University of Amsterdam

MSc System and network Engineering

Research Project 1

Techniques for detecting compromised IoT devices

Project Report

Ivo van der Elzen
Jeroen van Heugten

February 12, 2017

Abstract

The amount of connected Internet of Things (IoT) devices is growing, expected to hit 50 billion devices in 2020. Many of those devices lack proper security. This has led to malware families designed to infect IoT devices and use them in Distributed Denial of Service (DDoS) attacks. In this paper we do an in-depth analysis of two families of IoT malware to determine common properties which can be used for detection. Focusing on the ISP-level, we evaluate commonly available detection techniques and apply the results from our analysis to detect IoT malware activity in an ISP network. Applying our detection method to a real-world data set we find indications for a Mirai malware infection. Using generic honeypots we gain new insight in IoT malware behavior.

1 Introduction

September 20th 2016: A record setting Distributed Denial of Service (DDoS) attack of over 660 Gbps is launched on the website krebsonsecurity.com of infosec journalist Brian Krebs [1]. A few days later, web hosting company OVH reports they have been hit with a DDoS of over 1 Tbps [2]. These attacks were not only unprecedented in volume [3], but were performed using direct traffic without using any of the reflection/amplification techniques commonly associated with large-scale attacks [1].

DDoS attacks are not a new phenomenon. Before these events, most large-scale DDoS attacks were performed using reflection/amplification of UDP protocols such as DNS, NTP, SSDP and others [1]. These attacks rely on the availability of hosts serving vulnerable protocols to reflect the attack to the victim, and also to amplify the attack by increasing the volume. Two examples of the most powerful of such attacks to date were both described in detail by internet security company Cloudflare [4] [5]. Although still a significant source of DDoS traffic, those attacks can be mitigated at the source by implementing source address validation (BCP-38/84) [6] [7]. Efforts are underway to get these mitigating factors implemented across the internet [8].

The attacks on krebsonsecurity.com and OVH were leveraged using compromised Internet of Things (IoT) devices like IP cameras and Digital Video Recorder (DVR) boxes. The devices became infected with malware by simple Telnet dictionary attacks and were made part of a botnet, after which they were used the DDoS attacks. As direct connections were used, this would suggest a large number of compromised devices. This is because direct attacks do not amplify their traffic using other hosts not directly compromised by the malware. A couple of days after the attacks, the source code for the malware (dubbed “Mirai”) was published [9] [10].

With the rise of these IoT based DDoS attacks [11] it has become clear that this problem will only become bigger in the future, unless IoT device vendors accept responsibility and provide a mechanism for automatic security updates. Until such time that most vulnerable IoT devices are updated, it is important for network operators to have tools available to detect intrusion attempts and infected devices before they participate in a DDoS attack or other malicious activity. In this research we aim to determine which techniques are available to an ISP, and how they could be used to reach this goal.

1.1 Outline

Related work on the subject is outlined in section 2. Section 3 details the malware analysis of two leading IoT malware families; BASHLITE and Mirai. Section 4 gives an overview of detection techniques and it’s application to IoT malware. Our experiments in applying the techniques to real-world data are described in section 5. Finally, our results are discussed and concluded in section 6.

1.2 Approach

Our main research question is the following:

- Can leading IoT malware be detected using network-based techniques available to ISPs?

To answer this question, several sub questions have been formulated:

(A) What are important common properties of leading IoT malware?

(B) How can these properties be used to detect IoT malware activity in an ISPs network?

(C) Can generic honeypots be used to collect properties of new IoT malware?

To answers these questions, we analyze existing research on IoT malware available on various blogs, websites and scientific resources. Our goal is to learn about common properties of IoT malware; what techniques they employ and what their characteristics are. By analyzing malware source code we find common properties that apply to multiple IoT malware families.

Based on this analysis we create a description of the life cycle of an IoT malware infection. The characteristics of this infection life cycle are used to determine which techniques can be used to detect infected devices or intrusion attempts. We evaluate generic detection techniques likely available to network operators; such as NetFlow, packet capture analysis, honeypot data, open/closed ports, Domain Name System (DNS) analysis, etc.

We validate these techniques by applying them to real data retrieved from NetFlow and darknet packet captures. Additionally, we use honeypots to gather new information about the changing landscape of IoT malware.

1.3 Scope

In this research we describe several distinct stages in the malware infection life cycle. We focus on the early stages of the life cycle that are relevant to the spreading of the malware. These include scanning for other devices, attacking these devices and infecting them with the malware. Command and Control (C&C) traffic and DDoS attacks performed by the malware are not researched in any detail.
Due to the mercurial nature of IoT malware, it becomes difficult to be comprehensive in describing the behavior in a generic way. When the life cycle model is described in a broad manner it would inevitably also describe other types of malware. Or when described more narrowly the model will describe only one or a few different IoT malware variants. We choose to keep the model fairly narrow, but recommend to keep updating it to remain relevant. This means that when our approach is used in practice, the life cycle model used needs to be continually updated as well.

2 Related work

Various research has been done on the subject of detecting IoT malware, but most of this research is non-scientific (e.g. web logs, independent non-professional researchers). Nevertheless, formal research on IoT botnets has been done and is still ongoing.

In August 2015, Pa Pa et al. published a paper “IoTPOT: Analysing the Rise of IoT Compromises” [12]. They analyzed Telnet based scans in darknet, and used that information to create a honeypot that emulates specific Telnet services. With this honeypot they were able to analyze ongoing attacks in depth. This research shows that honeypots are a viable method of gathering information about IoT malware, but offers no opinion on how to use that information to detect attacks on an ISP level.

In January 2014, Lin, Chen, & Hung published a paper [13] about detecting botnets with a combination of an artificial fish swarm algorithm (AFSA) and support vector machines (SVM). This research focuses on identifying essential botnet attributes by analyzing data with optimized algorithms.

In August 2016, DARPA announced that it is going to fund the CAMELIA project with $9.4 million [14]. This project focuses on detecting IoT malware by analyzing the electromagnetic emanations of IoT devices. By comparing them with a database of signals of normal operation, it would be possible to detect running malware. However interesting, this approach is not practical for most network operators, as it requires special equipment, and physical proximity to the devices.

Other organizations, such as MITRE, are also investigating ways to detect infected IoT devices. In October 2016 they awarded a $50.000 bounty for this research [15], with the focus on non-traditional means of detection. This shows that this research subject is highly relevant.

3 Malware analysis

In this section we analyze the source code of two leading IoT malware families, BASHLITE and Mirai, to discover their common properties.

3.1 BASHLITE

BASHLITE is a type of malware that infects Linux devices and uses these devices to launch DDoS attacks. The malware is also known as **Gafgyt**, **Torlus**, **Lizkebab** and some others [16] [17]. It was created by a programmer working under the pseudonym "Sinden" (among other synonyms) [18].

The malware consists of server and client code. The server code is designed to run on one or more C&C servers. With those servers, the botnet owner can control the herd of bots (infected devices) running the client code. The two communicate with each other with a custom protocol inspired by IRC.

3.1.1 Scan

The client code incorporates a Telnet scanner "lel", referring to the authors, the "lelddos" group [19]. The Telnet scanner generates random /24 IP-subnets (256 addresses) [20]. Within those subnets, the malware scans the addresses sequentially (starting with .1 and ending with .255).

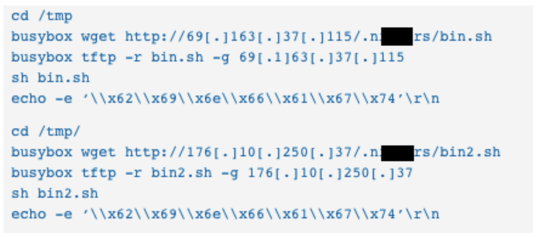
When generating the subnet, the malware checks if the subnet is part of the IANA special purpose addresses list (RFC 5735 [21]). If this is the case, the subnet will not be scanned.

The scanner sequentially loops over the previously generated IP addresses and creates a raw socket for each IP. There is no specific window size set, but the system default is used. The destination port of the connection is port 23 (Telnet), which is hard coded in the malware.

If an IP-address has an open Telnet port, the malware will try to login to the device using a list of predefined username / password combinations.

The first variant of the malware [23] (ELF_BASHLITE.A [24]) was first seen in September 2014. This variant does not take many actions once successfully logged into a device: It checks whether the device runs a BusyBox shell [25]. If this is the case it executes a command to echo the string "gayfgt", as can been seen in appendix B.

A later variant (ELF_BASHLITE.SMB [26]), detected in October 2014, is more harmful. This variant first downloads two scripts from a remote server, as can be seen in Figure 3. Those scripts are designed to gain full access to the system by abusing the ShellShock exploit [23] [27]. One of the shell scripts (see appendix A) downloads the malware for a wide range of architectures using wget and executes them all (hoping one will work).



```
cd /tmp
busybox wget http://69[.]163[.]37[.]115/.n[redacted]rs/bin.sh
busybox tftp -r bin.sh -g 69[.1]63[.]37[.]115
sh bin.sh
echo -e '\\x62\\x69\\x6e\\x66\\x61\\x67\\x74'\r\n

cd /tmp/
busybox wget http://176[.]10[.]250[.]37/.n[redacted]rs/bin2.sh
busybox tftp -r bin2.sh -g 176[.]10[.]250[.]37
sh bin2.sh
echo -e '\\x62\\x69\\x6e\\x66\\x61\\x67\\x74'\r\n
```

Source: *TrendLabs*

Figure 3: BASHLITE infection and ShellShock execution [28]

As the malware is self-replicating, the attacker needs one or more compromised devices to start with. This is achieved by using a Perl Telnet bot to infect the first devices [29].

