CnCHunter: An MITM-approach to identify live CnC servers

Ali Davanian, Ahmad Darki, Michalis Faloutsos
University of California, Riverside

Abstract

How can we identify active CnC servers? Answering this question is critical for containing and combating botnets. Finding CnC servers is not trivial because: CnC servers can change locations expressly to avoid detection, use proprietary communication protocols, and often use end-to-end encryption. Most prior efforts first "learn" a malware communication protocol, and then, scan the Internet in search of live CnC servers. Although useful, this approach will not work with sophisticated malware that may use encryption or a communication protocol that is hard to reverse engineer. In this work, we propose CnCHunter, a systematic tool that discovers live CnC servers of IoT malware efficiently. The novelty of our approach is that it uses real "activated" malware to search for live CnC servers, with CnCHunter acting as a Man-In-The-Middle. As a result, our approach overcomes the limitations of prior efforts. For example, the malware binary knows how to communicate with its server even if in the presence of encryption. We randomly selected 100 IoT malware samples collected between 2017 and 2021, and found their CnC servers. CnCHunter could activate 90% of the malware and automatically find the CnC servers with a 92% precision. Additionally, we demonstrate the potential of our system by activating old Gafgyt and Mirai malware samples and enabling them to communicate with live CnC servers for other recent samples of the same family. This proves that an old malware binary of a family can be used to scan the Internet and find live Cnc servers for that malware family.

1 Introduction

In the battle against IoT botnets, identifying Command and Control (CnC) servers is extremely important. Once the CnC servers of a malware are known, we can start containing its reach and power. For example, the defense could include monitoring or blocking traffic to these destination addresses. This is especially an effective defense in case of IoT devices, because they don't have enough computation power to have sophisticated on-device defenses like Antiviruses. In addition, current breaches can be identified from the network trace once CnC addresses are known. Finally, malware operations can be entirely disrupted once Internet Service Providers and Law Enforcement coordinate efforts to take down known CnC servers.

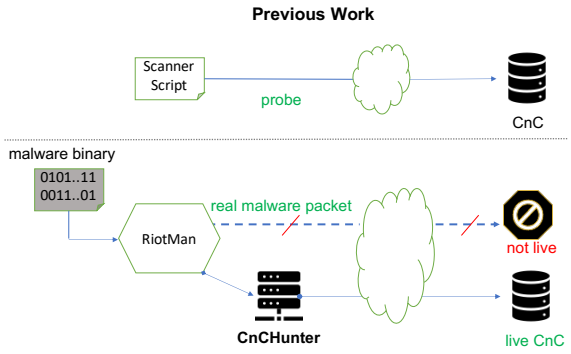

Figure 1: CnCHunter Overview

Finding CnC servers that are currently live is a challenging task. As we elaborate in section 2, CnC servers are very short lived, and once the security analysts find malicious samples, bot masters quickly move their servers and update the samples to include new addresses. Henceforth, by the time a new variant is publicly known, its CnC has already moved to a new address allowing the botnet to survive or revive itself.

Prior work attempts to find CnC servers via active probing using a learned communication protocol. In this approach, one scans the Internet, and attempts to engage with endpoints to confirm a pattern of communication. The main challenge is knowing the details of malware communication protocols. As we elaborate in section 2, IoT malware communication protocols are diverse and hence there is no "one size fits all" solution for active probing of all malware families. Related work such as [19, 23] try to automate the process of learning a botnet communication protocol, however they fall short when the communication protocol has some sort of encryption.

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 1: Application layer communication protocol of reputable IoT malware families.

In this work, we take a different direction. We exploit malware itself to communicate with live CnC servers. Our key intuition is that the communication protocol of a malware *family* barely changes from one sample to another; although it would change from one family to another. Specifically, while the CnC servers in the malware sample might not be alive, the malware sample is capable of talking to a live CnC server of its own family, if we redirect its traffic to the live CnC. This implies that old malware samples can be used to scan the Internet for new live CnC servers.

We present CnCHunter, an automated malware analysis tool that takes as input a malware sample of a family and a list of candidate addresses. CnCHunter automatically finds live CnC servers among the input candidate addresses. Figure 1 illustrates an overview of our system. In summary, we make the following contributions:

- We design and build CnCHunter that can automatically find an IoT malware's CnC servers dynamically.
- We propose a solution to identify live CnC servers of a malware family using old samples if the family is still operational but its new variants are not yet captured and analyzed. We implement the solution and open-source our tool for the community to use [1].
- We showcase the effectiveness of our tool by empirically evaluating it. We show that 18% of live CnC servers that CnCHunter can find are missed by VirusTotal. Furthermore, we show the MitM functionality of our tool by redirecting the traffic of Gafgyt and Mirai samples to live CnC servers of the same family.

2 Guiding Empirical-Derived Insight

Our work is guided by a few key observations in designing CnCHunter. These observations are based on preliminary analysis of malware samples. In the rest of this section, we elaborate on these observations.

[1] please see https://github.com/adava/CnCHunter.

2.1 Diversity of Communication Protocols

We observe that while communication protocols change from one family to another, they tend to remain nearly identical from one sample to another within the same malware family. We define a malware communication protocol as the application layer protocol plus all the contracts that the malware would have using the application layer protocol. For instance, a malware might use HTTP (not HTTPS) but encrypts the messages using RC4 encryption algorithm and expects a particular sequence of messages for a successful communication.

