Protecting IoT from Mirai botnets; IoT device hardening

Charles Frank
Charles.frank@trojans.dsu.edu

Cory Nance
Cory.nance@trojans.dsu.edu

Sam Jarocki
Samuel.jarocki@trojans.dsu.edu

Dr. Wayne E Pauli
Wayne.pauli@dsu.edu

Dakota State University
Madison, SD

Abstract

This paper details Mirai botnet capabilities, technical components, and original research in realistic hardening measures for protecting Internet of Things (IoT) devices. Mirai, and its various strains embody the consummate actions of nefarious, wide-spreading botnets capable of proliferating to hundreds of thousands of potentially vulnerable Internet of Things (IoT) devices to act as a delivery mechanism for a Distributed Denial of Service (DDoS) attack towards one or more service providing Internet applications. The authors present both a hardening and prevention script, executed on the actual device, to protect devices from becoming malicious bots, as part of the Mirai botnet. In a controlled test environment, the hardening script was shown to be successful in preventing the initial Mirai infection on the device and the detection script was successful in recognizing and stopping an already existing infection on the Mirai bot. The conclusion describes possible research directions.

Keywords: IoT, botnet, Mirai, OS hardening, OS security6

1. INTRODUCTION

Currently, there is an estimated 15 billion Internet of Things (IoT) devices. By 2020, the estimate is projected to be as high as 50 billion connected IoT devices (Higginbotham S, 2016). IoT is comprised of the internetworking of physical devices, smart devices, smart buildings, smart cars, medical device, etc.; embedded with electronics, software, sensors, actuators, and internet connectivity. These objects collect and exchange data (Internet of Things, n.d.). Clearly, there are many IoT devices and that number will grow exponentially over time (Higginbotham S, 2016).

The value of IoT comes from the data it generates. With real-time data analytics, IoT provides insights and improvements (Gorlich, K., 2016). There are many applications for IoT, ranging from non-critical applications, such as wearables (e.g. smart watches), to crucial applications, such as in healthcare (e.g. IoT smart medical device dispensing medicine to hospital patients (IoT Applications with Examples, 2016), to even military and battlefield applications (Goldstein, P., n.d.). Evidentially, IoT applications play an integrated role in people's everyday lives. Depending upon the IoT application, security could be paramount.

2. BOTNET HISTORY

In 1999, Sub7 (Gamblin, J., 2017) and Prettypark (Hariston K., Rozman, N., etal, n.d.) constructed an IRC channel to gain control of victim machines to issue malicious commands. In 2000, Global Threat Bot (GTBot) was based on the mIRC client (Fandom, n.d.). GTBot could run custom scripts in response to IRC events and had access to TCP and UDP sockets, allowing for Denial of Service (DoS) attacks. Also, GTBot scanned for Sub7 infected hosts and updated them to GTbots (Global Threat Bot, 2017).

In 2002 notable evolutions in botnet technology was observed with SDBot and Agobot. SDBot's source code was released to the public via the author; thus many subsequent bots include code or ideas from SDBot (Trend Micro, "Countermeasures..., n.d.). Agobot introduced the concept of a modular, staged attack, as payloads were delivered sequentially (Trend Micro, "SDBOT", n.d.). The initial attack installed a back door and used stealth techniques to avoid detection from antivirus. These early botnets concentrated on remote control and information theft (Global Threat Bot, 2017).

More advanced bot functionality began to set the stage for greater data exfiltration and service disruption and circumvention techniques. In 2003 Spybot (aka Rbot) included keylogging, information stealing, spam, and DDoS capabilities (Argobot, n.d.). The command and control (CNC) was conducted over IRC. Sinit was the first peer-to-peer botnet (Dark Reading, n.d.). Polybot employed polymorphism to avoid detection by changing its appearance as often as possible (Global Threat Bot, 2017). Later in 2005, Bagle and Bobax were the first spamming botnets, and the malware Mytob was a mailing worm based upon MyDoom and SDbot (Trendmicro, WORM_SPYBOT.A, n.d.); enabling large botnets distributed across many infected PCs. Soon after, in 2006, another invasive spamming botnet RuStock (Trendmicro, CounterMeasures Security, Privacy & Trust, n.d.) appeared, utilizing self-propagation. Undoubtedly, in a short period of time, botnets started to become more sophisticated in attacking, evading detection, and multiplying.

ZeuS is an information stealing tool that first appeared in 2010. ZeuS quickly became the most widely used information stealing botnet. Part of its appeal is that it includes simple point and click interfaces for managing infected machines. Zeus is regularly updated and new versions have been offered for sale, while older versions have been distributed online free of charge (Trendmicro, WORM_SPYBOT.A, n.d.). At this point, not only have botnets gotten more sophisticated in their method of infection via email spamming but they are now concerned with ease of use via point and click interfaces.

