the network (or operating system)? Was there error etc.?

At a high level, we summarize the flow by capturing: (a) its beginning and end, (b) its adherence to TCP flow rules, and (c) 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 $\Sigma = \{handshake, client, server, ack, synfin, rst, attr^*\}$ are the terminals symbols (we explain $attr^*$ below). Note that $Flow$ is the start symbol. The production rules set P consists of the following rules:

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

a. Summarizing a flow. Any 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

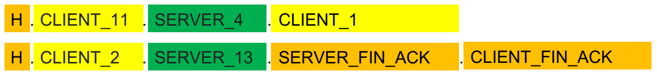

Figure 2: Two examples of phrases in our language: H is the TCP handshake, afterwards the client and server exchange packets of x bytes (CLIENT_X and SERVER_X, respectively)

or data packets from the endpoints denoted by "*Sender*" non-terminal and the *client* and *server* terminals. In our malware analysis, *client* is the malware and the *server* is the C2 server. Please note that sometimes for better readability (and to save space), we use the symbol "|" (corresponding to logical or) to present two rules in one line.

b. Summarizing each packet. Our grammar provides the ability to profile each packet in the flow using the symbol Attribute and $attr*$. Defining what are 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 two latter attributes require 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.

Consequently, we define $attr*$ in (11) to be simply the length of the packet. In section 4, we show that this attribute works sufficiently well with our dataset and the current sophistication level (or lack thereof) of the malware. However, in the future, we will consider additional attributes.

The best way to understand the grammar is through some 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 sends a packet with the FIN flag too.

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 perhaps 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 even 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. In fact, to increase the robustness of our approach, we use clusters of strings, which we discuss below. A cluster of similar fingerprint strings based on a distance function indicate 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 paper. Here we use it to: (a) determine if a target is a C2 server, (b) optimize the weaponization of large set of binaries. We revisit the latter in see section 5. The 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. $LCS(f_i, f_j)$ will be used 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:

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

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

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

Type	Breakdown	LOC
Language	Shell	636
	Python	2,897
Components	Sandbox	1,239
	MitM/Probing	553
	Profiler	1,231
	Other	510

Table 1: LOC breakdown of the C2Miner implementation.

Usage and practical considerations. In subsection 4.4, we discuss the choice of parameter k and the seeds for the k-means algorithm. We also discuss how homogeneous the clusters are in practice. In subsection 4.3, we use our grammar-based method to fingerprint the communication behavior between the malware and likely C2 servers and we compare these string with that of known C2 servers. There, we report a fingerprint match under the condition that LCS returns a string that contains exchange of data from both endpoints.

4 Evaluation

We evaluate our approach to answer the following questions:

(Q1) How accurately can we disambiguate C2-bound traffic from other malware traffic? (see subsection 4.2)

(Q2) How accurately can we determine that a probed target is indeed an active C2 server? (see subsection 4.3)

(Q3) Can our grammar-based fingerprinting help distinguish different families of malware? (see subsection 4.4)

(Q4) What is the likelihood that two binaries from the same malware family speak the same dialect, i.e., can communicate with the same C2 server? (see subsection 4.5)

This area of research is notorious for having limited to non-existent benchmarks and ground truth, despite several studies in this direction [37, 39]. Thus, we are left to create our own ground truth, which we will make available to the community. We start by discussing our general data collection strategy, and then detail the creation of ground truth in the respective section for each question separately.

Implementation and experimental setup. We implemented C2Miner in roughly 3,500 lines of code (LOC) in Python and Linux Shell scripts. More details about the effort breakdown can be found on Table 1. 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 is running an Ubuntu bionic 18.04 AMD64 operating system.

4.1 Malware Dataset

We collect IoT malware samples from MalwareBazaar [3] and VirusTotal [2] on a daily basis. We focus on the MIPS architecture since it is a common platform for IoT devices, and the least explored architecture by the security community [14].

Dataset	Description
DAll	575 binaries collected in total.
DIP	367 IP:port pairs of C2 servers verified by VirusTotal.
Ground1	97 randomly selected malware binaries used as ground truth.
Ground2	149 binaries with a live C2 used as ground truth.
Trace-1	230MB traffic of 49 binaries from *Ground2* redirected to "talk" to 32 C2 servers in Jan 2022.
Trace-2	317MB traffic of 80 binaries from *Ground2* redirected to 34 C2 servers and 39 benign servers.
DFinger	100 traffic fingerprints in our formal grammar of 100 binaries from *Ground2*.

Table 2: Our datasets consisting of a total of 575 IoT malware samples for MIPS 32BE from MalwareBazaar and VirusTotal.

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. 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 best practices [45].

To achieve greater variability, we collected binaries during four different phases: Mar 29, 2021 to Apr 5, 2021 (P1), Jun 6, 202 to Aug 11, 2021 (P2), Oct 25, 2021 to Nov 1, 2021 (P3), and Nov 10, 2021 to Jan 20, 2022 (P4). On average, we collect roughly 4 new MIPS binaries a day during our collection phases.