3.1.2 Attack

The "flooder" part of the malware incorporates four DDoS attacks and a commented (not active) e-mail function. The DDoS attacks are UDP, TCP, Junk and Hold floods. In appendix C we see that the newer variant features some extra control features and one new attack "GETFLOOD", which launches a HTTP GET flood DDoS on the target [30].

3.1.3 More variants

We have discussed two variants of the malware so far (ELF_BASHLITE.A and ELF_BASHLITE.SMB). According to Level3 there are more than a dozen variants [17].

MalwareMustDie evaluated another three variants on their blog [29], which are listed in table 1. The "Qbot" [31] variant has many similarities with the ELF_BASHLITE.SMB version. It is unclear if this is actually the same version, but simply given a different name by Trend Micro than MalwareMustDie. However, in the Qbot source code [31] there does not seem to be a DNS function, so they could be different.

version	particulars
Qbot	A bug fix in the scanning module Aimed at IoT devices [1]
Private	Lacks a scanning module [32]
BLJ [2]	Implements encrypted C&C connections and a persistent client process

Table 1: BASHLITE variants analyzed by MalwareMustDie

There are also versions aiming at web apps (e.g. WordPress) and FTP logins [17].

3.2 Mirai

Mirai may be the most well-known IoT botnet because it was used in high-profile DDoS attacks, showing unprecedented attack volumes [1].

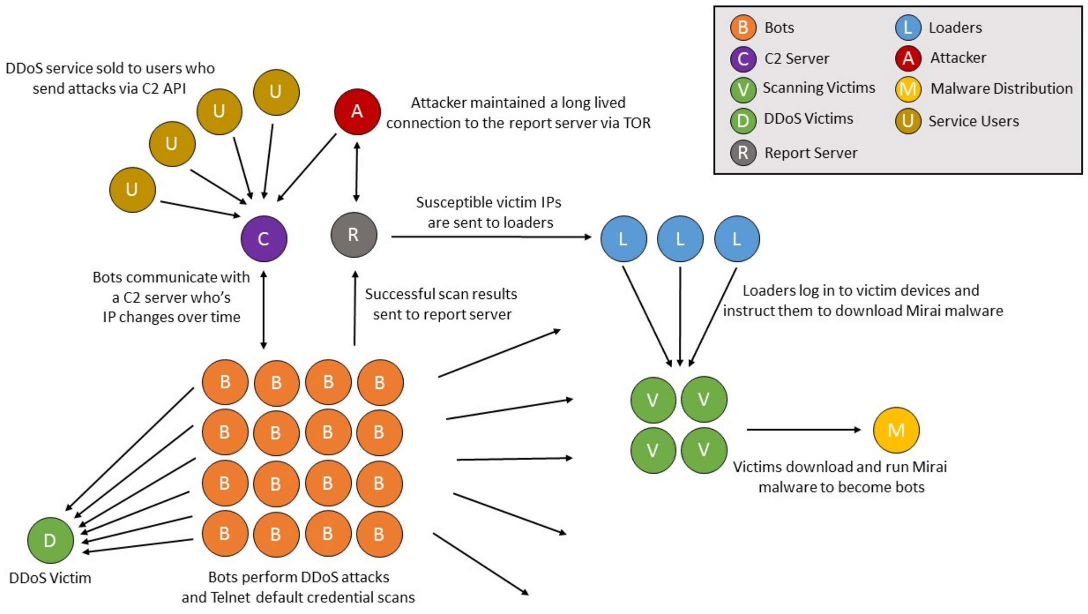
Mirai's source code was released on hackforums and later mirrored on GitHub [10]. This allows both source code analysis and behavioral analysis. We apply these techniques to learn the following behavior characteristics of Mirai: How does it scan for potentially vulnerable devices, how does it attack those devices, once a device has been compromised, how does it infect the device with Mirai and consequently Mirai's behavior once a device has been infected.

The following analysis is based mainly on this leaked source code, our own experiments by compiling and running the code and (non-primary) research published by various security researchers and companies.

Variants of Mirai have been reported with additional functionality not present in this source code [33]. Where possible, this analysis has been updated with information regarding those variants. However, much about the new variants is still unknown, because there have not been any further public source code releases.

3.2.1 Mirai overview

A Mirai botnet consists of four distinct parts: The *C&C server*, one or more *loader(s)*, a *report server* and finally the bots themselves, running on devices infected with the malware. Figure 4 shows an overview.


B Bots
C C2 Server
V Scanning Victims
D DDoS Victims
R Report Server
L Loaders
A Attacker
M Malware Distribution
U Service Users
DDoS service sold to users who send attacks via C2 API
Attacker maintained a long lived connection to the report server via TOR
Susceptible victim IPs are sent to loaders
Bots communicate with a C2 server who's IP changes over time
Successful scan results sent to report server
Loaders log in to victim devices and instruct them to download Mirai malware
Victims download and run Mirai malware to become bots
DDoS Victim
Bots perform DDoS attacks and Telnet default credential scans

Credit: *Level 3 Threat Research Labs*

Figure 4: Mirai overview

[1]Multiple architectures, IoT usernames & passwords.
[2]BadLuckJosh

In the following sections we will discuss the life cycle of Mirai in more detail.

3.2.2 Scanning for devices

Mirai bots include a scanning module that scans for Telnet services on ports 23 and 2323. According to researchers at 360 Netlab there are variants of Mirai that attack ports other than 23 and 2323, such as 23231, 5555, 6789, 7547 and 37777 [34] [33]. This was not present in the code we analyzed. The scanner generates a random IP, checks if it is inside a "bad" range and if it is not inside that range, sends out a SYN packet to that IP. The bad IP ranges are summarized in table 2.

127.0.0.0/8	Loopback
0.0.0.0/8	Invalid address space
3.0.0.0/8	General Electric Company
15.0.0.0/7	Hewlett-Packard Company
56.0.0.0/8	US Postal Service
10.0.0.0/8	Internal network
192.168.0.0/16	Internal network
172.16.0.0/14	Internal network
100.64.0.0/10	IANA NAT reserved
169.254.0.0/16	IANA NAT reserved
198.18.0.0/15	IANA Special use
224.*.*.*+	Multicast

Source: *Adapted from mirai/bot/scanner.c*

Table 2: Mirai IP range exclusions

Due to a peculiarity in the code, the TCP header is set up with a random window size per scanning session. This allows us to differentiate between Mirai and other malware families when analyzing incoming scans. This code can be seen in figure 5.

```
// Set up TCP header
tcph->dest = htons(23);
tcph->source = source_port;
tcph->doff = 5;
tcph->window = rand_next() &
        0xffff;
tcph->syn = TRUE;
```

Source: *mirai/bot/scanner.c*

Figure 5: Mirai TCP window size logic

Another peculiarity is the way which Mirai chooses its destination ports. When scanning 90% of the packets have TCP port 23 as destination, but 10% have port 2323 as their destination. This is governed by the code as shown in figure 6.

```
for (i = 0; i < SCANNER_RAW_PPS; i
  ++)
{
    struct sockaddr_in paddr =
        {0};
    struct iphdr *iph = (struct
        iphdr *)scanner_rawpkt;
    struct tcphdr *tcph = (struct
        tcphdr *)(iph + 1);

    iph->id = rand_next();
    iph->saddr = LOCAL_ADDR;
    iph->daddr = get_random_ip();
    iph->check = 0;
    iph->check = checksum_generic
        ((uint16_t *)iph, sizeof (
        struct iphdr));

    if (i % 10 == 0)
    {
        tcph->dest = htons(2323);
    }
    else
    {
        tcph->dest = htons(23);
    }
```

Source: *mirai/bot/scanner.c*

Figure 6: Mirai destination port logic

3.2.3 Attacking other devices

Once a host running a Telnet service has been found, Mirai tries to log in using a random choice out of a list of known default usernames and passwords[3].

If the login attempt succeeds, Mirai will attempt to run the command "/bin/busybox MIRAI", and checks for the reply of "MIRAI: applet not found". If this is successful, the results are sent to the report server and the next target will be processed.

[3]A full list of usernames and passwords contained in the Mirai source code can be found in appendix D.

3.2.4 Infect

Successful logins from the scanning bots are sent to the C&C infrastructure via the report server. These are then used by the loader to log into the newly discovered device and deliver the malware.

Once the loader has logged in, a similar check is performed, this time using "/bin/busybox ECCHI" as a command and "ECCHI: applet not found" as a response. Figure 7 shows the code snippet where these strings are stored in the loader code. These strings are not obfuscated because this code is not present in the malware itself.

```
#define TOKEN_QUERY
"/bin/busybox ECCHI"
#define TOKEN_RESPONSE
"ECCHI: applet not found"
```

Source: *loader/src/headers/includes.h*

Figure 7: Token query and response

After verifying the login, the loader checks the processor architecture of the target. If known the loader serves a binary built for the target. This is done via one of three methods: wget, tftp or direct file transfer using echo commands into a file over Telnet.

Mirai supports a large number of processor architectures[4]. This may be indicative of the malware author's intent to target as many devices as possible.

After delivering the payload, the loader runs the malware, waits for the malware to output the check string "listening tun0" and disconnects.