2014 witnessed many high-profile attacks; from an internet-connected refrigerator participating in a botnet sending over 750,000 spam emails (Rapid7, IOT Seeker, n.d.) to a DDoS attack of IoT devices successfully affecting availability of Sony and Microsoft's gaming networks Constantin, L., 2017). In December 2016, researchers from Imperva detected a colossal 650 Gbps DDoS attack generated by a new IoT botnet, named Leet (Simonroses.com, n.d.).

In April of 2017, Unit 42 researchers have identified a new variant of the IoT Linux botnet Tsunami, coined Amnesia (Jia, Y., Xiao, C., & Zheng, C., 2017). Amnesia targets an unpatched remote code execution vulnerability that was publicly disclosed in March 2016 in DVR (digital video recorder) devices made by TVT Digital. It is believed Amnesia is the first Linux malware to adopt virtual machine evasion techniques to defeat malware analysis sandboxes. Currently, Amnesia has not been used to mount large scale attacks.

Shown in Fig. 1, Wikipedia (Zeus, n.d.) presents a historical list of botnets, with many of the botnets described in the previous paragraphs. Currently, there are thousands of botnets that the Shadowserver Foundation is tracking (Botnet, n.d.). Typically, Trend Micro tracks tens of millions of infected PCs that are being used to send spam; and that does not include all the other infected PCs that are being used for information theft, DDoS or other botnet crimes (Trendmicro.eu, 2017).

3. MIRAI BOTNET

The Mirai botnet wreaked havoc on the internet in 2016. The botnet takes advantage of unsecured IoT devices that leave administrative channels (e.g. telnet/SSH) open and use well known, factory default, usernames and passwords. Mirai scans the internet looking for new systems to infest, such as those manufactured by XiongMai Technologies that had default passwords set in their firmware (prior to September 2015) which cannot be changed unless upgraded. These devices are especially vulnerable to the Mirai botnet, as well as other exploit payloads due to their insecure default firmware (Buntinx J.P., 2016). Mirai's size makes it a very powerful botnet capable of producing massive throughput. For example, in September of 2016, the Mirai botnet is reported to have generated 620 Gbps in its DDoS attack on "Kreb's on Security" (Mirai, n.d.).

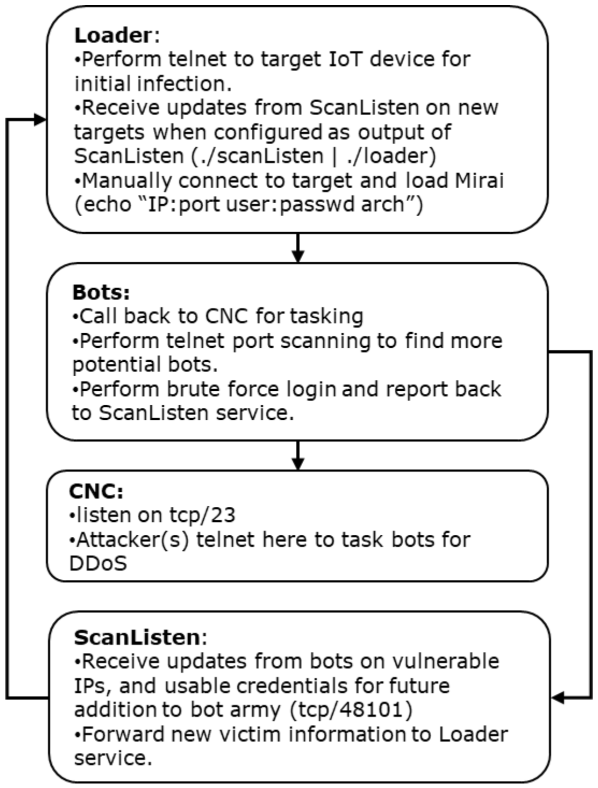
In October 2016, the source code for Mirai was leaked on HackForums (ShadowServer, n.d.). This release has helped security researchers to better understand Mirai capabilities and how it works. Mirai performs wide-ranging scans of IP addresses with the intentions of locating IoT devices that can be remotely accessed via easily guessable login credentials, usually factory default usernames and passwords (e.g., admin/admin) (ShadowServer, n.d.).

Mirai is using several functions from the Linux kernel API related to network operations. For example, in killer.c there is a function named *killer_init* that kills several services: telnet (port 23), ssh (port 22) and http (port 80) to prevent others from accessing the compromised IoT device. (Femerling, 2016).

Mirai comes with a list of default/weak passwords to perform brute force attacks on IoT devices [29]. Mirai's attack function enables it to launch HTTP floods and various network (OSI layer 3-4) DDoS attacks. For network layer assaults, Mirai is capable of launching GRE IP and GRE ETH floods, SYN and ACK floods, STOMP (Simple Text Oriented Message Protocol) floods, DNS floods and UDP flood attacks (ShadowServer, n.d.).