The union of all the samples that we collected is *DAll*. A summary of all our datasets is listed in Table 2. We initiated 3M exploration requests to 150K IP addresses over the course of our 24 weeks study period to augment and build these datasets. We identify 20K live services (IP:port), and weaponized 149 malware binaries 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.

Detection 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 alive.

Note that C2Miner activates 90% of the binaries, which is on par with success rates reported in earlier studies [14]. In the remainder of this work, we only focus on activated binaries. The failures to activate were mainly due to "illegal instruction error" within the QEMU operation, and in one instance, the malware killed itself, because it detected the emulation environment. Improving the activation rate is part of our future work.

Achieving reasonable family coverage. In our datasets, we find binaries from several prolific families, including Mirai, Gafgyt, Tsunami, Remaiten and LightAidra. We find that 95% of samples belong to Gafgyt and Mirai, which seem to be the dominant variants [5, 21, 35].

Obtaining C2 traffic for ground truth. In our case, having a malware binary is 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 server (which the malware attempts to contact initially on its own) are usually inactive: only 49% of the samples have a live C2 server by the time they appear in our two sources. Making things worse, many of the live servers become inactive before we finish our experiments.

In total, we were able to find 149 samples that had live C2 servers for this study, which we refer to as *Ground2*. These samples 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. We were initially surprised to see this, as we expected that a newly activated malware would focus on connecting to its C2 server. This would have made the detection of the C2 traffic easier, but, unfortunately, this is not the case.

4.2 Traffic Disambiguation Precision

To answer Q1, we want to evaluate the precision of the traffic disambiguation component, 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 an 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 [19], and even featured in an xkcd comic (`https://xkcd.com/1247`).

Ground Truth: In absence of a benchmark, we created the *Ground1* dataset by manually analyzing malware samples and the traffic they generate. We used the Ghidra [4] decompiler for statically reverse engineering the samples. Our aim was to find addresses that are not used in the malware proliferation phase, and can send commands that would change malware execution behaviors e.g., sending a DoS initiation command to the malware sample. These addresses are either IP:port or a DNS name. We find that only 9% of the binaries use DNS-based C2 addresses, and the rest are IP:port-based.

Result: We disambiguate C2-bound traffic with 92% precision. We evaluate our algorithm as follows. For every binary in the *Ground1* dataset, the algorithm returns the target that is most likely the C2 server. We verify manually if this is indeed a C2 server, and we report the precision of the algorithm in Table 3. C2Miner has a precision of 92% in correctly finding the C2-bound traffic on the *Ground1* dataset. For comparison, we also report a simple detection method based on finding IP addresses by statically analyzing binaries; we use Radare2 [1] scripting for this purpose. The detection based on this method has 34% precision. For this method, "Success" reports whether the method is able to report any IP address; the main reason for failure is code obfuscation.

Method	Samples	Precision
Radare2-Static	97	34%
C2Miner	97	**92%**

Table 3: Precision of C2-bound traffic disambiguation: comparing C2Miner against static analysis with Radare2 on *Ground1* dataset.

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 security intelligence combining threat intelligence feeds of 91 vendors. For IP:port-based C2 targets, out of the 47 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 dropped to 5% after 4 weeks. The identification of domain-based C2 servers is even worse as VirusTotal fails to detect 30% of domain-based C2 servers as malicious. We speculate that a contributing factor is that IoT is an emerging threat and tools and services around it need to mature further.

4.3 C2 Determination Accuracy

Now we turn to answer Q2 on C2Miner's ability to identify if a weaponized malware is talking to a real C2 server: Since, we have selected the target, there is no reason that this target is indeed a C2 server.

Ground Truth: The required ground truth here is quite unique. We need a collection 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 create our own *Trace-2* dataset as follows. First, we start from samples with live C2 server from the *Ground2* dataset. Second, we select /23 subnets from the C2 server of these samples; this is to simulate the real application of C2Miner in practice. Third, we perform a light-weight SYN stealth scanning to find the live services within those subnets. Fourth, we redirect the C2 traffic to all live services in those subnets. Finally, we manually analyze the contents of the traffic and decide what service the targets host.

In total, we communicate with 200 live services and receive responses containing a data payload from roughly 37% of these live services. The live services mainly host HTTP servers (Apache, Nginx, OrgaMon) but we find also OpenSSH, MySQL, ESMTP and IMAP servers. The rest are communications with live C2 servers. In total, we have malware redirected traffic to 34 C2 servers and 39 benign servers.