3.2.5 Infection characteristics

After successful execution of the malware, Mirai removes its own binary and continues to run only in memory. This could be an attempt to evade detection or thwart analysis, but it also means that a Mirai infection is not persistent. Once a device has been rebooted the malware will no longer be present.

Changes to device

Mirai attempts to kill any process bound to ports 22, 23 and 80 and rebinds these ports to itself. This prevents other attackers from (re)gaining access to the device. Mirai also binds to port 48101, however only on 127.0.0.1, meaning that this will not be detectable from the outside.

A rather aggressive approach is taken to remove other malware from the system that it knows about. It does this by grabbing the path of all executables from all running processes, scanning the binaries for certain known strings and when it finds one of the strings listed in table 3, it kills the associated process.

Variable name	**Decoded string**
m_qbot_report	REPORT %s:%s
m_qbot_http	HTTPFLOOD
m_qbot_dup	LOLNOGTFO
m_upx_str	\x58\x4D\x4E\x4E\x43\x50\x46\x22
m_zollard	zollard

Source: *mirai/bot/killer.c and table.c*

Table 3: Strings the killer searches for

Note that m_upx_str seems to be double-encoded with the same XOR key. Once decoded again, it becomes "zollard". This seems to be a mistake on the malware author's part.

As can be seen in figure 8, Mirai also searches for binaries with ".anime" in their path. When found the process gets killed and the binary deleted. Anime is a competing IoT malware family [35].

```
// If path contains ".anime" kill.
if (util_stristr(realpath, rp_len
    - 1,
     table_retrieve_val(
         TABLE_KILLER_ANIME, NULL))
         != -1)
{
    unlink(realpath);
    kill(pid, 9);
}
```

Source: *mirai/bot/killer.c*

Figure 8: Anime killer

The killer process keeps running while the

[4]A full list of architectures supported by Mirai can be found in appendix E.

malware runs, continually killing the competitors it finds.

C&C traffic

Certain C&C traffic happens in an early stage before attack commands are issued.

Mirai uses three C&C addresses that are hard coded in the source:

- CNC
- SCAN_CB
- FAKE_CNC

The CNC is the C&C domain used for getting commands for DDoS attacks etc. The SCAN_CB is the scan callback or "report" server used to report scanning results. Both of these are stored in obfuscated form. The FAKE_CNC is a hard-coded IP-address that the malware performs a reverse DNS lookup on, but it is not actually part of the C&C infrastructure. According to the malware's author this is intended to fool researchers, law enforcement and competing botnet operators [36]. Figure 9 shows the location of some of these constants in the code. A full list of decoded string constants is included in appendix F.

Both the FAKE_CNC lookup and the CB lookup happen during the spreading phase of the malware's life cycle. This could be used to detect Mirai infections in an early stage.

```
#define FAKE_CNC_ADDR
INET_ADDR(65,222,202,53)
#define FAKE_CNC_PORT    80
```

Source: *mirai/bot/includes.h*

Figure 9: Fake C&C domain and port

According to researchers at 360 Netlab, newer Mirai variants exist that use a Domain Generating Algorithm (DGA) to communicate with their C&C infrastructure [37]. This means that the C&C addresses are no longer present in the binary, but are generated dynamically instead.

Spreading

Once an infection has taken place, it exhibits the same scanning behavior as described in section 3.2.2. However, Mirai excludes local[5], link-local[6] and a number of other networks from its scanning. As such a Mirai infection does not attempt to scan inside a network if the network uses RFC1918 [38] addressing internally.

If a network uses a different addressing scheme which is not explicitly excluded by Mirai, it is possible for Mirai to attempt to infect additional devices in the network. However, these devices would also need to have internet connectivity in order to join the botnet because the mechanism for infection relies on the loader server to deliver the malware, which delivers its payload from the internet. This means that there is no additional risk for a device to become infected if it isn't already exposed to the internet.

Mirai does not seem to be designed to gain a foothold inside a network and then spread further inside that network. In fact, quite the opposite seems to be the case as evidenced by the exclusion of internal addresses when scanning.

DDoS attack

There are 9 known attack types implemented in Mirai, these are listed in table 4.

Attack	**Description**
UDP	UDP flood
VSE	Valve Source Engine query flood
DNS water torture	Recursive DNS query attack
SYN	SYN packet flood
ACK	ACK packet flood
STOMP	ACK flood with STOMP
GRE IP	GRE flood
GRE Ethernet	Ethernet encapsulated inside GRE flood
Plain UDP	UDP flood optimized for speed
HTTP	HTTP layer 7 flood

Table 4: Mirai DDoS attack types

[5]10.0.0.0/8, 172.16.0.0/12 and 192.168.0.0/16, see RFC1918 [38]
[6]169.254.0.0/16, see RFC3927 [39]

Because this research focuses on the early stages of IoT the details of these attacks were not included in our research. However, it is worth mentioning that the GRE attack had not previously been seen [1].

3.3 Common properties of IoT malware

In our analysis of Mirai and BASHLITE we have shown many similarities like the different stages in their life cycle and their actions within those stages.

However, none of these characteristics are new or exclusive to IoT malware. Scanning, dictionary attacks and vulnerable devices with exposed Telnet interfaces are all well-known problems that have plagued the security of the internet for decades. What makes IoT malware unique is the type of devices it targets. By targeting embedded Linux-based devices of many different processor architectures using simple methods like weak credentials or known vulnerabilities, IoT malware can be very successful.

Based on the analysis of the BASHLITE and Mirai malware, we have identified four separate stages in the life cycle of IoT malware: *Scan, Attack, Infect* and *Abuse*. In the paragraphs below the common properties of IoT malware are described per life cycle stage.

3.3.1 Scanning

One common behavior seen in all types of IoT malware we have researched is scanning for potentially vulnerable hosts. This is usually implemented by a scanning engine that scans random IPv4 addresses or subnets. The ports targeted vary, but mostly focused on ports known to be running a Telnet daemon, commonly 23 and less commonly 2323 [40]. In the case of specific exploits being used, ports are scanned that run the vulnerable service [33] [34] [41].

3.3.2 Attacking vulnerable hosts

The most common attack method is a Telnet dictionary attack using known usernames and passwords. These credentials can be generic such as "root/root" or "admin/admin", but also specific passwords such as "7ujMko0admin", "xc3511", "juantech" which are specific to certain hardware vendors. Some malware variants will use known exploits to gain entry, such as the TR-064 and TR-069 exploits that plagued routers last year [41]. These remain relatively specific and are not applicable to IoT devices in general. One commonality across all IoT malware variants we investigate is that they target embedded devices running Linux and busybox.

3.3.3 Infection

After a vulnerable device has been found, the malware will attempt to infect the device. This is done by downloading a compiled C binary of the malware itself. There are several ways this could be done:

- echo over Telnet
- TFTP
- HTTP download (wget)
- DHT/uTP[7] [43]

Whatever delivery method it uses, the malware will be available for many different processor architectures, ensuring the maximum possible amount of successful infections. This targeting of many processor architectures is a common factor in IoT malware.

Some malware is highly territorial and will attempt to remove other malware it encounters on the device. It will also close and rebind certain ports to make it harder for competing malware to attack the device.

3.3.4 Abuse

Currently most IoT botnets are used to mount DDoS attacks, mostly the typical attacks such as SYN or SYN/ACK floods, HTTP attacks, UDP floods etc. As we have seen, very powerful DDoS attacks can be performed with a large IoT botnet using only direct traffic. Traditional large DDoS attacks were mounted using spoofed traffic and some kind of protocol that provides reflection/amplification (DNS, NTP, SSDP, chargen etc.) Not so with IoT botnet attacks, this makes IoT botnets unique: sheer volume of traffic.

There have not been any recorded applications

[7]Distributed Hash Table (DHT) and Micro Transport Protocol (uTP), a BitTorrent-like protocol [42]

for IoT botnets other than DDoS attacks, although they could be used for almost anything. One special case bears mentioning: Hajime has not exhibited any malicious behavior as of yet, it is unclear what the purpose of this botnet is [43].

4 Detection techniques

In this section we evaluate generic detection techniques likely available to network operators. We specifically do not focus on techniques that require installation on end user devices, because network operators (e.g. ISPs) typically do not have access to them.

4.1 NetFlow

NetFlow, a network protocol originally developed by Cisco [44], can be used to collect IP traffic metrics on a router/switch. NetFlow has become a generic industry-wide term for traffic analysis. Nowadays, many hardware vendors support NetFlow [45] and/or similar technologies (e.g. sFlow, jFlow and NetStream).

NetFlow is able to characterize traffic flows up to (and including) layer 4 [46]. Network administrators use NetFlow to understand traffic patterns and monitor network bandwidth utilization. It is also used to locate network abusers and for several other things [47].

As previously stated, NetFlow characterizes traffic flows, which are then aggregated into entries in NetFlow packets. A flow is a unidirectional sequence of packets that share common properties. This in such a way that the sequence of packets are related. The exact definition of a flow in the context of NetFlow is defined in RFC3954 and 6437 [48] [49].

The "common properties" mentioned in the RFCs depend on the specific flow export protocol and version. The properties include at least (in this case NetFlow version 5) [46]:

- Source interface
- Source & destination IP address
- Layer 4 protocol
- Source & destination port number
- Type of service value (DSCP)

As soon as one of those variables changes, a new flow is created [50].