There is even a "don't mess with" list for IP addresses (e.g. the United States Post Office, Dept. of Defense, and private IP space) and several killer scripts meant to eradicate other worms and trojans [29]. Since the Mirai source code has been leaked, many variants have been detected. A few interesting variants include: the use of a DGA (Domain Generation Algorithm) Incapsula.com, n.d.) and trojanized Windows payloads that incorporate Mirai scanning (cfengine.com, n.d.).

To conclude, each bot scans for new bots to infect using the default list of usernames and passwords. Once a bot finds a new vulnerable device it forwards the IP, port, credentials, and device architecture to the ScanListener.


Loader:
•Perform telnet to target IoT device for initial infection.
•Receive updates from ScanListen on new targets when configured as output of ScanListen (./scanListen | ./loader)
•Manually connect to target and load Mirai (echo "IP:port user:passwd arch")
Bots:
•Call back to CNC for tasking
•Perform telnet port scanning to find more potential bots.
•Perform brute force login and report back to ScanListen service.
CNC:
•listen on tcp/23
•Attacker(s) telnet here to task bots for DDoS
ScanListen:
•Receive updates from bots on vulnerable IPs, and usable credentials for future addition to bot army (tcp/48101)
•Forward new victim information to Loader service.
Figure 2 Mirai Architecture

The ScanListener does the part of actually infecting the device. Once the IoT device has been infected with the Mirai malware via telnet and has become a bot, the CNC will communicate with the bot to execute DDOS attacks.

4. BOTNET DETECTION AND PREVENTION

Recent studies from the INSuRE (Information Security Research and Education) research group have focused on IoT botnets (INSuRE, Online). In Kovacoc and Vargas (n.d.), an analysis of current botnets and botnet operations, command and control infrastructure, and detection approaches were presented. Rudesh (n.d.) determines the characteristics of Thingbots, identifies IoT devices that can participate in the botnet and determines a detection, isolation, and mitigation technique for Thingbots by reviewing existing techniques. Another project detected IoT botnets through the spreading of the hosts which have the botnet detection tool installed on them. Baki presents peer-to-peer botnet detection through Machine Learning (ML) (n.d.) (Abay, C., Hagel, L., & Williams, K., n.d.) isolates and analyzes a Zeus botnet node, and (Freeman, L., Hickey, R., etal, n.d.) develops a testbed for botnet countermeasures.

There are also efforts to secure IoT devices. One novel approach is using blockchain technology where security software on the kernel of the IoT device could receive a blacklist of IP addresses over the blockchain. (Faife, C., n.d.). Another study found a stack buffer overflow vulnerability in the Mirai malware that allows the malware to be crashed on the bot (Leyden, J., 2016). Lastly, an anti-worm "nematode" has been developed that could help to patch vulnerable devices and to help prevent Mirai bots (Pauli, D., 2016).

Fig. 3 illustrates international research conducted by a team from Japan and Germany. The team, led by Yin Minn Pa Pa, and Shogo Suzuki authored an article on analyzing the rise of IoT compromises. The increasing threats against IoT devices show that telnet-based attacks that target IoT devices have skyrocketed since 2014 (Pa Pa, Y, Suzuki, S, etal, n.d.). With analysis from IotPOT, a honeypot for IoT, Fig. 3 indicates that there are at least four distinct DDoS malware families targeting telnet-enabled IoT devices.

Many of the patterns have common command sequences such as checking for the victim's shell and then eventually downloading the malicious binary. Compared to the other patterns, ZORRO 3 contained many more command sequences per day.

ShadowServer (Botnet, n.d.) suggests the best way to mitigate botnets is to keep them from forming. Botnets would not be a threat if they could not propagate and infect vast numbers of systems. IoT Seeker (Seals, T., n.d.) scans for IoT devices which could easily be hijacked by botnets. Methods of preventing IoT botnets from spreading are suggested by stopping the use of default/generic passwords and disabling all remote (WAN) access to your devices (ShadowServer, n.d.). CFEngine (Arghire, I., n.d.) significantly reduces end-point attack surface by: (1) closing any unnecessary services, especially remote access services, (2) changing factory default user accounts, (3) removing unnecessary software, and (4) avoiding legacy protocols and password logins. With the recent advent of trojanized Windows payloads that incorporate Mirai scanning and reporting within an intranet environment (cfengine.com, n.d.), the security offered by rejecting and blocking publicly accessible ports/services is diminished.

5. PROPOSED IOT HARDENING SCRIPTS

Two scripts are proposed for hardening IoT devices from Mirai: (1) antimirai.py and (2) secure.sh [69]. antimirai.py is a python script that makes various changes on the IoT device, such as: (1) changing the default password (2) creating a busybox wrapper to filter out applets used by Mirai (3) changing the logon banner and (4) implementing */etc/host.deny*. These changes attempt to prevent the infection of Mirai on the IoT device.