We determine C2 servers with 86% F1 score. We evaluate the effectiveness of our approach based on the Precision, Recall and the F1 score. We evaluate two C2 deter-

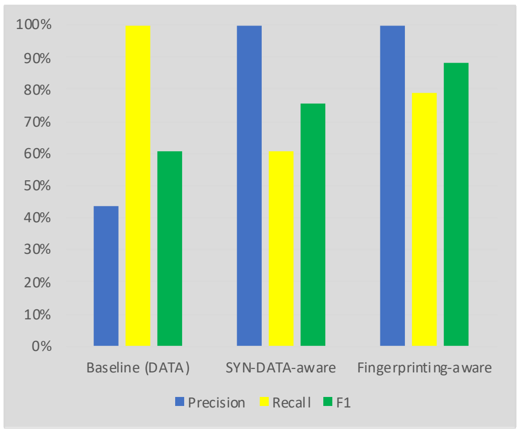

Figure 3: The precision of the C2 determination approaches based on the *Trace-2* dataset.

mination approaches: (a) **SYN-DATA-aware** (explained in subsection 4.3), and (b) **Fingerprinting-aware** (explained in section 3). As reference, we use a simple **baseline**, which assumes that a target is a C2 server as long as we receive any data response from it. We report the results in Figure 3.

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, as we mentioned, C2 servers will respond with data packets to malware requests.

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

4.4 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 clusters represent 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 need to create it. We point out that an already existing labeled malware datasets (with malware family labels) would not be useful because the malware should have a live C2 server so we can collect the communicated traffic. The main challenge is obtaining communicated traffic with C2 servers for IoT malware samples.

Malware family	# of samples
Mirai	49
Mirai-Gafgyt	21
Gafgyt-Mirai	20
Gafgyt	6
Gafgyt-Mirai-Lightaidra	4

Table 4: Malware labels of *DFinger* dataset based on the results reported by AVClass2. The first malware family name is reported by the majority of the AVs.

	mirai	gafgyt	gafgyt-mirai	mirai-gafgyt	gafgyt-mirai-lightaidra	Sum
C1	18%	7%	14%	**46%**	14%	100%
C3	**88%**	0%	0%	12%	0%	100%
C5	5%	19%	**71%**	5%	0%	100%

Table 5: Clustering effectiveness: the top largest cluster are reasonably aligned with malware family labels in the *DFinger* dataset. Clusters C3 and C5 represent 80% of the binaries.

We create the *DFinger* dataset which consists of 100 binaries and their fingerprints starting from the binaries in *Ground2* as follows. Among the *Ground2* binaries, we select the ones for which 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 contain communication with live C2 servers. We then transform the traffic to strings in our grammar as shown in section 3. To determine the malware family, we query VirusTotal and combine the results with AVClass2 [36]. Table 4 provides a summary of malware families for the *DFinger* dataset. AVClass2 reports two labels for nearly half of the samples, and labels samples as either as Mirai, Gafgyt, or a combination. We assign a label for all reported combinations for a more accurate representation.

Result: We hierarchically cluster the C2 traffic based on the method we explain in section 3. We randomly choose the seeds, and we observe that the choice of k has minimal effect for $k > 4$; they always converge to the same clusters. The malware family labels of the top level clusters is illustrated in Table 5. The top two clusters 3 and 5 have 80% of samples. All samples in cluster 3 are reported as Mirai by a majority of AVs (12% also reported as Gafgyt). Similarly, 90% samples are reported as Gafgyt by a majority of samples in cluster 5. On the other hand, the samples in the cluster 1 are mixed; only 25% are labeled only by a single malware family.

Overall, our hierarchical clustering results in 49 clusters shown in Figure 4. The interesting observation is the common patterns that appear in communications. Most Mirai samples

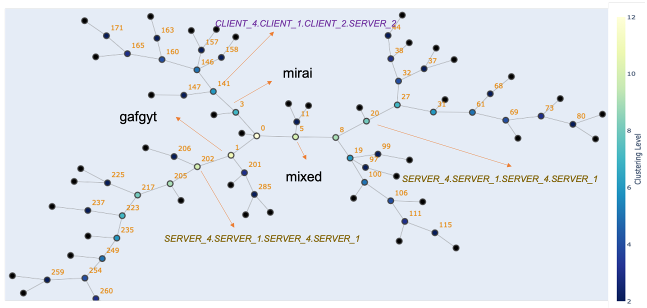

Figure 4: Hierarchical clustering of samples in *DFinger* based on their communicated patterns with C2. Clusters expand from cluster 0 (the initial dataset).

share the CLIENT_4.CLIENT_1.CLIENT_2.SERVER_2 pattern. On the other hand, most Gafgyt samples share the SERVER_4.SERVER_1.SERVER_4.SERVER_1 pattern. With these two patterns alone we can detect 76% of C2 communication.