We analyzed the communication protocols of 10 IoT malware families. While there are some similarities, there is no single protocol that can support all malware families or even a majority of them. The result of our analysis is reported in Table 1.

In addition, we evaluated the diversity of communication protocols within a single malware family between 2017 and 2020. We analyzed 1951 samples from 3 different malware families. We built a custom CnC server that imposters the CnC behavior that the malware would expect. Our goal is to measure how many samples from a single family can talk to our imposter without any changes. The result is shown in Table 2. The table shows that in majority of cases, the CnC communication protocol barely changes.

Family from Virustotal	Number of samples	Success
Gafgyt	1451	99.5%
Mirai	402	100%
Tsunami	98	69.5%

Table 2: The percentage of samples that can talk to a single CnC imposter for the malware family of the sample.

2.2 Frequency of Contacting CnC servers

We expect the malware to frequently contact its CnC server. A malware might communicate with different endpoints for

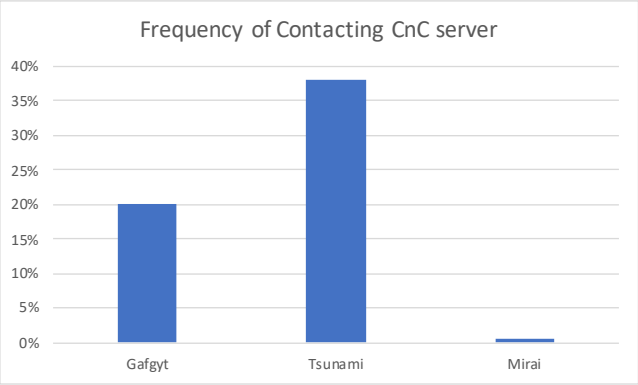

Figure 2: Frequency percentage of CnC connections that is the number of CnC connections to the total number of victim connections.

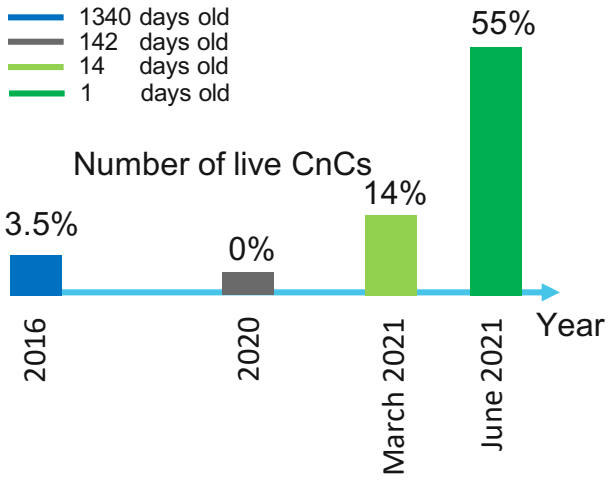

Figure 3: Percentage of malware samples with live CnC servers categorized based on the sample age.

various reasons. For instance, a malware might open a connection to the google to check for internet connectivity. Another example is scanning the internet for finding vulnerable targets. Nevertheless, the malware needs to regularly communicate with its CnC server. We believe that communication with CnC is more frequent than other endpoints. Malware would frequently try to communicate with its CnC even if the CnC is not live.

We manually analyzed nearly 2% of malware samples from our dataset. We randomly selected 40 samples from our dataset for manual analysis. We statically and dynamically analyzed the malware to find its CnC servers. We looked for an IP address that the malware tries to receive a command from and perform something on the victim machine. We further cross examined the IP address with Virustotal to validate that it is actually a CnC server. We looked for a history of communication with malicious binaries, and confirmed that all IP addresses except one have such a history.

Next, we measured the frequency of contacting CnC connections relative to the total number of connections that the victim machine would make. Figure 2 shows the result. On average, 18% of the infected machine's communications is with the CnC server. An exception is with Mirai family. This is because Mirai samples heavily scan the Internet, and hence the number of CnC connections looks small in comparison to the scanning activity. Nevertheless, the absolute value of number of CnC connections would be still the largest in comparison to other IP:PORT addresses even for Mirai family.

2.3 Short-Lived CnC servers

Malware CnC servers rapidly change their addresses. P. Vervier and S. Yun observe that 90% of malware download servers disappear after 5 days [22]. Similarly, R. Tanable et al. report that bots lose the connection to their CnC servers within one or two days [21]. This is because malware intelligence systems identify them and publish their addresses as malicious that will result in the botnet communication disruption. Henceforth, bot masters move their server to a new address and ask the bots to accordingly talk to the new address, or rebuild the botnet from scratch. Additionally, some malware use DNS addresses that can quickly change the corresponding IP address.

We measured the CnC liveness of the samples that are publicly available (e.g. through malware databases like MalwareBazaar [1]). We manually analyzed 100 random samples collected between 2016 and 2021, and found their CnC servers [2]. We define a CnC server live, if the malware can initiate a successful connection and the messages exchanges match the expected malware signature. We tried the malware connection from different source addresses using VPN connections. We tried United States, Brazil, India and Romania as sources. Figure 3 illustrates the percentage of samples that had a live CnC server based on the age of the sample. Roughly, only half of the samples have a live CnC server by the time the sample is publicly known.