Many network administrators use "Random Sampled NetFlow" [51] nowadays. This randomly selects packets passing the monitored interface(s) in an interval of n packets. A typical sample ratio would be 1 in 1000 [52], where roughly every 1000th packet will be analyzed. Because of this, NetFlow is less suitable for detecting short communication streams, as they are easily overlooked when using sampling.

Network administrators can analyze NetFlow data with a flow analysis application (like Nfsen and Ntop). In the most used NetFlow versions (5 and 9), only limited data is available (e.g. window size is not included).

4.2 Packet analysis

Packet analysis (also known as packet-sniffing) describes the process of capturing and interpreting data that flows through a network.

Well known packet capture tools are "tcpdump" and "Wireshark" [53]. They allow the user to choose one or multiple network interfaces. The interface is then put in promiscuous mode, which passes all the receiving frames to the CPU instead of only the frames addressed to it [54]. This is particularly useful when using port mirroring, because as nearly none of the frames are addressed to the watched interface, they would have been dropped before reaching the capture tool.

In comparison with NetFlow, more data (e.g. window-size) from the header is available for analysis. Besides that, NetFlow data is only based on header information, and not the payload. With packet capture tools, it is possible to inspect the payload as well.

Most of the tools can be used to analyze data in real-time or from previously captured data (pcap files). The data is first dissected, which means its part of the protocol is decoded and the data structured according to the specifications in the corresponding RFC [55]. Then, most often the data is filtered using a combination of zero or more selectors specified by the user to enable the user to only select the information they need.

Use for darknet monitoring

When monitoring a darknet[8], incoming packets could be represented in any number of ways. However, to fully utilize the available information, a full packet capture is most useful. This allows a researcher or IT professional to analyze all the properties of the incoming packets, as other representations of packet information, such as NetFlow, are incomplete.

4.3 Honeypots

Honeypots are systems designed to entice attackers to attempt to gain access to these systems [56]. To achieve this, the systems simulate some kind of vulnerability (e.g. weak password or software vulnerability) and they are usually designed to look important as to present an alluring target. The main goal of a honeypot is to analyze the techniques used by the attacker [57].

Honeypots are often classified by their interaction level, which is scaled from low (no interaction, just logging), medium (some interaction, but no arbitrary code executed) to full interaction, where the honeypot is basically a jailed environment capable of offering almost the same features as the real device. The security risk grows with the interaction level [58].

In our experiments, we create multiple honeypots with the Cowrie software [59]. Cowrie is a medium-interaction SSH & Telnet honeypot, written in Python. It is based on the Kippo honeypot but offers some extra features.

By analyzing honeypot logs (and collected files), it is possible to gather a lot of valuable information about botnets/malware. Depending on the malware behavior and the interaction level of the honeypot, it is possible to gather information such as:

- new malware families,
- variants in malware families,
- malware code,
- targeted type of devices (e.g. based on username:password combinations),
- address information (IP addresses / host names / port numbers) of command & control servers, attackers, targets, etc.

4.4 DNS analysis

DNS [60] is commonly used by malware to lookup the IP address of the C&C server. It is often used instead of a hard-coded IP address to offer the hacker the ability to change C&C servers when one is taken offline. Another use case is when the hacker wants to use fast-flux DNS to hide the C&C servers' real IP address. This technique abuses the load balancing system in the DNS by adding & removing multiple IP addresses behind a single domain name very often (e.g. every 5 minutes) [61].

Companies and institutions that operate large, internet-scale DNS servers, such as for example Google and OpenDNS [62], could monitor the amount of queries for specific domain names. Domain names that are used by C&C servers often show a large peak in the amount of lookups in a relatively short time frame.

Some malware uses the DNS as the main communication protocol between infected devices and the C&C server to avoid detection [63]. The messages are hidden in the the DNS questions and answers.

Blocking malicious DNS servers can be difficult due to the design of the DNS protocol. The DNS request is forwarded by recursive name servers until the authoritative DNS server is reached [63]. The client does not communicate directly with the authoritative server in most cases.

Some malware uses DGA, which enables the attacker to switch domain names often (e.g. every day) [64]. This will prevent the attacker from losing control over the botnet when the domain name is taken offline / sinkholed [9].

[8]Darknet being defined as a public network segment without any systems connected.

[9]Sinkholing is when a domain is taken over by researchers or law enforcement but kept operational to gather information about the infected devices connecting to the domain.

By analyzing DNS queries it is possible to classify suspicious requests. For example requests with very large request / answer sections might include communication between malware and a C&C server [63]. The IP addresses and domain names involved in the DNS traffic can then be used for further investigation. An advantage of this type of analysis is that it does not matter if the communication is encrypted or not [61].

4.5 Application of techniques

We have described several commonly available techniques which could be used for malware detection on an ISP-level and we have analyzed leading IoT malware. In this section we will evaluate each detection technique with respects to detecting IoT malware activity.

4.5.1 NetFlow

NetFlow is a very useful tool for detecting activity in certain phases of the malware life cycle, depending on certain factors. Certain limitations exist that limit its usefulness however.

Flow sampling
NetFlow is often configured using a flow sampling of one in N packets, common values for N are 100, 500, 1000, etc. This means that the application of NetFlow for detecting IoT malware has limitations. Given that a single infected device will only send a single TCP SYN packet per scanning session, the chance of capturing this single packet is low, namely the same as the NetFlow sampling rate, such as 1 in 100. This means that in a typical configuration NetFlow is not very useful for detecting incoming scans from the internet.

TCP header fields captured
NetFlow only captures a subset of the header data and no packet contents. One property of Mirai's scanning behavior in particular, as discussed in section 3.2.2, is that a random TCP window size is set each scanning session. Typical NetFlow configurations do not capture this information. Therefore, it would not be possible to distinguish between Mirai and other families of IoT malware.

Use in forensic analysis
One of the benefits of NetFlow is that it is much more storage-efficient than full packet captures. Storing historic data is therefore far more viable compared to other solutions. This means that historic data is often available for some time and can be analyzed when information about new threats becomes available. When faced with a compromised device inside one's own network, NetFlow can be useful to capture scanning behavior because even though only a subset of outgoing packets is captured, the scanning behavior of IoT malware is such that many packets will be generated and thus captured by the NetFlow sampling. If the scanning behavior is consistent, certain patterns will appear in the statistical sample that can be compared with scanning behavior of known malware. We demonstrate such a scenario in our experiments.

4.5.2 Packet capture

A full packet capture of traffic has the most detailed information available from the mentioned detection techniques. Header data that is missing in NetFlow (e.g. window size) is available, but also inspection of packet payload is possible. This means that all network-based properties of IoT malware can be detected and analyzed. However, this comes at a certain cost. For large networks with high amounts of bandwidth, many resources are needed to inspect (and store) all packets that go through it. Care should also be taken to avoid privacy concerns associated with deep packet inspection and long term storage of packet contents. This makes packet capture especially useful for live (or short time) captures of packets to/from specific hosts, but due to the aforementioned concerns often no historical data is available.

Darknet
When capturing packets from a darknet, full packet capture is a possibility, since the volumes are much lower and the privacy concerns are alleviated due to the fact that no legitimate traffic is present on a darknet. Darknet traffic is especially useful for detecting scanning behavior from all kinds of botnets, not in the least of which IoT botnets. In our experiments we will show that a darknet is a useful tool in analyzing the scanning behavior of IoT malware.

4.5.3 Honeypots

In this section we discuss the applicability of honeypots for detecting IoT malware.

Use for IoT malware
Because IoT malware mainly targets Linux-based devices over telnet, a generic linux telnet honeypot can be used to gather information about IoT malware without much work in adapting the honeypot specifically for IoT use. Depending on the type of honeypot (low, medium or full interaction) they can be used to capture data about all phases of the malware life cycle.

Tracking variants
Honeypots are an excellent source of information about new malware families and new variants in existing families. This is because they log commands issued and binary files downloaded, allowing analysis to be done on the behavior of the malware. In our experiments we show an example of how the "busybox strings" can be used to differentiate between certain IoT malware families.

Gathering IP addresses
Collecting IP addresses of compromised devices is also possible with honeypots, because they log IP information about any attack logged against the honeypot. This IP information can then be used to inspect network traffic for malicious activity from these hosts or to notify ISPs and system owners about malicious activity in their networks. This information is more detailed than from capturing scanning behavior alone.

Updating analysis
Combining the aforementioned information can lead to new insights into IoT malware activity. This can be used to further tweak the detection methods to more effectively detect IoT malware.

4.6 DNS analysis

DNS analysis has a stronger focus on detecting C&C communication. C&C communication often happens after the infection has already taken place i.e in the abuse phase. As a result, DNS analysis is not useful for detecting most intrusion attempts, only established malware. It also does not work when malware communicates to IP addresses directly.

5 Experiments

In our experiments we apply the characteristics from the malware analysis to the discussed detection techniques.

5.1 Mirai scanning behavior

In this section we discuss our experiments that focus on the scanning behavior of Mirai.

5.1.1 TCP window size

Our analysis of Mirai shows that Mirai uses a randomized window size in it's scanning behavior. This is discussed in section 3.2.2. The TCP window size is a 16-bit field in the TCP header. This means that the window size value in the TCP header has an upper bound of 65335 [65]. A window scale option can be used to extend this, but Mirai does not use this so it has not been taken into consideration.

The goal of this experiment is to compare the window size values generated in Mirai's scanning module with real world TCP window sizes. To achieve this we take two steps: First, we take six 5-minute samples of all incoming SYN packets on a /15 darknet (131072 IP addresses) and measure the TCP window sizes of those packets. This is to get a sample of all different window sizes one might expect to see on a darknet. We do the same but only for TCP SYN packets destined for port 23. This port is selected because it is a port that shows the most IoT scanning activity, as well as non IoT malware scanning activity.