4.5 MitM Functionality Assessment

The answer to Q4 on whether two binaries of the same family talk to the same C2 server indicates the promise that the weaponization of a binary can hold. Our initial analysis of publicly available IoT malware source code suggests this is true (please see Appendix C), nevertheless, we empirically assess this hypothesis. Our assessment has two goals. First, we want to see whether the MitM component 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 weaponizing an old binary can lead us to currently live 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 create the *Trace-1* dataset from our collected binaries as follows. In more detail, we start with the list of malware with live C2 addresses in *Ground2* 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 check if the binaries of the previous step communicate with the server successfully, which we validate with manual inspection. Our manual evaluation method is similar to what we described in subsection 4.3. The number of binaries in *Trace-1* 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 well in practice. We measure how successful our MitM approach is in practice based on 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 enough to find the C2 servers for the entire cluster. We use the fingerprinting-based method for determining C2 servers. We find that 84% of sample pairs can successfully cross-communicate with the C2 of the malware cluster.

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

Table 6: The target selection for our probing: (a) roughly 1536 addresses within 6 /24 subnets and (b) 12 ports in order of "popularity" in the default malware operation.

Family	MD5 Hash
Gafgyt	46501d723f368c22e5401f7c95d928ab
Mirai	800af659256f0232a27f955a4430aed0

Table 7: The family and MD5 hashes of the two binaries selected for probing in our case study.

5 Case Study: Weaponizing C2Miner

How can we use C2Miner in practice to find live C2 servers? This is the question that we answer in this section.

Our proof of concept deployment: limited effort, big results. We show case the value of C2Miner with a proof of concept deployment. For efficiency, we do our probing in two phases. First, we use a light-weight SYN stealth scan using MASSCAN tool [20] to establish liveness. Once this is established, we deploy the more resource consuming C2Miner probing. We carefully select binaries and IP:port space for exploration to efficiently use the computation resources.

To show the promise of our approach, we decided to weaponize only two old malware binaries. We selected: (a) one from the Mirai family, and (b) one from the Gafgyt family as shown in Table 7. We chose these two because their communication fingerprint is "close" to 76% of the samples, which is an application of our grammar-based clustering (see subsection 4.4). These samples are at least six month old: the samples were first reported to VirusTotal on 06/22/2021. These samples' MD5 hashes are reported on Table 7.

We made the following operational choices for the IP:port space selection. We conduct roughly 331K probes across 18K IP:port combinations for 1536 IP addresses and 12 ports with three probes per day for 6 days using the two binaries. We identified 6 /24 subnets and 12 "popular" ports as we explain below and shown in Table 6 based on the increased likelihood of observing malicious activity from past C2 hosts [10, 15]. We perform these experiments in January 2022.

This limited size case study is partly on purpose as it showcases the potential value that we can obtain, even with resource constraints. In the future, we intend to do a wide-scale longitudinal study using a few hundred binaries and an expended IP:port target space.

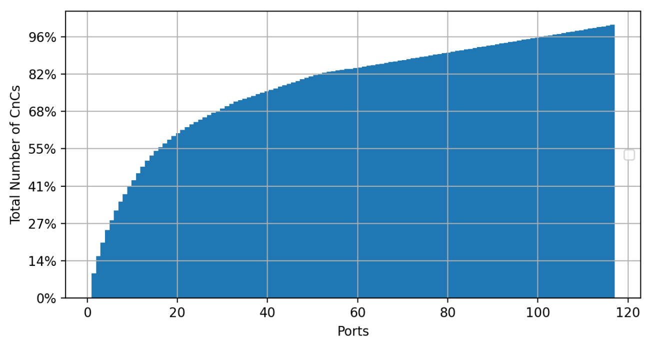

Figure 5: Cumulative distribution of C2 servers across port numbers ranked by popularity.

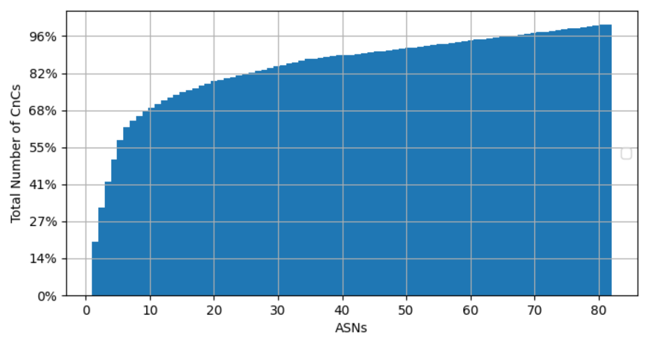

Figure 6: Cumulative distribution of C2 servers across Autonomous System Numbers (ASNs) ranked by popularity.

Spatial patterns of IoT C2 Servers. Deviating from the problem formulation, we are required to identify our own promising target space here. We study the malware-specified C2 servers with our dynamic analysis of the binaries in our *DAll* dataset. The result is the *DIP* dataset: each row is a C2 IP:port pair associated with its hosting Autonomous System and country. We also associate the IP:port pair with the date when the binary first appeared in our binary stream as described in subsection 4.1. We study the IP-space spatial properties of these malware-specified C2 servers in order to identify locality patterns.