3 Malware Emulation

We emulate the malware execution and analyze it dynamically. At the heart of every automated malware analysis is malware execution. The malware execution is either emulated or virtualized because there could be harm to the underlying guest system. Furthermore, emulation and virtualization give the analyst more control over the malware in contrast to a

[2]Our dataset is available upon request.

real execution. Qemu [3] is a reputable emulator and is extensively used by the community for malware analysis tasks. In this work, we focus on the malware emulation and Qemu although the techniques discussed here are applicable to other infrastructures.

Automated malware emulation is done with a set of goals in mind. These goals might be collecting one or multiple of the following: the instruction execution trace, the call stack trace, the system call trace, the accessed files, the memory snapshot(s) and the generated network traffic. The log of each of the above items provide a different view on how the malware behaves. In this work, we are interested in the network traffic, and the system call traces. The former provides us the data for CnC communication analysis while the latter allows us verify the correctness of the emulation.

3.1 Automated Analysis process

Our malware analysis process is fully automated. In general, malware emulation process is very similar regardless of the analysis goal. The process is as follows:

- **Prepare the VM configuration**: the architecture and the platform that malware can execute on should be determined. Based on the configuration, a VM should be selected for the malware emulation.
- **Logging setup**: based on the analysis task, the logging functionalities should be enabled. This could be a custom code to record every executed instruction, or an off-the-shelf tool like tcpdump to log the traffic.
- **Copying the malware**: the malware should be copied to the VM for the emulation. Based on the setup, this could be modifying the file system or uploading the sample to the VM.
- **Executing the malware**: this step involves booting up the operating system, and then executing the malware. A common practice is to configure the OS to start the malware execution at the startup stage. There should be a timeout for the malware emulation, otherwise the analysis never terminates.
- **Collecting the logs:** after the execution times out, the generated log should be collected and stored for later analysis.
- **Clean up:** finally, there should be a cleanup stage because the malware would likely corrupt the system. This could be done either by entirely deleting a VM instance or deleting the filesystem (entirely or partially).

3.2 RiotMan

We rely on RiotMan [6] for our malware emulation process. RiotMan follows the same set of steps explained in subsection 3.1, and uses Qemu system level emulation. We further explain how RiotMan performs each of the steps described earlier.

For the **VM configuration**, RiotMan relies on an iterative learning process. It starts by executing the malware with a clean Linux Kernel. If the malware fails to successfully execute, RiotMan looks up the last system call and finds the filename (if any) that resulted in an error. RiotMan has a database of different IoT firmware files, and searches for the file in this database. It either retrieves this file from the database or just creates a file with the expected name in the path that the system call requested. RiotMan continues this process by re-executing the malware until no system call results in errors.

For **logging**, RiotMan uses *strace* and qemu traffic record functionality. RiotMan executes the malware (in the VM) with the *strace* logging:

```
/usr/bin/strace -ftttT -s999 /malware 2>
$RESULT_DIR/syscalls/$MALWARE_FILE.log >
$RESULT_DIR/log/$MALWARE_FILE.out
```

RiotMan enables Qemu network bridging functionality, and records the traffic in pcap format via the following Qemu option:

```
-object filter-dump,id=wan,netdev=wan,
file=$DIR_TO_PCAP/$NAME.pcap
```

RiotMan provides a separate file system for the emulated VM. This file system is a clean "*ext4*" partition with all the malware dependant files. RiotMan **copies** the malware binary to the partition root, and updates */etc/rc.local* to automatically **execute** the malware binary after the startup. Finally, RiotMan directs Qemu to use the staged partition as the file system:

```
-drive file=$PATH_TO_FILESYSTEM,index=0,
media=disk,format=raw
```

RiotMan terminates the Qemu process after 20 minutes of execution. Then, it **collects** the VM system call log from the VM filesystem. Afterwards, it **cleans up** by deleting the filesystem. After the emulation, the system call log and the recorded traffic will be available in an analysis folder for further examinations.

4 CnCHunter Design and Implementation

CnCHunter exploits a malware sample to find live CnC servers of its family. CnCHunter takes as an input a malware binary. It also takes as input a list of IP and port addresses that are potentially CnC servers; this list might be the result of a massive scanning or based on the past reputation. CnCHunter analyzes the network traffic that the malware generates. It finds (A) the CnC servers that the malware tries to contact, and (B) other live CnC servers that this malware sample can