Shown in Fig. 4, *replace_busybox()* will copy the existing busybox binary, on the IoT device, to *tmp_busybox*. Then, a busybox wrapper is created and the commands that are executed by the Mirai loader to upload the malware are detected [*words="telnet wget tftp"*]. In an attempt to prevent Mirai infection, these commands will return a success [0], even though the commands are prevented from being executed on the actual IoT device.

Shown in Fig. 5, the *change_passwd_telnet()* method will generate a new random password for the administrator of the IoT device. Lastly, *upload_run_script()* will upload and run the *secure.sh* script. *secure.sh* is a script that detects Mirai infections and reacts by stopping the Mirai malware from running.

Fig. 6 shows *secure.sh*, a busybox ash (Almquist shell) script. Once a bot is infected with Mirai, it opens a connection back to the CNC server on port 23 and also runs 3 processes with the same randomly generated name. This script works by checking */proc/net/tcp* for a socket that has a remote connection to port 23 (0x17). It then locates the PID of the socket. With the socket's PID, the script locates the process's name and then sends the SIGKILL signal to each process with the same name, effectively stopping the infection and any communication with the CNC server.

In conclusion, *antimiarai.py* is a python script to harden the IoT device from Mirai infections. Not only will *antimaria.py* make configuration changes that prevent Mirai infections but it will also upload *secure.sh*. The script *secure.sh* is an ash script that will continuously check for an indication of the Mirai malware running. Once Mirai is found to be running, it is immediately killed. This combination of scripts should prevent an IoT device from becoming a part of the Mirai botnet.

6. TESTING ENVIRONMENT

The testing environment runs in virtual machines (VMs) on an isolated private network. It consists of two VMs. One houses the CNC and loader, while the other represents a vulnerable Linux-based IoT device. The vulnerable VM is running Ubuntu 14.04 with busybox, is configured with a default username and password, and is running busybox's telnetd on port 23. For each test, the loader was manually executed to attempt Mirai infections against the vulnerable VM.

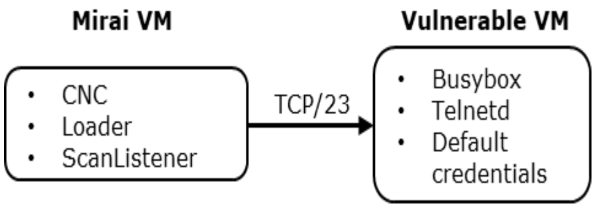


Figure 7 Testing Server Equipment

As shown above, the Mirai server and the simulated IoT Device contain private IP addresses. These private IP addresses isolate the testing environment from the publically routable internet. A Vagrant file (http://vagrantup.com) was used to orchestrate the creation of the VMs and private network (Nance, C., n.d.).

7. TESTING RESULTS

The hardening script (*antimirai.py*) was tested to determine the feasibility and outcome from basic protection (changing default password), obfuscation (modifying banner and changing server port), and redirection (wrapping busybox applets used for malicious functions). Not all functionality was incorporated due to platform and time limitations. Platform limitations such as telnet and/or ssh services not being compiled with the tcp wrapper library (libwrap) lack the host-based access control lists system to leverage additions to */etc/hosts.allow* and */etc/hosts.deny*, thereby rendering that specific hardening action ineffectual. Inability to disable/modify the superuser account (e.g. root), while not hindering device functionality, constituted a time limitation. Additional obfuscation techniques would provide demonstrations of change without furthering a proof-of-concept.

Execution of *antimirai.py* hardening script produced predictable results based on the testing environment and conditions. Issuing a change of default password to a random alphanumeric string is an effectual method for thwarting Mirai's scanner, and subsequent infection. Changing banner (via */etc/motd, /etc/issue, /etc/issue.net*) was not successful in preventing infection. Mirai inspects login prompt, such as *$, :, #*, etc. provided by telnetd/sshd. There are multiple methods to changing login prompt based on platform; and available commands and configurations on host, thereby not feasible for implementation within time constraints. Modification of default service port prevented infection, however this method does not prevent a port scan from discovering new listening port.

Action	Expected Outcome	Actual Outcome
Harden: change password	Not infected	Not infected
Harden: change banner	Not infected	Infected
Harden: change service port	Not infected	Not infected
Harden: change user	Not infected	Not implemented
Harden: wrap busybox applets	Not infected	Not infected
Harden: Upload secure.sh script	Infected then removed	Infected then removed

Figure 8 antimirai.py test results

Service detection paired with banner return may offer additional obfuscation (not tested). On-the-fly creation and deployment of a busybox wrapper script to intercept applets Mirai requires to download it's binary to a target device (e.g. wget, telnet, ftp) was successful in preventing infection.