First, we observe that 12 ports are used by half of the C2 servers. Figure 5 shows the distribution of ports. Using this insight, we focus our probing to to these frequently-used ports. Second, to identify useful IP spaces, we follow a multi-step approach. We plot the distribution of the C2 servers across Autonomous Systems (ASes) in Figure 6. We find that the 7 most "popular" ASes host roughly 65% of the malware-specified C2 servers. We take one more step to understand how consistent this popularity of the ASes is w.r.t. hosting C2 servers. We plot the presence of these seven ASes over time in Figure 7 as the malware-specified IPs appear in our *DIP* datset. Each of these ASes host more than 10 potential C2 servers. The heatmap shows that the top four ASes have potential C2 servers consistently during our collection period of 24 weeks. Extrapolating from this observation, these four

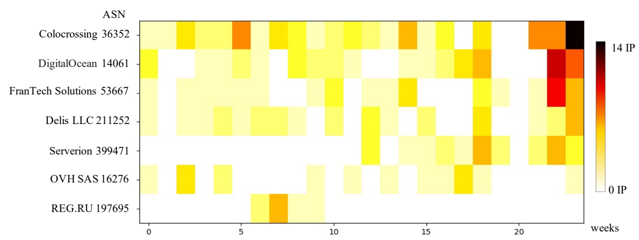

Figure 7: Heatmap showing the distribution of malware-specified C2 servers from our malware feed (MalwareBazaar and VirusTotal) across ASes. Darker color shows more servers. The top four ASs are consistently more active.

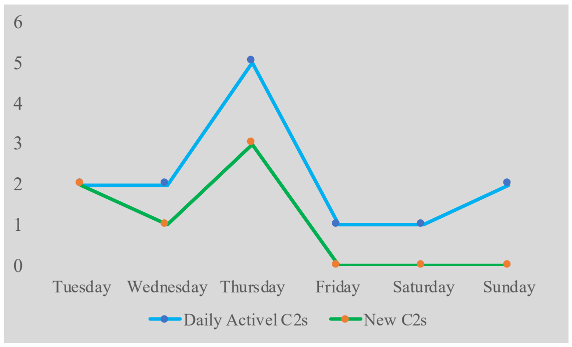

Figure 8: Weaponizing C2Miner: The number of responsive C2 servers per day and the number of distinct new C2 servers during our 6 day probing over only 1536 IP addresses across 6 subnets and 12 ports.

ASes servers seem to be good candidates for our study. Note that the "empty" column on week 20 coincides with the week between Christmas and New Year's Eve, and the binaries on that week were fewer and did not activate.

Based on the aforementioned observations, we create our target list of IP:port tuples as follows. We choose the top /24 subnets from the top four ASes, and the top 12 popular ports as reported on Table 6.

Temporal patterns: Daily report of active C2 servers. We employ C2Miner's probing mode in practice for 6 days. Below are some highlights of the results.

a. Probing success: 6 live C2 servers in 6 days. The results are very promising: we find a total of 6 active C2 servers for our binaries: 5 Gafgyt and 1 Mirai. In Figure 8, we plot the number of active servers each day and the new distinct servers discovered that day.

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 the limited IP space we explored, and the heuristic approach in finding IP space with likely C2 servers. Note that these are servers that without a MitM approach would not have been discovered even if one had activated these two malware in a sandbox (their malware-specified are inactive).

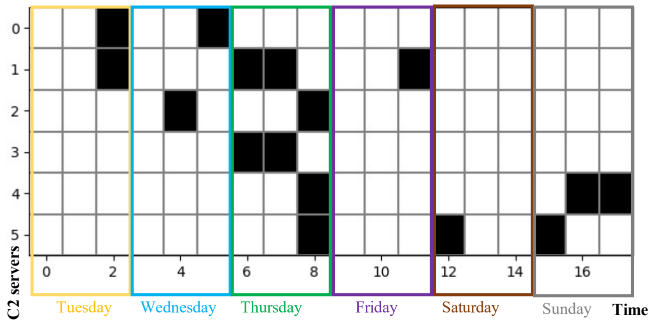

Figure 9: Weaponizing C2Miner: The responsiveness of the responsive C2 servers over time to our three daily probes which is 15% per probe assuming they are alive for the full duration. The exact IP:port values of these C2 servers are shown in Appendix B.

b. The temporal behavior of C2 servers. Intrigued by the seemingly high temporal variability of the C2 servers, we investigate further the daily liveliness of the found C2 servers, which we show in Figure 9.

Key Observation: Live C2 servers response rate is low even for bots of their family! Our preliminary statistics based on our limited study here seems to suggest that even a live server will not respond consistently to a bot request. We can count 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.

Best practice recommendation: These initial results suggest that we need to send many probes to a server to validate that it is not live given the low success rate that we observed above.