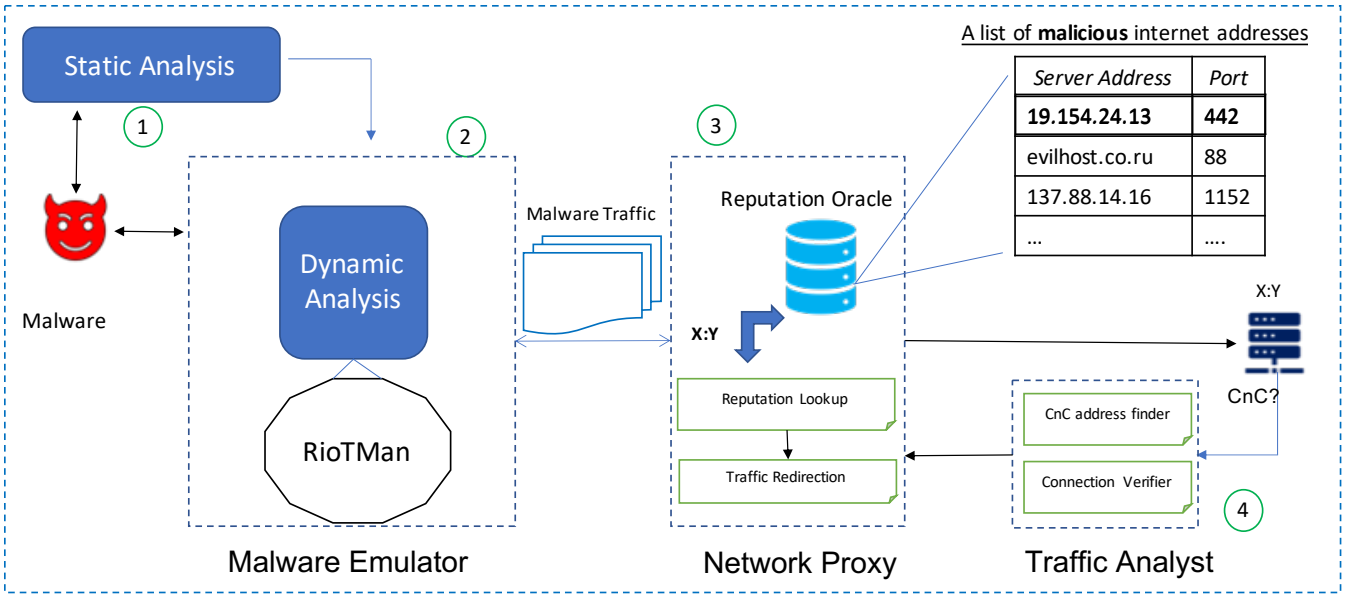

Figure 4: CnCHunter System

successfully communicate with. Figure 4 shows CnCHunter system design.

CnCHunter starts by statically analyzing malware. First, we determine the compatible binary architecture and platform. Second, CnCHunter can be configured to look for hardcoded IP addresses. These IP addresses might be CnC servers.

Next, the **malware emulator** component executes the malware. We use RiotMan as the Malware emulator because it is lightweight and effective for IoT malware analysis[3]. The emulation results in a traffic generation that the **Traffic Analyst** component would look into. This component finds the CnC address from the traffic, and informs the **Network Proxy** component about it. Network proxy replaces this address with an item from the list of candidate CnC addresses and redirects the traffic to those candidate nodes in the next iterations of the emulation. **Traffic Analyst** analyzes the responses from candidate addresses, and decides whether the address is a CnC based on the success of the communication. In the rest of this section, we expand on the details of the Network Proxy and the Traffic Analyst components.

4.1 Network Proxy

Network proxy goal in our system is to redirect the malware CnC traffic to some candidate addresses. The candidate addresses are inputs to the network proxy. Ideally, the candidate addresses are the CnC servers of other samples (but not yet publicly known) of the same family that the analyzed malware belongs too. Such a list can be compiled via a scanning phase, and/or based on the past reputations that the IP blacklists track. However, CnCHunter can still work even if the entire IP space are inputted as the candidate addresses; it would only slow down the system.

The CnC address of the malware is also an input to the Network Proxy module. The CnC address of the analyzed malware is not known beforehand, and network proxy expects the Traffic Analyst module to provide this address; we will elaborate on this in the subsection 4.2. The malware exchanges traffic with the CnC ultimately through an IP address. However, the malware might use a DNS address, in which case there is a DNS resolution process. Our network proxy module supports both DNS and IP based CnC addresses.

We use iptables for proxying IP based CnC traffic. A natural solution would be asking iptables to replace the given IP and PORT destination with a candidate CnC address at the network perimeter. If we wanted to do so, we should have used the POSTROUTING chain of iptables. However, this functionality is not supported and redirecting traffic to another address is not allowed on iptables installed at network perimeters. That said, iptables can change a destination address if it is installed on the source. In this case, one would need to use the OUTPUT chain. For this reason, we rely on the guest's (infected machine) iptables to provide the redirection. We add the following iptables OUTPUT chain rule to the guest startup script, and we make sure it runs before the malware executes:

```
iptables -t nat -A OUTPUT -p tcp -d $CNC_IP
--dport $CNC_PORT -j DNAT --to-destination
```

[3]other malware malware analysis framework/emulators can be used as long as they can record traffic in pcap format.

```
$CANDIDATE_IP:$CANDIDATE_PORT
```

For DNS based addresses, we take a different approach. Operating systems use different DNS resolution methods. Proprietary operating systems like Linux start by looking up the DNS address in a host file that maps DNS names to IP addresses. They use other methods only if this method fails[4]. We take advantage of this process, and modify the host file of the guest machine that would map DNS addresses to IPs. We simply add an entry that maps the CnC DNS address to the candidate address, similar to the IP case, the CnC DNS address is found by the traffic analyst module. Doing so, the traffic would automatically be redirected to the candidate address that we desire. The above can be done using the following:

```
echo "$CnC_DNS_ADDR $CANDIDATE_IP" >>
/etc/hosts
```

Additionally, Network Proxy taps the traffic for analysis by the Traffic Analyst. We use Qemu bridging functionality. The following shows the option we provide to Qemu for the network bridging:

```
-netdev bridge,id=wan,br="$BR_WAN,
helper=$Qemu_bridging_HELPER"
```

Finally, to communicate with the candidate addresses, the guest should have internet connection, so we activate forwarding on the Linux host that is the network proxy:

```
sudo sysctl -w net.ipv4.ip_forward=1
sudo sysctl -w net.ipv4.conf.
"$BR_WAN".proxy_arp=1
sudo iptables -t nat -A POSTROUTING
-o "$IF_INET" -j MASQUERADE
```

4.2 Traffic Analyst

Traffic Analyst component of CnCHunter accomplishes two tasks. First, it finds the CnC address that the analyzed malware tries to contact within the traffic that the infected virtual machine would generate. We look at the VM traffic instead of the malware process traffic so the malware evasion techniques wouldn't affect our analysis; regardless of malware efforts, the VM traffic has to go through our proxy.