The 5-minute samples are combined and the frequency of occurrence of each window size value is counted. In this way over 5 million TCP SYN packets were captured, of which over 3 million were destined for TCP port 23.
Figure 11 and 12 show a top 10 tally of the results of these measurements.

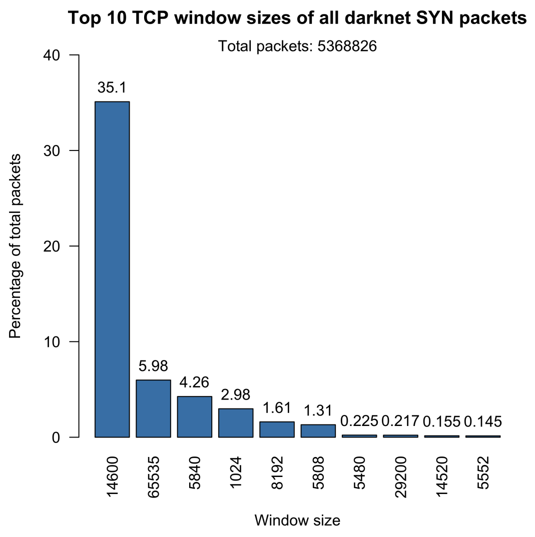


Figure 11: Darknet window size graph

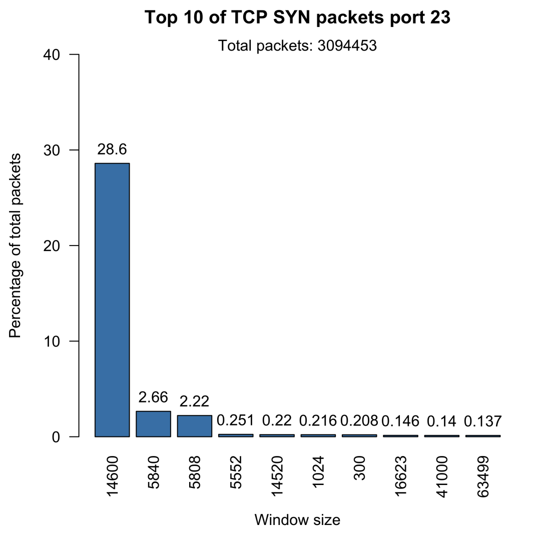


Figure 12: Darknet port 23 window size graph

These figures show that in real-world darknet data, a clear preference is seen for a few popular window sizes, meaning that the distribution of window sizes is not uniform. This is true for the packets destined for port 23, as well as the packets destined for other ports. A difference is seen between which window sizes are popular in the port 23 data and the other data, but in both cases it can clearly be seen that the distributions of window sizes are not uniform. This means that when capturing packets from a darknet, we expect to see a non-uniform distribution of window sizes.

Next, we attempt to determine if Mirai's window sizes when scanning are distributed uniformly across the entire 0-65535 region. The goal of this was not to determine the relative quality of Mirai's PRNG, but to create a sample to compare the real-world data from the darknet against. For this we take the code that generates the window sizes and create a standalone implementation to generate an arbitrary number of TCP window sizes. This simulates Mirai's scanning behavior without having to rely on capturing actual Mirai scans. The modified source code is included in appendix G.

Result

Figure 13 shows a comparison of the the real-world data of TCP SYN packets destined for port 23 on the darknet, compared with the generated window sizes from Mirai. The graph plots the number (count) of occurrences (in log scale) of each window size in the overall darknet (blue) and Mirai (red) samples.

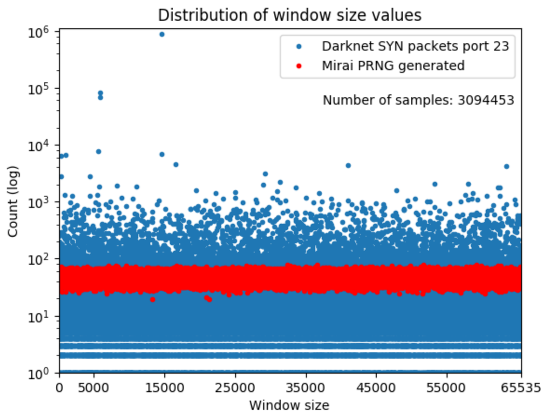


Figure 13: Darknet v.s. Mirai comparison

The comparison shows that Mirai's TCP window sizes are indeed distributed very evenly over the possible range of 0-65535. It also shows that this is not at all the case for real-world TCP SYN packets that are captured by the darknet.

This means that any SYN packet with an uncommon window size, destined for a port that

Mirai scans on, can at least be suspected to be a scanning attempt from a Mirai bot.

5.1.2 Distribution of destination ports

This variant of Mirai has a very specific distribution of destination ports of 90% port 23, and 10% port 2323 when scanning. We verify this by modifying Mirai's code to only scan a single IP under our control, and to not take any further action. The code that determines the destination port is left intact. In this manner we capture a sample of 1300 destination ports. The packet capture for scanning sessions using this modified code shows a clear 90/10 distribution as expected. This means that it is possible to detect Mirai's outgoing scanning behavior from NetFlow data or other sources such as packet captures or firewall logs.

To verify the effectiveness of this detection technique we analyze the historical NetFlow data of a device suspected to be infected with Mirai. This data is available to us because the ISP was previously notified about a possible Mirai infection inside their network. To mitigate the risk, the ISP blocked outgoing traffic for the device's IP at the network's edge. However, the malware was left active on the device and still able to attempt outgoing scans. These scanning attempts are picked up by the NetFlow collector on the router before the traffic is dropped. This means that only outgoing traffic data is available, but it is still a very valuable sample of data to test against.

This sample consists of over 58 hours of NetFlow data using 1 in 100 sampling. In total 14597 packets out of 1459700, giving a margin of error of 0.48%. Figure 14 and table 5 show the result of this analysis (as reported by nfdump). It is clear that the distribution of destination ports matches the expected value closely.

Port	No. of packets	Percentage
23	1304100	89.3
2323	155600	10.7

Table 5: Aggregate NetFlow data

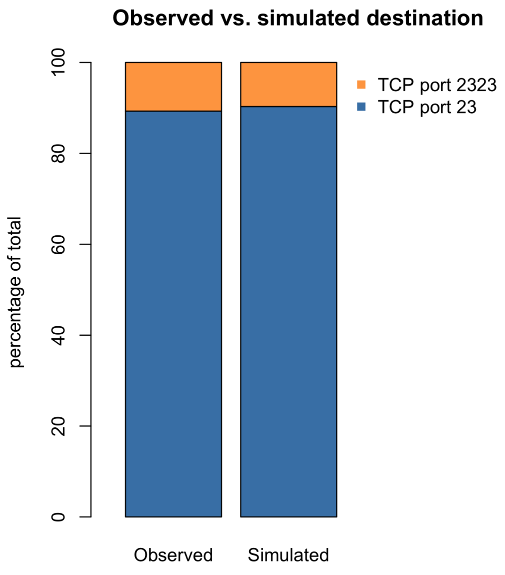


Figure 14: NetFlow data comparison

5.2 Honeypot data

We set up two Telnet honeypots using Cowrie[10] on two separate public IP addresses. The honeypots listen to port 23 and accept the username/-password combinations that were included with the Mirai and BASHLITE source code. The goal of this experiment is to see if a generic Telnet honeypot could be adapted to track variants of IoT malware by capturing the "busybox" check strings. Over a period of a week we capture many reconnaissance and infection attempts. To determine number of unique variants we count the number of unique "busybox" check strings per IP. This results in the data shown in table 6.

In our malware analysis we only find two strings: MIRAI and ECCHI. As can be seen from our honeypot data, even though these two strings are common, there are many more strings that we have not seen in our analysis such as MASUTA and OBJPRN. These malware variants could only have gained access to our honeypots by using the same default credentials used by Mirai. However, not only Mirai has gained access to our honeypots. In all we find 8 new

[10] https://github.com/micheloosterhof/cowrie

busybox strings, and we find that there are one or more variants that use a random 5-character string every time.
This shows that a generic Telnet honeypot can be used to learn about new IoT malware variants.

MIRAI	3147
MASUTA	1835
MM	309
OBJPRN	215
MEMES	29
THTC	18
ECCHI	18
TERROR	5
LLDAN	2
TASKF	2
FBI	2
Subtotal	**5582**
5 random characters	7624
Total	**13224**

Table 6: Unique IP/string combinations seen

6 Discussion & Conclusion

In this section we discuss the results of our research, its limitations and identify possible avenues of future research.

6.1 Limitations

We have identified some limitations in our research. The sample size of the darknet data might be considered low. However, due to the random nature of IoT botnet scanning behavior that we observe, there should not be any difference in scanning activity over time. Furthermore, smaller samples keep the file size of the packet captures more practical. Another limitation is restricting to Telnet-based services. Certain variants of Mirai also target other ports. However, port 23 is associated with all known families of IoT malware. Also, according to our darknet data, and by Internet Storm Center's port report [66], port 23 is still the most popular target out of the ports known to be targeted by IoT malware. One last limitation is a lack of results showing IPv6 malware activity. Now or in the future there may be many IoT devices directly connected to the internet via IPv6. However, IoT malware relies on IP scanning. Currently, scanning the entire IPv4 address space is quite fast and can be done in about two hours [67]. Scanning the IPv6 address space using the traditional brute-force means, employed by IoT malware, is not considered feasible due to a greatly increased search space of 128 bits as opposed to IPv4's 32 bits [68].