6 Discussion and Limitations

We explain the opportunities and limitations of C2Miner in the form of questions below.

a. What is the value of the MitM capability? The MitM capability allows us to weaponize the *old* malware and turn it into an active probing tool. It should be clear by now that without the MitM capability each binary can only reveal a small number of potentially obsolete C2 servers.

b. How can C2Miner be deployed at a large-scale? We envision our work as the basis for a powerful framework that can lead to an Internet-wide search for C2 servers assuming community-wide collaborative action and sufficient computational resources. For efficient use of the computation resources, as we explained, binaries and IP:port spaces can be prioritized based on the likelihood of success. For binary selection, cluster sampling can be applied, and for IP spaces, spatial locality can be leveraged. In addition, the targets can be determined by the purpose of the study and it can be driven by: (a) a given IP space, such as a customer-owned space), (b) the aggregation of external sources of IP:port reputation, and (c) an adaptive inference of likely destinations based on prior probing results. The first of these scenarios is especially applicable for assessing Autonomous Systems (ASes), some of which have become safe haven for attackers. Using C2Miner, ASes can scan their IP space and take actions to remove C2 servers.

c. Are the evaluation results generalizable? 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 evaluation studies [18, 32]. We continue to collect data and we will share the results with the community.

d. How can 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 calculate the similarity, and seeing if the traffic belongs to a cluster.

e. What if the malware does not activate? This is a concern for any work that relies on dynamic analysis. Our intention is to use the latest sandboxing technology, as this is not where the novelty of our work lies. If a "smart" malware refuses to activate, we cannot weaponize it. It has been observed [14] that current IoT malware is not yet as sophisticated as PC-focused malware. Although this may change in the future, we can hope that novel sandbox techniques will also evolve. In addition, note that we do not need to activate all the binaries, as long as we activate enough binaries that "talk" to most types of C2 servers.

f. Can C2Miner extend to non-IoT malware? The overarching approach applies to any malware that can be activated in a sandbox. However, our current methods will need significant fine-tuning to adapt to "behaviors" of other malware. For example, the algorithmic solutions of our approach have been inspired by and trained on the worm-like behaviors of IoT malware. IoT malware often generates large traffic for scanning and exploitation that would hide C2 communication, and this might not be the case for other malware. However, with sufficient customization the overarching framework could be made to expand to additional types of malware and platforms.

g. How difficult is it to evade C2Miner? We consider two cases. First, if the malware does not use encryption and obfuscation at the TCP/IP layer, we argue that we can engage with the C2 server, as along as we can activate the binary. Our rational is that it is difficult for a C2 server to detect MitM-intervention by looking only at a bot request at the network layer and our initial results seem to corroborate this.

Second, if the malware uses encryption and obfuscation 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 not commonly used. In fact, many related security solutions will not work in the presence of encryption.

More generally, security is an arms race, and a security approach is successful if it forces hackers to modify their behavior. Especially, if this modification is heavy and costly as is the case with encryption at the network layer.

7 Related Work

There are several categories of related work to our research. Below, we discuss each category and explain how this research is different.

Overall, none of the related efforts has focused on the problem as framed here: finding live C2 servers for an unknown binary and within a target IP:port space as we do here. In addition, the weaponization of the binary using a MitM approach is also quite unique.

a. IoT malware analysis. Studying the behavior of 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, 21, 24]. These works characterize the infected IoT devices, the infection vectors, and the life-cycle of the malware. While these studies provide a deep insight into a single malware family and its life-cycle, they are less likely to lead to generalizable methods for detecting C2 servers dynamically.

Another group of prior efforts characterize the behaviors of several malware families at the same time [12–14, 30, 35]. Finally, several studies analyze C2 server communication from a networking point of view [35, 37, 39]. Although interesting and informative, these studies focus on understanding the infrastructure that supports its operation, and profiling aspects of the malware behavior, but do not engage in active probing like we do.

b. Active probing of malware: The most relevant studies to our work focus on active probing. Such efforts require an understanding of the malware communication protocol and the encryption algorithm, if any. Assuming that this can be accomplished effectively, searching for C2 servers of a single malware family becomes easier [5, 17]. In addition, there is prior work that took the first steps in automating the active probing for a more widespread group of malware families [33, 42]. The main problem with these approaches is that they can not handle the communication protocol when there is encryption. In contrast, we overcome this issue as we let the activated binary "speak" to the server with or without encryption, and we simply observe the communication.

c. Engaging with malware: Prior work has attempted to engage with malware to understand different aspects of its behavior [18, 32, 34]. Two earlier works [32, 34] try to reveal the malware's alternative methods of communicating with C2 servers. In contrast, IoT botnets are disposable and hence alternative addresses are not a broad phenomena for them [37]. A most recent work [18] develops techniques to detect and spoof bot-to-C2 communications, but their technique is applicable only to malware with over-permissioned protocols and they do not do active probing.

d. Network Traffic Modeling and Analysis: Modeling network traffic enables the identification of particular protocols or activities. The first group of work in this area tries to infer the application layer protocol [9, 11, 40]. However, this is not effective in presence of encryption, slow because of deep packet inspection, and requires access to the execution context. In comparison, we only need the network traffic for fingerprinting. The second group of work is more closely related and focuses on network flows [7, 38, 41, 44] and use features that include IP, port, timestamps, number of bytes, connection duration etc., which are more difficult to generalize. Instead, our approach creates an abstract representation of a flow by modeling using a formal language.