The generated traffic by the malware is an input to our *FindCnC* Algorithm illustrated in algorithm 1. *FindCnC* analyzes every destination address and assigns a score that shows the likelihood of being a CnC server for that address. An address is either an IP and PORT tuple or a DNS address. In lines 5-10, we analyze each packet and count the number of times addresses (IP:PORT or DNS) and ports are contacted.

For TCP connections, we count the failure or success to the target IP address. Malware keeps trying to contact its CnC even when the TCP handshake fails. We build upon this observation and count the number of times connection to an IP address fails by looking at the TCP RST or SYN flags. Otherwise, if there is no failure, we count the number of successful connections.

For DNS based addresses, we count the number of times the DNS queries fail. This is based on the observation that a blocked CNC DNS address would not resolve, and hence there will not be a connection to an IP address. We filter out the traffic from unrelated protocols: icmp, dhcp, arp and ntp.

After processing all packets and pre-processing the number of connections and ports, we can assign a score to each address (lines 11 to 16 of algorithm 1). For DNS based addresses, we check the reputation of the domain. We whitelist the top X (X is 1000 but is configurable) number of domains based on the Alexa ranking. This is to avoid false positives. Note that we check the entire DNS address rank, and not only the second level domain (Microsoft ranking is different than its subdomains). This means that a cloud based CnC address (e.g. hosted on Microsoft or Amazon) would not have a good Alexa ranking. If the DNS address passes the reputation check, the score is the number of times it was queried.

For IP and Port based addresses, the score is proportional to the number of times it was contacted. Additionally, the score is inversely proportional to the number of times the port was used in the entire traffic. This is based on the observation that the traffic to CnC is through a unique destination port while other malware activities such as scanning for vulnerable hosts target the same port. Henceforth, to calculate the score, we can simply divide the number of connections to the number of times the port was used; we found that this simple formula works well in practice. We use Python pyshark library [15] to implement the algorithm 1.

Second, Traffic Analyst component finds whether a connection to a candidate CnC address has been successful. The success of a connection could mean different behaviors for different protocols. Furthermore, encryption and checksums make differential analysis hard; looking at whether a pair of messages is different than before. Therefore, we rely on an analysis that is protocol agnostic and does not rely on lack of encryption.

We find that a simple analysis of the number of synchronizations between an address and the malware is a reliable indicator. To illustrate the concept, we make an analogy to the human communication. If two persons do not understand each other's language, they restart the conversation and repeat themselves. Similarly, if a candidate address listens on our contacted port but would not speak our malware application layer protocol, it assumes the connection was not properly established and would ask for a synchronization. This allows us to find successful connections by looking at those candidate addresses that respond by much fewer SYN flags on. To find these addresses, we run a simple outlier analysis on the number of times SYN has been set for the candidate address.

[4]This order can be modified.

Algorithm 1 FindCnC

Input: Packets
Output: Addresses_Analysis

```
Addrs_Analysis ← {}          ▷ A hashtable tracking the number of connections to each address (ip:port or DNS).
Ports ← {}  ▷ A hashtable tracking the number of times a destination port is seen.
Scores ← [] ▷ A list of tuples storing addresses with their CnC likelihood score
i ← 0
while i < len(Packets) do
    pkt ← Packets[i]
    analysis_res ← Analyze_Address(pkt)
    if analysis_res! = NULL then
        Update_Addresses(analysis_res, Addrs_Analysis)
        Update_Ports(analysis_res, Ports)
while j < len(Addrs_Analysis) do
    addr ← Addrs_Analysis[j]
    if IS_DNS(addr) and Alexa_rank(addr) then
        Scores[j] = Calculate_DNS_Score(addr)
    if IS_IP(addr) then
        Scores[j] = Calculate_IP_Score(addr, Ports)
Sort_Desc(Scores)
return Scores
```

5 Evaluation

We empirically evaluate CnCHunter. Our goal is to answer the following questions:

1. Can CnCHunter accurately identify a sample's CnC address?

2. Can a live CnC server of a malware family serve the requests of the malware family's old samples with dead CnCs?

3. How can we use CnCHunter in practice to find live CnC servers using old samples?

We answer the first question by comparing CnCHunter's results with the ones compiled from manually finding CnC servers of 100 samples. All these samples are for MIPS architecture since it is, reportedly, the least explored architecture by the security community [6] and more specifically is a target for IoT devices. We elaborate more on this in subsection 5.1. Next, we answer the second question by using CnCHunter to redirect the traffic of an old gafgyt sample's traffic to a recent live CnC server (please see subsection 5.1 for more details). Finally, we answer question 3 by showing how we use CnCHunter to find a live Mirai CnC server that can not be detected by analyzing the publicly available samples.