6.2 Research questions concluded

Internet of Things (IoT) malware is prevalent and a major source of DDoS traffic [1]. Currently, most IoT device vendors do not provide a mechanism to update devices in the field automatically. Because of this, and the increase in IoT devices connected to the internet, this problem will likely continue to grow [69]. To combat this threat and mitigate the associated risks, feasible techniques for detecting IoT malware activity on an ISP network are needed.

To achieve this goal, common properties of IoT malware have to be found to be able to detect this activity. This is the subject of research sub question A. In section 3 we analyze leading IoT malware and extract common properties of IoT malware. We summarize and describe these properties in a life cycle model in section 3.3. Common properties include the targeting of internet-connected embedded Linux devices such as DVRs, IP cameras, etc. Other commonalities lie in the methods used in attacking and infecting other devices. Different methods of spreading exist, but the main method is by using default credentials via telnet.

Using these properties to detect IoT malware is the subject of sub question B. In section 4.5 we show how techniques available to ISPs such as NetFlow and packet analysis can be used to detect IoT malware activity in their network. Our experiments with TCP window size analysis and dark web scanning behavior as outlined in section 5.1 show that these techniques are indeed effective.

Often malware source code is not available for analysis. This means that gathering properties of new IoT malware requires a different technique than malware analysis. Sub question C is about using generic honeypots to gather insight into the development of IoT malware without having new source code available. In section 4.3 we show what types of properties can be gathered using honeypots. In the experiment outlined in section 5.2 we apply this technique and find new busybox check strings. These strings are different that the strings we find in our analysis of the Mirai malware. This shows that a generic Telnet-based honeypot can be used to gather properties of new IoT malware.

Our analysis of two leading IoT malware families resulted in properties that can be used to detect their activity on a network level. Their scanning behavior is such that outgoing scans can be detected using NetFlow, and incoming scans can be detected using a suitably-sized darknet. Both these techniques are available to ISPs. This means that leading IoT malware be detected using network-based techniques available to ISPs.

The detection of IoT malware depends on the known properties of the malware. Many similarities exist between IoT malware families. However, there are also differences, such as using known vulnerabilities instead of Telnet brute forcing. In order to combat IoT malware a clear picture of these differences and commonalities is needed. This means methods of gathering insight into IoT malware and how it evolves over time are needed. Techniques such as honeypots and malware analysis, possibly supported by darknet analysis, are crucial to gather this information and complete the picture.

In summary: Network-based detection techniques can be effective in detecting IoT malware activity on an ISP level, when applied with knowledge of malware gained from sources such as honeypots and malware analysis.

6.3 Future work

We have identified several avenues for future research on this topic.

- In this research project we analyzed two malware families: BASHLITE and Mirai. To improve the reliability of our life cycle model for IoT malware detection, more malware families would need to be analyzed. Families we would suggest to analyze are Hajime, NyaDrop and LuaBot.

- Because the short release cycle of malware variants, constant analysis work is necessary to be able to detect the latest variants. Future research could look into automated updates of the model, by automatically parsing honeypot logs and automatically analyzing new malware samples in a sand-boxed environment.

- In our research we focused on techniques that are commonly available to ISPs. Because our research of the two malware families did not result in many DNS-based characteristics, we limited ourselves to only mentioning the Mirai domain generation algorithm. However, other malware families might use DNS for other means such as communication with their C&C server, fast-flux to hide addresses etc. This would offer plenty of new possibilities to detect malware.

- Researching out-of-the-box detection techniques is also possible. An example of this is the CAMELIA project, where researchers try to detect infected devices by analyzing changes in electromagnetic emissions. In this category, we have thought about comparing network latency of IoT devices in normal operation and after infection.

- Creating a baseline for open/closed ports for every IoT device and then scanning for devices that have differences to the baseline could be a reliable way to detect infections. This could be done both inside one's network, or on the internet using services such as Shodan.

- In our research we do not see any IPv6 activity, but there may already be IPv6 IoT malware active that we haven't seen. New techniques have been developed that use certain assumptions and technical properties of IPv6-related protocols. These techniques can be used to greatly reduce the search space of IPv6, possibly making IPv6 scanning viable [70]. For IoT malware these techniques could, for example, be used to scan for specific manufacturers by limiting scans to addresses generated by the EUI-64 protocol that includes the Organizationally Unique Identifier (OUI) corresponding to the targeted manufacturer [71]. Although no IoT malware that uses these techniques is known, once IPv6-connected IoT devices become numerous enough, we might start seeing malware applying these techniques. IPv6-only honeypots that have IP addresses generated in the range of EUI-64 addresses for known IoT device manufacturers might be a viable approach to gather data.

- More darknet data could be gathered to follow trends in scanning behavior from IoT botnets, and this could be correlated with data on which services are associated with those ports and possible security vulnerabilities in those services.
- Gathering the TCP window sizes and other TCP header values of all incoming SYN packets on a darknet and grouping them by source IP could be used to determine the distribution of window sizes generated by a single IP/bot. This allows for statistical analysis of these values and might be used to differentiate malware families by scanning behavior more reliably.

7 Acknowledgments

The authors would like to thank the following people for contributing to our research project:

- Rogier Spoor, research supervisor, for his guidance during our research project,
- Xander Jansen for his help and enthusiasm with applying our technique to real-world NetFlow data,
- Roland van Rijswijk-Deij for his help with our darknet experiments,
- Wim Biemolt for sharing his expertise in network monitoring.

We would also like to extend our thanks to all the other fine people at SURF, who hosted us during our research.

References

[1] Brian Krebs. KrebsOnSecurity hit with record DDoS. `https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos`.

[2] OVH. The DDoS that didn't break the camel's VAC. `https://www.ovh.com/us/news/articles/a2367.the-ddos-that-didnt-break-the-camels-vac`.

[3] Akamai. State of the internet Q3. Technical report, Akamai, 2016. `https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q3-2016-state-of-the-internet-security-report.pdf`.

[4] Matthew Prince. The DDoS that almost broke the internet. `https://blog.cloudflare.com/the-ddos-that-almost-broke-the-internet`.

[5] Matthew Prince. Technical details behind a 400Gbps NTP amplification DDoS attack. `https://blog.cloudflare.com/technical-details-behind-a-400gbps-ntp-amplification-ddos-attack`.

[6] D. Senie P. Ferguson. BCP-38. `https://tools.ietf.org/html/bcp38`.

[7] P. Savola F. Baker. BCP-84. `https://tools.ietf.org/html/bcp84`.

[8] Internet Society. Mutually agreed norms for routing security (MANRS). `https://www.routingmanifesto.org/`.

[9] Brian Krebs. Source code for IoT botnet Mirai released. `https://krebsonsecurity.com/2016/10/source-code-for-iot-botnet-mirai-released`.

[10] Anna-Senpai. Mirai source code. `https://github.com/jgamblin/Mirai-Source-Code`.

[11] Kelly Jackson Higgins. Root & the new age of IoT-based DDoS attacks. `http://www.darkreading.com/vulnerabilities---threats/root-and-the-new-age-of-iot-based-ddos-attacks-/d/d-id/1327281`.

[12] Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. Iotpot: Analysing the rise of iot compromises. In *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, Washington, D.C., 2015. USENIX Association.

[13] Kuan-Cheng Lin, Sih-Yang Chen, and Jason C. Hung. Botnet detection using support vector machines with artificial fish swarm algorithm. *Journal of Applied Mathematics*, 2014.

[14] Frank Konkel for Nextgov. DARPA funds IOT malware detection project. http://www.nextgov.com/cybersecurity/2016/08/darpa-funds-iot-malware-detection-project/130460/.

[15] Pierluigi Paganini for Security Affairs. MITRE will award $50,000 for a solution that detects rogue IoT devices. http://securityaffairs.co/wordpress/52115/iot/mitre-iot-devices.html.

[16] Wikipedia. BASHLITE. https://en.wikipedia.org/wiki/BASHLITE.

[17] Level 3 Threat Research Labs. Attack of things! http://blog.level3.com/security/attack-of-things/.

[18] MalwareMustDie. CoderID: SINDEN. http://x.malwaremustdie.org/stat/sinden.html.

[19] Brian Krebs. Who is anna-senpai, the mirai worm author? https://krebsonsecurity.com/tag/lelddos/.

[20] Brian Prince. B́ashlitem̀alware leverages ShellShock in BusyBox attack. http://www.darkreading.com/attacks-breaches/bashlite-malware-leverages-shellshock-in-busybox-attack/d/d-id/1317508.

[21] M. Cotton and L. Vegoda. Rfc 5735 - special use ipv4 addresses. https://tools.ietf.org/html/rfc5735.

[22] Anthony Tellez. An archive of BASHLITE source code. https://github.com/anthonygtellez/BASHLITE.

[23] Eduard Kovacs. BASHLITE malware uses ShellShock to hijack devices running BusyBox. http://www.securityweek.com/bashlite-malware-uses-shellshock-hijack-devices-running-busybox.

[24] Trend Micro Incorporated. ELF_BASHLITE.A. http://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/elf_bashlite.a.

[25] Pierluigi Paganini. A new BASHLITE variant infects devices running BusyBox. http://securityaffairs.co/wordpress/30225/cyber-crime/bashlite-exploits-shellshock.html.