Studying the network behavior of malware has been the subject of another line of related work [16, 22, 23, 28] trying to find botnet traffic within a network trace. Thus, they need the network trace, and they can only find servers whose flows happen to be in that trace, in contrast to our ability to selectively target any potential device in the Internet.

8 Conclusion and Future Work

We propose C2Miner, a novel approach to weaponize malware binaries to make them reveal their currently live C2 servers. The novelty of our approach is that we fool the malware twice: (a) we convince the malware to activate in a sandbox and start searching for its C2 server, and (b) we perform a MiTM attack on the C2-bound traffic and use it to search for C2 servers in an IP:port space of our choice. 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 85%, and c) fingerprint and cluster C2 communications effectively.

In the future, we plan to extend our work in two ways. First, we plan to systematize and automate the task of finding candidate targets by combining external sources of information and using adaptive techniques. Second, we will conduct a large-scale longitudinal study that will provide: (a) an extensive list of live C2 servers, and (b) spatio-temporal properties of their behavior and migration patterns.

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.

References

[1] Radare2 - libre and portable reverse engineering framework. https://www.radare.org/n/.

[2] VirusTotal. https://www.virustotal.com.

[3] Abuse.ch. MalwareBazaar. https://bazaar.abuse.ch/.

[4] National Security Agency. Ghidra - software reverse engineering framework. https://www.nsa.gov/resources/everyone/ghidra/.

[5] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the Mirai Botnet. In *Proceedings of the USENIX Security Symposium*, 2017.

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

[7] Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, 2012.

[8] Xander Bouwman, Harm Griffioen, Jelle Egbers, Christian Doerr, Bram Klievink, and Michel van Eeten. A different cup of TI? the added value of commercial threat intelligence. In *Proceedings of the USENIX Security Symposium*, 2020.

[9] Juan Caballero, Pongsin Poosankam, Christian Kreibich, and Dawn Song. Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2009.

[10] M Patrick Collins, Timothy J Shimeall, Sidney Faber, Jeff Janies, Rhiannon Weaver, Markus De Shon, and Joseph Kadane. Using uncleanliness to predict future botnet addresses. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement (IMC)*, 2007.

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

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

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

[14] Ahmad Darki and Michalis Faloutsos. RIoTMAN: a systematic analysis of IoT malware behavior. In *Proceedings of International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2020.

[15] Ali Davanian. Effective granularity in internet badhood detection: Detection rate, precision and implementation performance. Master's thesis, University of Twente, 2017.

[16] Lorenzo De Carli, Ruben Torres, Gaspar Modelo-Howard, Alok Tongaonkar, and Somesh Jha. Botnet protocol inference in the presence of encrypted traffic. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, 2017.

[17] Brown Farinholt, Mohammad Rezaeirad, Paul Pearce, Hitesh Dharmdasani, Haikuo Yin, Stevens Le Blond, Damon McCoy, and Kirill Levchenko. 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)*, 2017.

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

[19] Sean Ghallager. 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/, 2013.

[20] Robert David Graham. Masscan: Mass ip port scanner. https://github.com/robertdavidgraham/masscan.

[21] Harm Griffioen and Christian Doerr. Examining mirai's battle over the internet of things. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2020.

[22] Guofei Gu, Vinod Yegneswaran, Phillip Porras, Jennifer Stoll, and Wenke Lee. Active botnet probing to identify obscure command and control channels. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, 2009.

[23] Guofei Gu, Junjie Zhang, and Wenke Lee. Botsniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2008.

[24] Stephen Herwig, Katura Harvey, George Hughey, Richard Roberts, and Dave Levin. Measurement and analysis of hajime, a peer-to-peer iot botnet. In *Proceedings of the Network and Distributed Systems Security Symposium (NDSS)*, 2019.

[25] KimiNewt. pyshark - python wrapper for tshark. https://github.com/KimiNewt/pyshark/.

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

[27] Victor Le Pochat, Sourena Maroofi, Tom Van Goethem, Davy Preuveneers, Andrzej Duda, Wouter Joosen, Maciej Korczyński, et al. A practical approach for taking down avalanche botnets under real-world constraints. In *Proceedings of the 27th Annual Network and Distributed System Security Symposium*. Internet Society, 2020.