5.1 CnC finding precision

Our approach is prone to both false positives and false negatives because it is general; in contrast to a signature matching approach that might be false positive free but at a much higher false negative rate. We need a ground truth in order to evaluate the CnCHunter's CnC finding functionality. Such a ground truth does not exist even though finding CnC address of IoT malware has been conducted in previous work [21, 22].

We start by compiling such ground truth manually. We randomly selected 100 MIPS architecture samples collected between 2016 to 2021. The samples were collected daily from VirusTotal. We expect at least 5 engines to identify the sample as malicious. We collect all elf samples. Later, in our static analysis stage, we only analyze MIPS samples. The reason is the focus of our work that is IoT malware, and MIPS samples are all IoT malware according to our dataset. Our query is:

```
fs:1d+ type:elf positives:5+
```

We observe samples of Mirai, Gafgyt, Tsunami, Remaiten, LightAidra and VPNFilter in our dataset. That said, majority of samples are from Gafgyt and Mirai. This follows the trend previously reported that Mirai is the largest malware family [2, 11], and Gafgyt is the second largest malware family [20].

We manually identified the CnC servers of these samples. Our manual analysis involved both Static and Dynamic analysis. Statically, we try to find hardcoded IP addresses and dynamically we look at the generated traffic by the malware. In the traffic, we look for known patterns of CnC communication. We further cross examine our analysis' results by checking the reputation of the addresses from VirusTotal. We find that 91% of samples have IP based CnC addresses and 9% have DNS based addresses. Furthermore, 29 samples had live CnC servers.

We measured CnCHunter's CnC finding functionality precision based on our manually crafted ground truth of 100 MIPS samples. CnCHunter was not able to activate 10% of the samples. We define activation as execution of the malware to the point where traffic is generated from the malware process. The failures were mainly due to "illegal instruction error" that is a Qemu emulation limitation.

CnCHunter has a precision of 92% on the activated samples. Out of the live CnC servers that CnCHunter finds, 18% are missed by VirusTotal and are reported as not malicious. This shows that there is room for improvement when it comes to identifying CnC addresses by Antivirus engines. Table 3 shows a summary of our results.

Samples	Activated	Precision	Missed live CnCs by VT
100	90%	92%	18%

Table 3: CnCHunter CnC finding precision

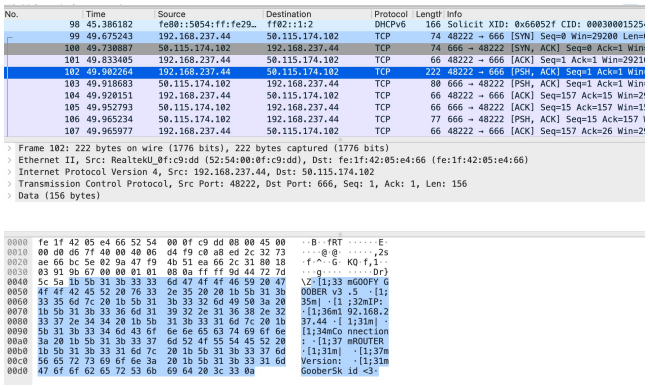

Figure 5: Malware sample first packet with its CnC

5.2 Case Study: MitM functionality

In this case study, we present a proof of concept that shows how an old malware sample can successfully be exploited to communicate with a live CnC server of a recent sample. More precisely, we show:

- How CnCHunter automatically identifies the malware CnC from its traffic.
- How CnCHunter automatically Man-In-The-Middles the malware and make it talk to another CnC server.

To this end, we find a live CnC server for a recently found malware and examine whether an old sample from the same family can talk to it. As we explained in the subsection 2.3, finding a live CnC server is an extremly hard task. Most IoT CnC servers are very short lived because bot masters move them to stay hidden. To address this problem, we rely on CnCHunter itself to find a live CnC server through analysis.

Recent Gafgyt malware sample: We took one of the samples in our dataset that has a live Cnc server that would communicate with the malware and receives commands from the CnC. Below at Table 4 is the malware hash and the CnC address that was live on Mar 30, 2021.

Hash	CnC Address
785f781c4bbd96b207e4d0c77a5afe36a13126cf9a9c33f7afda6ed12103ea6b	50.115.174.102:666

Table 4: Gafgyt malware sample with live CnC server

Figure 6 shows the first packet that the infected machine would send to CnC after the successful TCP handshake. Afterwards, CnC sends a "Scanner On" and a "Fat Cock" message to the infected machine. The communication continues with a series of Pings and Pongs messages.

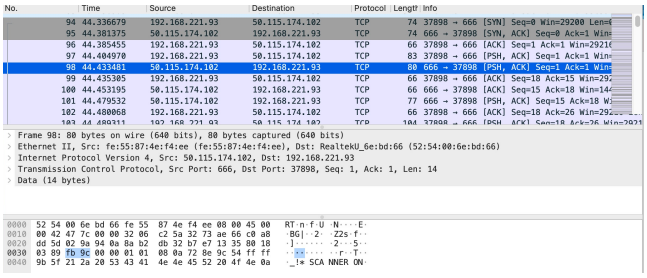

Figure 6: Old Malware sample receiving the "Scanner On" command from another CnC server of the family.