Finally, deployment and execution of the *secure.sh* script, barring any other hardening techniques, successfully terminated repeated Mirai bot infections on target host. The *secure.sh* script was successful in detecting and subsequently terminating all infection attempts based on defined parameters of time and service port detection. Logically, any combination of multiple hardening techniques deployed to a viable host would offer increased protection within their individual limitations to provide a multi-faceted strategy of defense.

8. CONCLUSION

While this paper has explored multiple methods of stopping or detecting a Mirai infection, there is still more work that needs to be done. New strains of Mirai could drastically change the way the botnet malware is delivered or behaves, which could in turn render many of the methods used here, useless. The scripts, antimarai.py and secure.sh, need to be tested on the public internet to see how they perform against new strains of Mirai.

Additionally, more indicators of compromise need to be found and developed for Mirai to help identify an infection. The current method used can generate a false positive an IoT device legitimately needs to communicate over port 23 to a remote server. New strains of Mirai could avoid detection by changing the port it uses for bot communication or change the name of its processes.

This research proposed methods to harden IoT devices from becoming bots controlled by the Mirai botnet. Two hardening scripts have been proposed: (1) *antimirai.py* and (2) *secure.sh*. *antimirai.py* is a python script that makes configuration changes to prevent Mirai infections. *secure.sh* is an ash script, uploaded via *antimirai.py*, that is constantly searching for and closing processes that are identified as being part of Mirai. Testing in a controlled laboratory environment, with a simulated IoT device, has shown that *antimirai.py* is successful in preventing Mirai infections and *secure.sh* is successful in detecting and stopping an infection.

9. REFERENCES

Abay, C., Hagel, L., & Williams, K. (n.d.). Peer-to-Peer Botnet Detection, Retrieved 29-Jan-2017 from https://purr.purdue.edu/projects/insurefall2016/files/browse?subdir=Projects/Botnet%20Study.

Arghire, I., (n.d.) New Mirai Variants Have Built-in Domain Generation Algorithm, Retrieved from http://www.securityweek.com/new-mirai-variants-have-built-domain-generation-algorithm

Argobot, (n.d.). Wikipedia. Retrieved 07-Feb-2017 from https://en.wikipedia.org/wiki/Agobot.

Baki, S., (n.d.). Network Under Control: Optimal Node Selection for Installing Botnet Detection Software Retrieved 29-Jan-2017 from https://purr.purdue.edu/projects/insurefall2016/files/browse?subdir=Projects/Botnet%20Study.

Botnet, (n.d.). Wikipedia Retrieved 07-Feb-2017 from https://en.wikipedia.org/wiki/Botnet.

Buntinx, J.P., (2016, Oct. 24). XiongMai Technologies Admits Their Devices Are Susceptible To Mirai Malware. *The Merkle*. Retrieved 30-Jan-2017 from https://themerkle.com/xiongmai-technologies-admits-their-devices-are-susceptible-to-Mirai-malware/.

Buntinx, J.P., (n.d.), Updated Mirai Botnet Malware Executes 54-hour DDoS Attack Retrieved 09-April-2017 from https://themerkle.com/updated-mirai-botnet-malware-executes-54-hour-ddos-attack/.

CFEngine, (n.d.), Industrial Internet of Things – Systems Hardening, Retrieved from https://cfengine.com/solutions/industrial-iot-systems-hardening/

Constantin, L., (n.d.) Windows Trojan hacks into embedded devices to install Mirai, *PCWorld*, 09-Feb-2017

DarkReading, (n.d.). The World's Biggest Botnets, Retrieved 07-Feb-2017 from http://www.darkreading.com/the-worlds-biggest-botnets-/d/d-id/1129117?

Faife, C., (n.d.). This Bitcoin Botnet is Vying to Be Future of Secure IoT Retrieved 09-April-2017 from http://www.coindesk.com/this-bitcoin-botnet-is-vying-to-be-future-of-secure-iot/

Fandom, (n.d.). Virus Information. Prettypark, Retrieved 06-Feb-2017 from http://virus.wikia.com/wiki/Prettypark.

Femerling, S.R., (n.d.), "Mirai DDoS Botnet: Source Code & Binary Analysis," Retrieved from http://www.simonroses.com/2016/10/mirai-ddos-botnet-source-code-binary-analysis/

Freeman, R., Hickey, R., Robertson, J., & Yeske, J., (n.d.).Botnet Study Retrieved 29-Jan-2017 from https://purr.purdue.edu/projects/insurefall2

016/files/browse?subdir=Projects/Botnet%20Study.

Gamblin, J., (2017, Jan.07). Mirai-Source-Code. *GitHub*. Retrieved 30-Jan-2017 from https://github.com/jgamblin/Mirai-Source-Code.