[26] Trend Micro Incorporated. ELF_BASHLITE.SMB. http://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/ELF_BASHLITE.SMB.

[27] Wikipedia. Shellshock (software bug). https://en.wikipedia.org/wiki/Shellshock_(software_bug).

[28] Rhena Inocencio. BASHLITE affects devices running on BusyBox. http://blog.trendmicro.com/trendlabs-security-intelligence/bashlite-affects-devices-running-on-busybox/.

[29] MalwareMustDie. MMD-0052-2016 - overview of SkidDDoS elf++ IRC botnet. http://blog.malwaremustdie.org/2016/02/mmd-0052-2016-skidddos-elf-distribution.html.

[30] Dietmar Kenzle. Seite 2 - aufgepasst: berwachungskameras mutieren zu Botnets. http://www.it-administrator.de/themen/sicherheit/fachartikel/204048.html.

[31] Jeffery Wilkins. QBOT-ART-OF-WAR. https://github.com/CanadianJeff/QBOT-ART-OF-WAR/.

[32] Kernelmode.info unixfreaxjp. Malware collection. http://www.kernelmode.info/forum/viewtopic.php?f=16&t=3966&p=28461#p28461.

[33] 360 Netlab. A few observations of the new mirai variant on port 7547. http://blog.netlab.360.com/a-few-observations-of-the-new-mirai-variant-on-port-7547/.

[34] 360 Netlab. Mirai scanner. http://data.netlab.360.com/mirai-scanner.

[35] Evosec. New IoT Malware? Anime/Kami. https://evosec.eu/nl/new-iot-malware/.

[36] Anna-Senpai. Mirai source code release accompanying forums post. https://github.com/jgamblin/Mirai-Source-Code/blob/master/ForumPost.md.

[37] 360 Netlab. Now Mirai has DGA feature built in. http://blog.netlab.360.com/new-mirai-variant-with-dga.

[38] Daniel Karrenberg, Yakov Rekhter, Eliot Lear, and Geert Jan de Groot. Address Allocation for Private Internets. RFC 1918, February 1996.

[39] Dr. Stuart D. Cheshire, Dr. Bernard D. Aboba Ph.D., and Erik Guttman. Dynamic Configuration of IPv4 Link-Local Addresses. RFC 3927, May 2005.

[40] Rick Wanner. What is happening on 2323/tcp? https://isc.sans.edu/forums/diary/What+is+happening+on+2323TCP/21563/.

[41] Johannes B. Ullrich. Tr-069 newntpserver exploits: What we know so far. https://isc.sans.edu/forums/diary/TR069+NewNTPServer+Exploits+What+we+know+so+far/21763/.

[42] Arvid Norberg. utorrent transport protocol. BEP 29.

[43] Sam Edwards and Ioannis Profetis. Hajime: Analysis of a decentralized internet worm for iot devices. https://security.rapiditynetworks.com/publications/2016-10-16/hajime.pdf.

[44] Cisco Systems. Cisco ios netflow. http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html.

[45] Brad Reese. Netflow versus sflow. http://www.techworld.com/networking/netflow-versus-sflow-3865/.

[46] SolarWinds Worldwide LLC. Netflow basics and deployment strategies. http://www.solarwinds.com/documentation/Netflow/docs/NetFlowBasicsandDeploymentStrategies.pdf.

[47] Brad Hale. Netflow v9 datagram knowledge series: Part 1 - netflow overview. https://thwack.solarwinds.com/community/solarwinds-community/geek-speak_tht/blog/2012/09/06/netflow-v9-datagram-knowledge-series-part-1--netflow-overview.

[48] Ed. B. Claise. Rfc 3954 - cisco systems netflow services export version 9. https://www.ietf.org/rfc/rfc3954.txt.

[49] M. Cotton and L. Vegoda. Rfc 6437 - ipv6 flow label specification. https://www.ietf.org/rfc/rfc6437.txt.

[50] NetFort Technologies. Flow analysis versus packet analysis. what should you choose? https://www.netfort.com/wp-content/uploads/PDF/WhitePapers/NetFlow-Vs-Packet-Analysis-What-Should-You-Choose.pdf.

[51] Michael Scheck. Netflow for incident detection. https://www.first.org/global/practices/Netflow.pdf.

[52] Valentín Carela-Español, Pere Barlet-Ros, Albert Cabellos-Aparicio, Josep Solé-Pareta. Analysis of the impact of sampling on netflow traffic classification. Technical report, Universitat Politècnica de Catalunya, 2010. http://people.ac.upc.edu/pbarlet/papers/netflow-classification.comnet2010.pdf.

[53] Chris Sanders. *Practical Packet Analysis.* No Starch Press, 2011.

[54] Margaret Rouse. promiscuous mode. http://searchsecurity.techtarget.com/definition/promiscuous-mode.

[55] Wireshark.org. Chapter 9. packet dissection. https://www.wireshark.org/docs/wsdg_html_chunked/ChapterDissection.html.

[56] Occupytheweb. How to set up a honeypot & how to avoid them. https://null-byte.wonderhowto.com/how-to/hack-like-pro-set-up-honeypot-avoid-them-0153391/.

[57] Margaret Rouse. honeypot (honey pot). http://searchsecurity.techtarget.com/definition/honey-pot.

[58] Michele Adams Iyatiti Mokube. Honeypots: Concepts, approaches, and challenges. Technical report, Armstrong Atlantic State University, 2007. `http://ai2-s2-pdfs.s3.amazonaws.com/88d5/f45a8fc8c6947cc02e8fb8b9cc3a53227ee3.pdf`.

[59] Michel Oosterhof. Cowrie ssh/telnet honeypot. `https://github.com/micheloosterhof/cowrie`.

[60] P. Mockapetris. Rfc 1035 - domain names - implementation and specification. `https://www.ietf.org/rfc/rfc1035.txt`.

[61] Futai Zou & Siyu Zhang & Weixiong Rao & Ping Yi. Detecting malware based on DNS graph mining. Technical report, Shanghai Jiao Tong University, 2015. `http://journals.sagepub.com/doi/full/10.1155/2015/102687`.

[62] Bradley Mitchell. Top public internet dns servers. `https://www.lifewire.com/top-public-internet-dns-servers-817519`.

[63] Lucian Constantin. Malware increasingly uses dns to avoid detection, experts say. `http://www.computerworld.com/article/2501796/network-security/malware-increasingly-uses-dns-to-avoid-detection--experts-say.html`.

[64] Pieter Arntz. Explained: Domain generating algorithm. `https://blog.malwarebytes.com/security-world/2016/12/explained-domain-generating-algorithm/`.

[65] Information Sciences Institute. Transmission Control Protocol. RFC 793, September 1981.

[66] SANS Internet Storm Center. Port report.

[67] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. ZMap: Fast Internet-wide scanning and its security applications. In *Proceedings of the 22nd USENIX Security Symposium*, August 2013.

[68] Tim Chown. IPv6 Implications for Network Scanning. RFC 5157, March 2008.

[69] Mirko Presser. The rise of Iot - why today? `http://iot.ieee.org/newsletter/january-2016/the-rise-of-iot-why-today.html`.

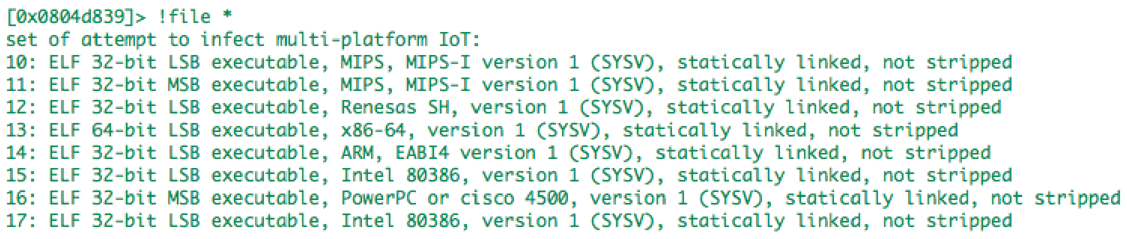
[70] Fernando Gont and Tim Chown. Network Reconnaissance in IPv6 Networks. RFC 7707, March 2016.

[71] IEEE Standards Association. Guidelines for 64-bit global identifier (eui-64). `http://standards.ieee.org/develop/regauth/tut/eui64.pdf`.

[72] Kernelmode.info unixfreaxjp. Linux/bash0day alias shellshock alias bashdoor. `http://www.kernelmode.info/forum/viewtopic.php?f=16&t=3505&start=10`.