Old Gafgyt sample: We randomly selected an old Gafgyt sample from the dataset we mentioned at subsection 2.1. This sample has been submitted to Virustotal on 2017-04-05. The sample hash and the CnC address found by CnCHunter is reported in Table 5. The malware can not communicate with its CnC because it is not live anymore. As we mentioned earlier, we aim to redirect its traffic to the live CnC server for another sample from the same family.

Hash	CnC Address
9ad3a408bd09e45a68408aa25caecda695693d49b741c694c076190a6b86284f	138.197.104.187:23

Table 5: Gafgyt malware sample with a dead CnC server

Result: We find that the old sample can successfully communicate with the live CnC server. Originally, the old malware sample (with the dead CnC) generates around 30KB of traffic and the connection to the CnC address terminates; the handshake wouldn't complete. However, with the live CnC server, the old sample would generate around 6MB of traffic in a few minutes and successfully exchanges packets with the CnC. Similar to the new sample, the malware receives the "Scanner on" command from the CnC server. Figure 6 shows this message from the CnC.

5.3 Case Study: Active Probing

As we explained in the introduction, proactively finding CnC servers is really an important step towards defending against Zero-Day malware. In order to evaluate the efficacy if CnCHunter in active probing, we aim to accomplish the following goals:

- Finding some candidate addresses for active probing
- Using CnCHunter to probe the candidate addresses

It has been previously demonstrated that some IP subnets tend to be used for malicious activities [7]. We build on this

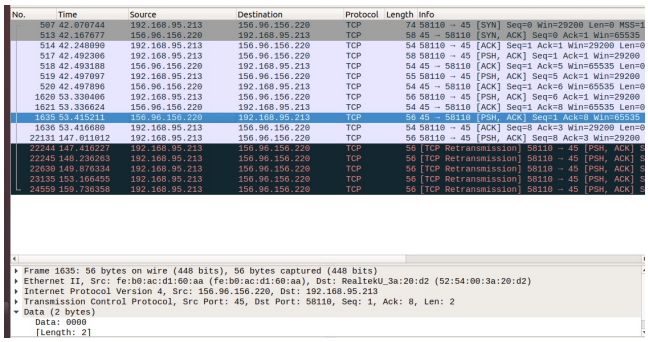

Figure 7: Mirai Sample exploited to active probe and find a live hidden CnC server.

observation, and probe IP subnets from our malware dataset that had hosted more than one CnC server. We found 4 such IP subnets with CIDR[5] 24.

Next, we used masscan [10] to scan these IP subnets for frequent appearing ports in our dataset. We conducted this experiment on July 16, 2021. We found two candidate addresses in 156.96.156.0/24 subnet on ports 45 and 7854.

Finally, we used CnCHunter to MitM a malware sample and probe these IP addresses. According to our dataset, the aforementioned subnet was hosting Mirai CnC servers. Henceforth, we used the following Mirai sample for the probing:

15c48d1133f994942a1fbe17bc95535689319778375ec15b9677229cfe2a543

Result: CnCHunter found that 156.96.156.220:45 is hosting a Mirai CnC server. On the same day, we downloaded MIPS samples from Malware Bazaar [1] and used CnCHunter to find the samples' CnC servers. According to our analysis, no sample was communicating with this CnC address. Furthermore, we queried VirusTotal, and even after 4 days, the IP address is reported benign. However, by manually analyzing the traffic we can confirm this address was indeed hosting a CnC server that is hidden in the wild. Figure 7 shows the CnC traffic and the CnC server Mirai signature bytes response.

6 Related work

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

6.1 IoT Malware Behavior Analysis

Studying the behavior of IoT malware has become a hot topic both for Academia and Industry. Prior work on this topic can be grouped to two categories. The first category focuses on characterizing the behavior of a single Malware [2, 11, 14]. [2, 11] employ static and dynamic analysis, and Internet scanning to study the echo system of Mirai malware. They characterize the infected IoT devices, the infection vectors and the evolution through Mirai's life cycle. S. Herwig et al. study the echo system of the Hajime IoT botnet [14]. While these prior work provide a deep insight into a single malware family life cycle, we are interested in a general method to study CnC servers of all malware families.

The second group of prior work in this category characterize the behaviors of several malware families at the same time [5, 6, 18, 20]. Authors of [5] employ static analysis techniques to analyze 10 malware families and find the evolution of the malware within a family as well as comparing to other families. In contrast, E. Cozzi et al. [4] employ dynamic analysis techniques to characterize behaviors of different malware families and their similarities. A. Darki et al. [6] not only characterize different behaviors of 8 malware families but also provide a tool that can automatically configure the dynamic analysis environment for a malware sample. In contrast to the aforementioned work, [18] proposes a generic method to direct a malware sample attack to a compatible IoT device.

Finally, several work target the CnC communication aspect of the IoT malware [20–22]. [20] proposes a framework to study the IoT malware from 5 perspectives including the C&C Infrastructure while [21, 22] analyze the CnC traffic of the malware. Although these prior work provide details about different behaviors of malware, none would address the problem of dissecting C&C communication protocol automatically and generally when the CnC server of a malware sample is not live that is a common case.