Global Threat Bot (GTBot). (2017, Feb. 8). Technopedia Retrieved 08-Feb-2017 from https://www.techopedia.com/definition/59/global-threat-bot-gtbot.

Goldstein, P., (2016, May). The Internet of Things for the Battlefield Needs to Be Flexible, Army Official Says Retrieved 09-APR-2017 from http://www.fedtechmagazine.com/article/2016/05/internet-things-battlefield-needs-be-flexible-army-official-says.

Görlich, K., (2016, Jun. 20). Live Business: The Importance of the Internet of Things. *Digitalist Magazine*

Hariston, J., Rozman, K., Sissom, N., & Wright, D., (n.d.). Botnet Counterstrike: Implementation of Botnet Enclave Testbed Retrieved 29-Jan-2017 from https://purr.purdue.edu/projects/insurefall2016/files/browse?subdir=Projects/Botnet%20Study.

Hertig, A., (n.d.), Mirai, The Infamous Internet of Things Army, Can Now Mine Bitcoin Retrieved 10-April-2017 from http://www.coindesk.com/mirai-infamous-internet-things-army-can-now-mine-bitcoin/

Higginbotham, S. (2016, Mar. 18). Prediction: there won't be 50B connected IoT devices by 2020. *Structure Connect*. Retrieved 28-Jan-2017 from http://www.structureconnect.com/prediction-there-wont-be-50b-connected-iot-devices-by-2020/.

Imperva Incapsula (n.d.), Breaking Down Mirai: An IoT DDoS Botnet Analysis, Retrieved 29-Jan-2017 from https://www.incapsula.com/blog/malware-analysis-mirai-ddos-botnet.html

INSuRE, Information Security Research and Education. (n.d.). Retrieved 29-Jan-2017 from https://purr.purdue.edu/projects/insurefall2016/files/browse?subdir=Projects/Botnet%20Study.

IoT Applications with Examples. (2016, Oct, 24). Internet of Things Wiki. Retrieved 28-Jan-2017 from http://internetofthingswiki.com/iot-applications-examples/541/

Internet of Things. (n.d.). Retrieved 28-Jan-2017 from https://en.wikipedia.org/wiki/Internet_of_things

Jia, Y., Xiao C., & Zheng, C., (April 2017) New IoT/Linux Malware Targets DVRs, Forms Botnet. Retrieved April 9, 2017 from http://researchcenter.paloaltonetworks.com/2017/04/unit42-new-iotlinux-malware-targets-dvrs-forms-botnet/?utm_source=hs_email&utm_medium=email&utm_content=50167168&_hsenc=p2ANqtz-9HGnfET3w5_BRVaC_tp_iEiHppZRK2tQPfem4dhiM3iP-7N6HvbaHLQBBKeebc_OFkSk_mw_1A7uzGlXIIUIt8HaASWw&_hsmi=50167168

Kovacoc, T., & Vargas, J., Botnet Study Retrieved 29-Jan-2017 from https://purr.purdue.edu/projects/insurefall2016/files/browse?subdir=Projects/Bo

Leyden, J., (2016, Oct) Researchers expose Mirai vulnerabilities that could be used to hack back against botnet Retrieved 09-April-2017 from http://www.theregister.co.uk/2016/10/28/mirai_botnet_hack_back/botnet%20Study.

Mirai. (n.d.). Wikipedia. Retrieved 26-Jan-2017 from https://en.wikipedia.org/wiki/Mirai.

Nance, C. (n.d.). miai. *GitHub* Retrieved 10-APR-2017. from https://github.com/canance/mirai?files=1

Pa Pa, Y.M., Suzuki, S., Yoshioka, K., Matsumoto, T., Kasama, T., & Rossow, C., (n.d.) IoTPOT: analyzing the rise of IoT compromises Retrieved 26-Feb-2017 from https://www.usenix.org/system/files/conference/woot15/woot15-paper-pa.pdf

Pauli, D., (2016, Oct). Boffin's anti-worm bot could silence epic Mirai DDoS attack army Retrieved 09-April-2017 from https://www.theregister.co.uk/2016/10/31/this_antiworm_patch_bot_could_silence_epic_mirai_ddos_attack_army/

Rapid7, (n.d.). IOT Seeker, Retrieved from https://information.rapid7.com/iotseeker

Rudesh, V., (n.d.). Thing bot Analysis and Detection Retrieved 29-Jan-2017 from https://purr.purdue.edu/projects/insurefall2016/files/browse?subdir=Projects/Botnet%20Study.

Seals, T., (n.d.). Leet IoT Botnet Bursts on the Scene with Massive DDoS Attack, Retrieved from https://www.infosecurity-magazine.com/news/leet-iot-botnet-bursts-on-the-scene/

ShadowServer, (n.d.). Retrieved 7-Feb-2017 from https://www.shadowserver.org/wiki/.