Appendices

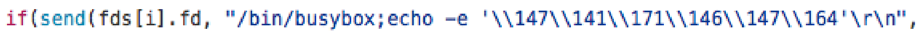
A BASHLITE architectures



```
[0x0804d839]> !file *
set of attempt to infect multi-platform IoT:
10: ELF 32-bit LSB executable, MIPS, MIPS-I version 1 (SYSV), statically linked, not stripped
11: ELF 32-bit MSB executable, MIPS, MIPS-I version 1 (SYSV), statically linked, not stripped
12: ELF 32-bit LSB executable, Renesas SH, version 1 (SYSV), statically linked, not stripped
13: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, not stripped
14: ELF 32-bit LSB executable, ARM, EABI4 version 1 (SYSV), statically linked, not stripped
15: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, not stripped
16: ELF 32-bit MSB executable, PowerPC or cisco 4500, version 1 (SYSV), statically linked, not stripped
17: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, not stripped
```

Figure 15: BASHLITE malware for different architectures [72]

B BASHLITE gayfgt echo

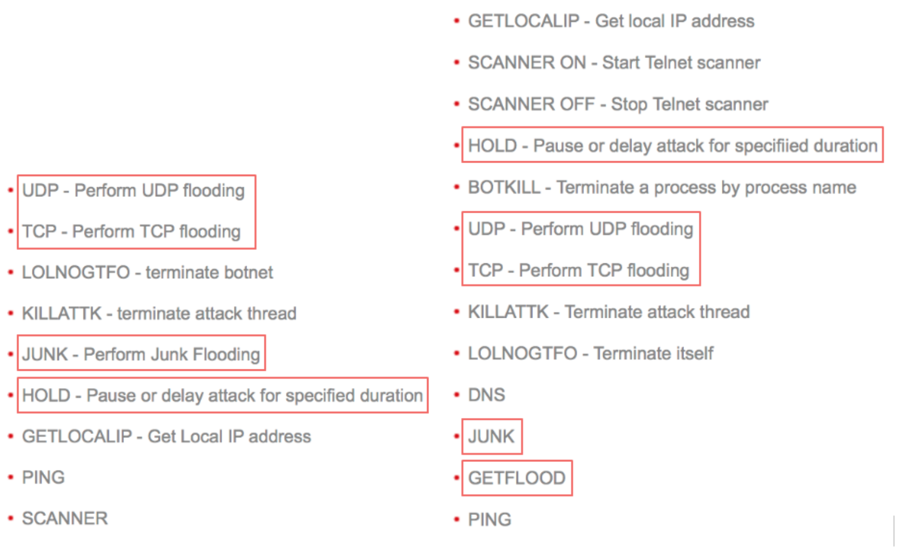


```
if(send(fds[i].fd, "/bin/busybox;echo -e '\\147\\141\\171\\146\\147\\164'\r\n",
```

Source: *bashlite/client.c*

Figure 16: BASHLITE gayfgt in octal representation [22]

C BASHLITE functions



- UDP - Perform UDP flooding
- TCP - Perform TCP flooding
- LOLNOGTFO - terminate botnet
- KILLATTK - terminate attack thread
- JUNK - Perform Junk Flooding
- HOLD - Pause or delay attack for specified duration
- GETLOCALIP - Get Local IP address
- PING
- SCANNER

- GETLOCALIP - Get local IP address
- SCANNER ON - Start Telnet scanner
- SCANNER OFF - Stop Telnet scanner
- HOLD - Pause or delay attack for specifiied duration
- BOTKILL - Terminate a process by process name
- UDP - Perform UDP flooding
- TCP - Perform TCP flooding
- KILLATTK - Terminate attack thread
- LOLNOGTFO - Terminate itself
- DNS
- JUNK
- GETFLOOD
- PING

Figure 17: BASHLITE functions variant A (left) and SMB (right) [24] [26]

D Mirai default credentials

Username	Password	Username	Password
root	admin	admin	admin
root	888888	root	xmhdipc
root	default	root	juantech
root	123456	root	54321
support	support	root	(none)
admin	password	root	root
root	12345	user	user
admin	(none)	root	pass
admin	admin1234	root	1111
admin	smcadmin	admin	1111
root	666666	root	password
root	1234	root	klv123
Administrator	admin	service	service
supervisor	supervisor	guest	guest
guest	12345	guest	12345
admin1	password	administrator	1234
666666	666666	888888	888888
ubnt	ubnt	root	klv1234
root	Zte521	root	hi3518
root	jvbzd	root	anko
root	zlxx.	root	7ujMko0vizxv
root	7ujMko0admin	root	system
root	ikwb	root	dreambox
root	user	root	realtek
root	00000000	admin	1111111
admin	1234	admin	12345
admin	54321	admin	123456
admin	7ujMko0admin	admin	1234
admin	pass	admin	meinsm
tech	tech	mother	fucker

Source: *mirai/bot/table.c*

Table 7: Usernames and passwords included in Mirai

E Mirai supported processor architectures

Sun SPARC
Intel i386
Motorola 68000
Intel x86
MIPS R3000 big-endian
MIPS R3000 little-endian
MIPS R4000 big-endian
HPPA
Sun's "v8plus"
PowerPC
PowerPC64
Cell BE SPU
ARM 32 bit
SuperH
SPARC v9 64-bit
Renesas H8/300
HP/Intel IA-64
AMD x86-64
IBM S/390
Axis Communications 32-bit embedded processor
Renesas M32R
Panasonic/MEI MN10300, AM33
OpenRISC 32-bit embedded processor
ADI Blackfin Processor
Altera Nios II soft-core processor
TI C6X DSPs
ARM 64 bit
Tilera TILEPro
Xilinx MicroBlaze
Tilera TILE-Gx
Fujitsu FR-V
Atmel AVR32

Source: *loader/src/headers/util.h*

Table 8: Processor architectures able to be detected by Mirai's loader

F Mirai strings

Variable	**Decoded value**
TABLE_CNC_DOMAIN	cnc.changeme.com
TABLE_CNC_PORT	23
TABLE_SCAN_CB_DOMAIN	report.changeme.com
TABLE_SCAN_CB_PORT	48101
TABLE_EXEC_SUCCESS	listening tun0
TABLE_KILLER_SAFE	https://youtu.be/dQw4w9WgXcQ
TABLE_KILLER_PROC	/proc/
TABLE_KILLER_EXE	/exe
TABLE_KILLER_DELETED	(deleted)
TABLE_KILLER_FD	/fd
TABLE_KILLER_ANIME	.anime
TABLE_KILLER_STATUS	/status
TABLE_MEM_QBOT	REPORT %s:%s
TABLE_MEM_QBOT2	HTTPFLOOD
TABLE_MEM_QBOT3	LOLNOGTFO
TABLE_MEM_UPX	\x58\x4D\x4E\x4E\x43\x50\x46\x22
TABLE_MEM_ZOLLARD	zollard
TABLE_MEM_REMAITEN	GETLOCALIP
TABLE_SCAN_SHELL	shell
TABLE_SCAN_ENABLE	enable
TABLE_SCAN_SYSTEM	system
TABLE_SCAN_SH	sh
TABLE_SCAN_QUERY	/bin/busybox MIRAI
TABLE_SCAN_RESP	MIRAI: applet not found
TABLE_SCAN_NCORRECT	ncorrect
TABLE_SCAN_PS	/bin/busybox ps
TABLE_SCAN_KILL_9	/bin/busybox kill -9
TABLE_ATK_VSE	TSource Engine Query
TABLE_ATK_RESOLVER	/etc/resolv.conf
TABLE_ATK_NSERV	nameserver
TABLE_ATK_KEEP_ALIVE	Connection: keep-alive
TABLE_ATK_ACCEPT	Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
TABLE_ATK_ACCEPT_LNG	Accept-Language: en-US,en;q=0.8
TABLE_ATK_CONTENT_TYPE	Content-Type: application/x-www-form-urlencoded
TABLE_ATK_SET_COOKIE	setCookie('
TABLE_ATK_REFRESH_HDR	refresh:
TABLE_ATK_LOCATION_HDR	location:
TABLE_ATK_SET_COOKIE_HDR	set-cookie:
TABLE_ATK_CONTENT_LENGTH_HDR	content-length:
TABLE_ATK_TRANSFER_ENCODING_HDR	transfer-encoding:
TABLE_ATK_CHUNKED	chunked
TABLE_ATK_KEEP_ALIVE_HDR	keep-alive
TABLE_ATK_CONNECTION_HDR	connection:
TABLE_ATK_DOSARREST	server: dosarrest
TABLE_ATK_CLOUDFLARE_NGINX	server: cloudflare-nginx
TABLE_HTTP_ONE	Mozilla/5.0 (Windows NT 10.0; WOW64) \ AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36
TABLE_HTTP_TWO	Mozilla/5.0 (Windows NT 10.0; WOW64) \ AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36
TABLE_HTTP_THREE	Mozilla/5.0 (Windows NT 6.1; WOW64) \ AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36
TABLE_HTTP_FOUR	Mozilla/5.0 (Windows NT 6.1; WOW64) \ AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36
TABLE_HTTP_FIVE	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) \ AppleWebKit/601.7.7 (KHTML, like Gecko) Version/9.1.2 Safari/601.7.7

Source: *mirai/bot/table.c*

Table 9: Full list of decoded strings

G Mirai window size generator

```
#define _GNU_SOURCE
#include <stdint.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

void rand_init(void);
uint32_t rand_next(void);

int i;
int num;
static uint32_t x, y, z, w;

void rand_init(void)
{
    x = time(NULL);
    y = getpid() ^ getppid();
    z = clock();
    w = z ^ y;
}

uint32_t rand_next(void) //period 2^96-1
{
    uint32_t t = x;
    t ^= t << 11;
    t ^= t >> 8;
    x = y; y = z; z = w;
    w ^= w >> 19;
    w ^= t;
    return w;
}

int main( int argc, char *argv[] ){
    if (argc != 2) {
        printf("Usage: %s <num>\n", argv[0]);
        exit(1);
    }

   num = atoi(argv[1]);
   fprintf(stderr, "Generating %u window sizes\n", num);
   rand_init();

   for(i = 0; i < num; i++){
       printf("%u\n", rand_next() & 0xffff);
   }
}
```

Source: *based on mirai/bot/rand.c*