6.2 Active Probing

The closest category to CnCHunter are the prior work on active probing. Active probing is the practice of actively scanning Internet in pursuit of finding live targets; in our case, live CnC servers. Such efforts require an understanding of the malware communication protocol and the encryption algorithm, if any. Henceforth, searching for CnC servers of a single malware family is a much easier task [2, 9]. [9] actively scans the Internet searching for Dark Comet Trojan servers, while [2] scans the Internet for Mirai CnC servers and the commands it would issue. In addition, there are prior work that took the first steps in automating the active probing for a more widespread group of malware families [19, 23]. The main problem with these two prior work is that they can not handle the communication protocol when there is encryption. In contrast, in our case, the malware knows the encryption protocol and can communicate with its CnC without our meddling.

[5] Classless Inter-Domain Routing

6.3 Malware Network Trace Analysis

Studying the network trace of malware has been the subject of many related work [8,12,13,16]. These work try to find botnet traffic from a network trace. Although these approaches don't occupy bandwidth and are lightweight since they don't require Internet scanning, they are limited to the network trace that they have access to. In other words, they can not find the CnC communications that have not been made yet to their sample network. [16] reports that the network traffic is the best indicator of malicious activity, usually weeks before a binary is reported malicious; a finding that was previously reported by [17]. We build on this finding by proposing an approach that can find malicious connections without the particular binary that is still a zero day.

7 Conclusion and Future work

We showed that finding live CnC servers is a challenging task particularly because they are short-lived and the malware communication protocols are diverse. We presented CnCHunter, an automatic malware analysis tool that can find live CnC servers using a malware binary sample and a set of candidate addresses. We showed how an old Gafgyt sample is capable of finding a live CnC server for a recent Gafgyt sample.

In future, we plan to extend our work in two ways. First, we plan to automate the task of finding candidate addresses. Currently, CnCHunter relies on a list of addresses provided as an input. We vision to use publicly available malware intelligence data and find networks with a high likelihood of having the target CnC server. CnCHunter would start the search by giving priority to networks with a higher likelihood of having the CnC servers.

Second, we plan to extensively evaluate our system using a variety of malware family samples. We hope we can provide an online system that can report the live CnC servers based on the output of CnCHunter. We believe this would largely benefit the security community and Internet users.

8 Acknowledgement

The authors would like to especially thank Martina Lindorfer for her comments, and providing dataset for recent malware. We also would like to thank VirusTotal for giving us academic access. Finally, we thank BlackHat reviewers for giving us the opportunity to present this work in BlackHat US 2021.

References

[1] Malwarebazaar database. https://bazaar.abuse.ch/.

[2] 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 *26th USENIX security symposium (Security 17)*, pages 1093–1110, 2017.

[3] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *USENIX annual technical conference, FREENIX Track*, volume 41, page 46. Califor-nia, USA, 2005.

[4] Emanuele Cozzi, Mariano Graziano, Yanick Fratantonio, and Davide Balzarotti. Understanding linux malware. In *2018 IEEE symposium on security and privacy (SP)*, pages 161–175. IEEE, 2018.

[5] Emanuele Cozzi, Pierre-Antoine Vervier, Matteo Dell'Amico, Yun Shen, Leyla Bilge, and Davide Balzarotti. The tangled genealogy of iot malware. In *Annual Computer Security Applications Conference*, pages 1–16, 2020.

[6] Ahmad Darki and Michalis Faloutsos. Riotman: a systematic analysis of iot malware behavior. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, pages 169–182, 2020.

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

[8] Lorenzo De Carli, Ruben Torres, Gaspar Modelo-Howard, Alok Tongaonkar, and Somesh Jha. Botnet protocol inference in the presence of encrypted traffic. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2017.

[9] 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 *2017 IEEE symposium on Security and Privacy (SP)*, pages 770–787. Ieee, 2017.

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

[11] Harm Griffioen and Christian Doerr. Examining mirai's battle over the internet of things. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 743–756, 2020.

[12] Guofei Gu, Vinod Yegneswaran, Phillip Porras, Jennifer Stoll, and Wenke Lee. Active botnet probing to identify obscure command and control channels. In *2009 annual computer security applications conference*, pages 241–253. IEEE, 2009.

[13] Guofei Gu, Junjie Zhang, and Wenke Lee. Botsniffer: Detecting botnet command and control channels in network traffic. 2008.

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

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

[16] Chaz Lever, Platon Kotzias, Davide Balzarotti, Juan Caballero, and Manos Antonakakis. A lustrum of malware network communication: Evolution and insights. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 788–804. IEEE, 2017.

[17] Martina Lindorfer, Alessandro Di Federico, Federico Maggi, Paolo Milani Comparetti, and Stefano Zanero. Lines of malicious code: Insights into the malicious software industry. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 349–358, 2012.

[18] Tongbo Luo, Zhaoyan Xu, Xing Jin, Yanhui Jia, and Xin Ouyang. Iotcandyjar: Towards an intelligent-interaction honeypot for iot devices. *Black Hat*, pages 1–11, 2017.

[19] Antonio Nappa, Zhaoyan Xu, M Zubair Rafique, Juan Caballero, and Guofei Gu. Cyberprobe: Towards internet-scale active detection of malicious servers. In *In Proceedings of the 2014 Network and Distributed System Security Symposium (NDSS 2014)*, pages 1–15, 2014.

[20] 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 *30th USENIX Security Symposium (Security 21)*.

[21] 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 15th International Conference on Availability, Reliability and Security*, pages 1–10, 2020.

[22] Pierre-Antoine Vervier and Yun Shen. Before toasters rise up: A view into the emerging iot threat landscape. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 556–576. Springer, 2018.

[23] 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 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 179–190, 2014.