Trend Micro, (n.d.). CounterMeasures Security, Privacy & Trust. The history of the botnet – Part I Retrieved 06-Feb-2017 from http://countermeasures.trendmicro.eu/the-history-of-the-botnet-part-i/.

Trend Micro, (n.d.). CounterMeasures Security, Privacy & Trust. The history of the botnet – Part II Retrieved 08-Feb-2017 from http://countermeasures.trendmicro.eu/the-history-of-the-botnet-part-ii/.

Trend Micro, (n.d.). SDBOT, Retrieved 06-Feb-2017 from http://countermeasures.trendmicro. .com/vinfo/us/threat-encyclopedia/malware/sdbot

Trend Micro, (n.d.). WORM_SPYBOT.A Retrieved 07-Feb-2017 from https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/WORM_SPYBOT.A

Zeus, (n.d.). Wikipedia. Retrieved 07-Feb-2017 from https://en.wikipedia.org/wiki/Zeus.

APPENDIX

Date created	Name	Estimated no. of bots	Aliases
2004 (Early)	Bagle	230,000	Beagle, Mitglieder, Lodeight
	Marina Botnet	6,215,000	Damon Briant, BOB.dc, Cotmonger, Hacktool.Spammer, Kraken
	Torpig	180,000	Sinowal, Anserin
	Storm	160,000	Nuwar, Peacomm, Zhelatin
2006 (around)	Rustock	150,000	RKRustok, Costrat
	Donbot	125,000	Buzus, Bachsoy
2007 (around)	Cutwail	1,500,000	Pandex, Mutant (related to: Wigon, Pushdo)
2007	Akbot	1,300,000	
2007 (March)	Srizbi	450,000	Cbeplay, Exchanger
	Lethic	260,000	none
2007 (September)	dBot	10,000+ (Europe)	dentaoBot, d-net, SDBOT
	Xarvester	10,000	Rlsloup, Pixoliz
2008 (around)	Sality	1,000,000	Sector, Kuku
2008 (around)	Mariposa	12,000,000	
2008 (November)	Conficker	10,500,000+	DownUp, DownAndUp, DownAdUp, Kido
2008 (November)	Waledac	80,000	Waled, Waledpak
	Maazben	50,000	None
	Onewordsub	40,000	
	Gheg	30,000	Tofsee, Mondera
	Nucrypt	20,000	Loosky, Locksky
	Wopla	20,000	Pokier, Slogger, Cryptic
2008 (around)	Asprox	15,000	Danmec, Hydraflux
	Spamthru	12,000	Spam-DComServ, Covesmer, Xmiler
2008 (around)	Gumblar		
2009 (May)	BredoLab	30,000,000	Oficla
2009 (Around)	Grum	560,000	Tedroo
	Mega-D	509,000	Ozdok
	Kraken	495,000	Kracken
2009 (August)	Festi	250,000	Spamnost
2010 (January)	LowSec	11,000+	LowSecurity, FreeMoney, Ring0.Tools
2010 (around)	TDL4	4,500,000	TDSS, Alureon
	Zeus	3,600,000 (US only)	Zbot, PRG, Wsnpoem, Gorhax, Kneber
2010	Kelihos	300,000+	Hlux
2011 or earlier	Ramnit	3,000,000	
2012 (Around)	Chameleon	120,000	None
2016 (August)	Mirai (malware)	380,000	None

Figure 1 Wikipedia Historical Timeline of Botnets

Pattern Name	Pattern of Command Sequence	Set of Command Sequence Day - Avg
ZORRO 1	1. Check type of victim shell with command "sh" 2. Check error reply of victim by running non-existing command-- ZORRO. 3. Check whether wget command is usable or not. 4. Check whether busybox shell can be used or not by echoing ZORRO.	#
	5. Remove various command and files under /usr/bin/, /bin, var/run/, /dev. 6. Copy /bin/sh to random file name 7. Append series of binaries to random file name of step 6 and make attacker's own shell 8. Using attacker's own shell, download binary . IP Address and port number of malware download server can be seen in the command. 9. Run binary	
ZORRO 2	1. Check type of victim shell with command "sh" 2. Check error reply of victim by running non-existing command - ZORRO. 3. Check whether wget command is usable or not. 4. Check whether busybox shell can be used or not by echoing ZORRO.	#
	6. Copy /bin/sh to random file name 7. Append series of binaries to random file name of step 6 and make attacker's own shell 8. Using attacker's own shell, download binary . IP Address and port number of malware download server cannot be seen in the command because it is hard coded in the attacker's own shell. 9. Run binary	
ZORRO 3	1. Check type of victim shell with command "sh" 2. Check error reply of victim by running non-existing command - ZORRO. 3. Check whether wget command is usable or not. 4. Check whether busybox shell can be used or not by echoing.	174
	5. Remove all under /var/run, /dev, /tmp, /var/tmp 6. Copy /bin/sh to random file name 7. Append series of binaries to random file name of step 6 and make attacker's own shell 8. Using attacker's own shell, download binary. IP Address of malware download server can be seen in the command and port number cannot be seen in the command 9. Run binary	1,353
Bashlite	1. Check whether shell can be used or not by echoing "gayfgt" 2. Download shell script. 3. Using downloaded shell script, kill previously running malicious process, download malware binaries of different CPU architectures and block 23/TCP in order to prevent other infection. 4. Run all downloaded malware binaries.	606
nttpd	1. Check whether shell can be used or not by echoing "welcome" 2. Download binary to /tmp directory. 3. Run Binary.	3.2
KOS	1. Check whether shell can be used or not by echoing "$?K_O_S_T_Y_P_E" 2. List /proc/self/exe 3. Check all running process 4. Download malware binary using tftp to /mnt folder 5. Run Malware 6. Check CPU information	3.5