[28] Chaz Lever, Platon Kotzias, Davide Balzarotti, Juan Caballero, and Manos Antonakakis. A lustrum of malware network communication: Evolution and insights. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2017.

[29] Vector Guo Li, Matthew Dunn, Paul Pearce, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. Reading the tea leaves: A comparative analysis of threat intelligence. In *Proceedings of the USENIX Security Symposium*, 2019.

[30] Tongbo Luo, Zhaoyan Xu, Xing Jin, Yanhui Jia, and Xin Ouyang. Iotcandyjar: Towards an intelligent-interaction honeypot for iot devices. *Black Hat*, 2017.

[31] Yacin Nadji, Manos Antonakakis, Roberto Perdisci, David Dagon, and Wenke Lee. Beheading hydras: performing effective botnet takedowns. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 121–132, 2013.

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

[33] Antonio Nappa, Zhaoyan Xu, M Zubair Rafique, Juan Caballero, and Guofei Gu. Cyberprobe: Towards internet-scale active detection of malicious servers. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2014.

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

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

[36] Silvia Sebastián and Juan Caballero. Avclass2: Massive malware tag extraction from av labels. In *Procedings of the Annual Computer Security Applications Conference (ACSAC)*, 2020.

[37] Rui Tanabe, Tatsuya Tamai, Akira Fujita, Ryoichi Isawa, Katsunari Yoshioka, Tsutomu Matsumoto, Carlos Gañán, and Michel Van Eeten. Disposable botnets: examining the anatomy of iot botnet infrastructure. In *Proceedings of the International Conference on Availability, Reliability and Security (ARES)*, 2020.

[38] Florian Tegeler, Xiaoming Fu, Giovanni Vigna, and Christopher Kruegel. Botfinder: Finding bots in network traffic without deep packet inspection. In *Proceedings of International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2012.

[39] Pierre-Antoine Vervier and Yun Shen. 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)*, 2018.

[40] Zhi Wang, Xuxian Jiang, Weidong Cui, Xinyuan Wang, and Mike Grace. Reformat: Automatic reverse engineering of encrypted messages. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, 2009.

[41] Peter Wurzinger, Leyla Bilge, Thorsten Holz, Jan Goebel, Christopher Kruegel, and Engin Kirda. Automatically generating models for botnet detection. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, 2009.

[42] Zhaoyan Xu, Antonio Nappa, Robert Baykov, Guangliang Yang, Juan Caballero, and Guofei Gu. Autoprobe: Towards automatic active malicious server probing using dynamic binary analysis. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2014.

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

[44] Ali Zand, Giovanni Vigna, Xifeng Yan, and Christopher Kruegel. Extracting probable command and control signatures for detecting botnets. In *Proceedings of the Annual ACM Symposium on Applied Computing (SAC)*, 2014.

[45] Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. Measuring and modeling the label dynamics of online anti-malware engines. In *Proceedings of the USENIX Security Symposium*, 2020.

Appendix

A Ethical Considerations

We adhered to an ethical code of conduct and best practices for doing research with live malware samples. First, we did not violate anyone's privacy, as we never dealt 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 has been lightweight and we did not increase in any significant way the load on servers or the traffic on the network. Third, we took all possible measures to limit any potential harm. We only let C2 traffic communicate with real servers, which are already compromised devices and we filter out traffic that goes to any destination other than the C2 server. Fourth, even as the malware running in our sandbox had temporarily joined a botnet, we were 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.

B C2 Server Addresses

C2 server ID	IP	port
0	2.58.149.34	:5555
1	212.193.30.91	:666
2	45.95.169.119	:666
3	136.144.41.240	:666
4	212.193.30.123	:5555
5	107.173.176.144	:42516

Table 8: C2 IP:ports found during our case study in section 5.

C Preliminary Experiments

IoT malware communication protocols. We conducted a small-scale manual study to 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 10 IoT malware families based on their source code found on GitHub. A summary of the protocols is reported in Table 9.

a. The bad news. We observe that communication protocols vary significantly between families. As a result, 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 section 5.

These insights suggest that C2Miner could work reasonably well in practice as long as we have arbitrary binaries of the family of interest, even if these binaries are not the latest versions.

Malware	Communication	Details
Gafgyt	Custom	PONG command is communicated via IRC, and 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.
Linux.wifatch	Custom	All C2 commands are custom binary based.
Remaiten	IRC	Similar to Lightaidra but commands are different.
Lizkebab	Custom	Similar to Gafgyt but commands are different.
LuaBot	Encrypted payload	Uses MatrixSSL library for payload encryption.
Torlus	Custom	Similar to Gafgyt but commands are different.
Tsunami	IRC	All C2 commands are wrapped inside IRC NOTICE messages.
BASHLIFE	Custom	Similar to Gafgyt but commands are different.

Table 9: Application layer communication protocol of reputable IoT malware families.