Figure 3 IoT Pot Patterns of Attack

```
…
def replace_bosybox(tn):
       tn.read_until(CMD_PROMPT, 1)
       tn.write('echo $(which busybox) > tmp_busybox; cp $(cat tmp_busybox)
$(cat tmp_busybox).' + DATETIME + '\n')
       tn.write('if [ ! -f "${mybusybox}.bin" ]; then cp $(cat tmp_busybox)
$(cat tmp_busybox).bin; fi\n')
       tn.write('echo \'#!/bin/sh\' > tmp_bb\n')
       tn.write('echo \'mybusybox=$(which busybox)\' >> tmp_bb \n')
       tn.write('echo \'BADFLAG=0  \'  >> tmp_bb \n')
       tn.write('echo \'string="$*" \'  >> tmp_bb \n')
       tn.write('echo \'words="telnet wget tftp" \'  >> tmp_bb \n')

       tn.write('echo \'for word in $words; do if [ "${string#*$word}" !=
"$string" ]; then return 0; else BADFLAG=1; fi; done \'  >> tmp_bb \n')
       tn.write('echo \'if [ $BADFLAG = 1 ]; then ${mybusybox}.bin "$@"; fi
\'  >> tmp_bb \n')
       tn.write('mv tmp_bb $(cat tmp_busybox); chmod +x $(cat
tmp_busybox)\n')
       print tn.read_until(CMD_PROMPT, 1)
```

Figure 4 Replace_busybox

```
...
def change_passwd_telnet(tn):
      p = random_gen()
      tn.write("passwd " + user + "\n")
      tn.read_until("(current) UNIX password: ")
      tn.write(password + "\n")
      tn.read_until("Enter new UNIX password: ")
      tn.write(p + "\n")
      tn.read_until("Retype new UNIX password: ")
      tn.write(p + "\n")
      targetDetails = "%s:%d:%s:%s:%s" % (target, port, proto, user, p,)
      log.info("Changed values: \t%s" % targetDetails)
...
def upload_run_script():
      ...
      with open(file_exec) as f:
         content = f.read()
      _execFile = file_exec.strip('.\\')
      # convert file contents to base64 and split into chunks to send
reliably over telnet
      content_serialized = split_by_length(base64.b64encode(content),
FILE_CHUNK)
      execFile = RUN_LOCATION + DATETIME + "_" + _execFile
      decodedFile = RUN_LOCATION + DATETIME + "_RUN_" + _execFile
      ...
      # write file in FILE_CHUNK sections
      for c in content_serialized:
         tn.write("echo \"" + c + "\" >> " + execFile + " \n")
         tn.read_until(CMD_PROMPT, 3)
      ...
      print tn.read_until(CMD_PROMPT, 3)
      # execute script on device
      tn.write("cd " + RUN_LOCATION + " && /usr/bin/nohup /bin/sh " +
decodedFile + " " + arg_str +
              " >/dev/null 2>&1 &\n")
  print tn.read_until(CMD_PROMPT, 3)
```

Figure 5 antimmirai.py

```
PS="/bin/busybox ps"
while true; do
      socket=$(grep /proc/net/tcp -e '[0-9]*: [A-Z0-9]*:[A-Z0-9]\{4\} [A-Z0-
9]\{8\}:0017' | tr -s ' ' | cut -d' ' -f 11)
      if [ ! -z "$socket" ]; then
      master_pid=$(find /proc/ -type l 2>/dev/null | grep /fd/ | xargs ls -la
2>/dev/null | grep $socket | head -1 | tr -s ' ' | cut -f 9  -d ' ' | cut -f 3 -d '/')
      name=$($PS aux | grep $master_pid | head -1 | tr -s ' ' | cut -d ' ' -f 4)
      $PS aux | grep $name | sed \$d | awk '{print $1}' | xargs kill -9 2>/dev/null
      fi
      sleep 2
done
```

Figure 6. secure.sh