Hella Booters

•••

Why IoT Botnets Aren’t Going Anywhere

Who Am I?

- Senior Reverse Engineer @ *REDACTED*
- Primarily works on embedded devices, firmware, ICS, and proprietary network protocols.
- Online as netspooky / yuu
- Contributes OSS tooling and other errata for threat intel, reverse engineering, and offensive security.

Why do this talk?

- IoT botnets are still incredibly prevalent.
- We are all affected by IoT botnets whether we like it or not.
- People don't take IoT botnets as seriously (kiddie stuff etc.)

- Spent a good amount of time collecting malware sources
- Studied commonly exploited vulnerabilities and why they were so prevalent

- Wanted to inform others on the impact of certain technology choices
- Wanted to propose some ideas for how to address issues

Outline

IoT Botnet History	A brief overview of IoT Botnets and Modern History
Botnet Scene	An examination of the botnet scene
Architecture	A discussion of botnet architecture and how they spread
Vulnerabilities	Overview of exploited vulnerabilities and their underlying causes
Moving Forward	Things we can do to lessen the impact of IoT Botnets

IoT Botnet History

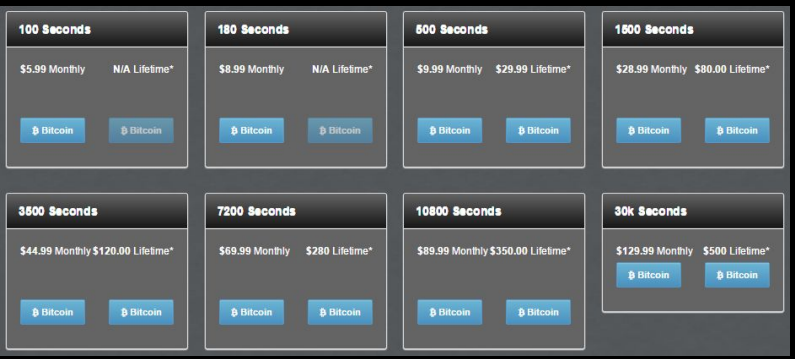
What is an IoT Botnet?

A network of hacked IoT devices, mainly comprised of internet connected devices, routers, set top boxes, webcams, etc.

Used primarily for DDoS

Sometimes used for cryptocurrency mining and tunneling/proxying traffic.

Origins - BASHLITE (2014)



Origins of modern IoT botnets can be traced back to BASHLITE (AKA so many things: lulzbot, Torlus, Lizkebab, LizardStresser, Ballpit, Gafgyt, and a few other names), a botnet that spread by exploiting shellshock vulnerabilities in Busybox on various devices. Note: There were many different bots that were distributed during this time, Kaiten (IRC based), and a number of perl based "shell bots". [1]

The source code was leaked in 2015, and many people began modifying it.

These botnets came to be known as "QBOT", which is unrelated to the "Qakbot" banking malware.

New devices with unpatched shellshock vulns still appear online to this day.

Mirai (2016)

Mirai first appeared in August 2016. The streamlined command and control structure allowed for much more stable management and operation of the botnet. [2]

Mirai's codebase was far more modular, using multiple files and standardized functionality that made it easily modifiable by even the most novice of botnet operators. [3]

It also included an SQL server for the backend, which made user management, roles, and permissions for running the botnet much easier.

Pictured: Mirai Author Anna-senpai

Other IoT Botnets

Satori/F-Bot/Okiru: A well known Mirai fork that was more actively developed and contained far more advanced evasion and propagation techniques than others. Author recently jailed. [4]

BrickerBot: A destructive botnet that promised to "brick" IoT devices. Various iterations. [5]

Kaiji: Golang based, SSH Bruteforcer, Installs Rootkit [6]

Axis-R: A rewrite of the QBot style botnet, written in C, modular.

Various bitcoin miner botnets (Trinity, etc)

Various botnets now targeting FPGAs and more exotic architectures [7]

Botnet Activity Growth

Similarly to QBOT's growth and usage, Mirai variants began popping up all over once the source code was leaked online.

This spawned a large marketplace for people to sell "spots" on the botnet, as well as affiliate programs and incentives for supporting the botnet's growth.

The popularity of "booting", or knocking people offline, spread rapidly in the gaming community and beyond, as a method of settling disputes.

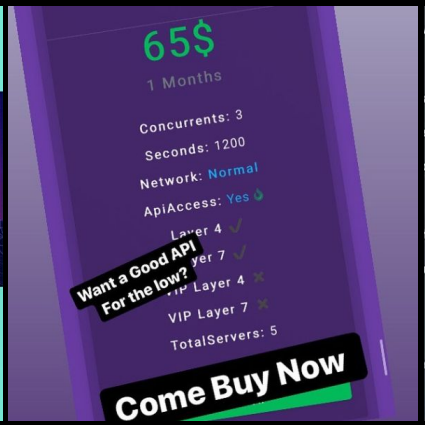
Botnet Scene

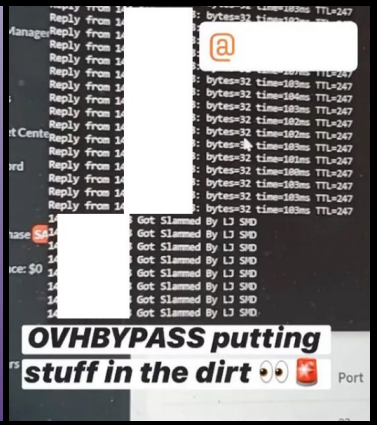
The scene at a glance

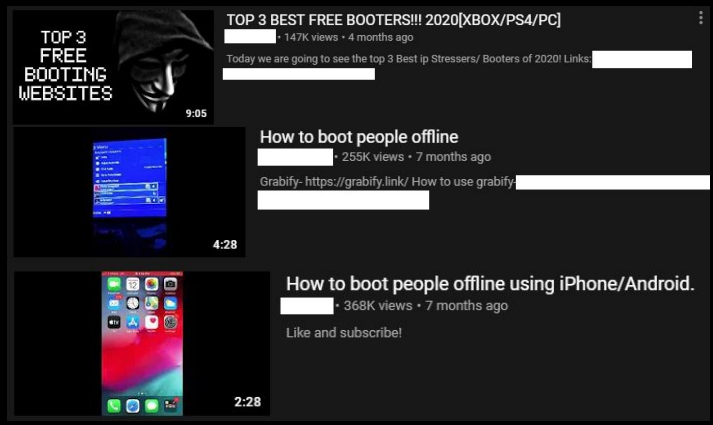
Entire communities dedicated to specific botnets and groups, usually communicating via Discord, Forums, or IRC.

Advertising is done on virtually every social media platform.

“Booter” time is sold to customers for DDoS, either through a web panel (web stresser) or through a telnet interface on the C2.







Distribution: Sources

Pictured: Botnet author after a long day on exploit-db

Botnet sources are usually distributed in archive files (zip, rar, tar.gz) and are sold for around $50-$300 USD.

Authors typically change very little of the actual base source code, usually just changing some ASCII art, variable names, and sometimes adding new exploits.

Sometimes exploits themselves are sold, but many of them are ripped straight from exploit-db (and are frequently backdoored).

When authors scam, rip-off, or are deemed untrustworthy by other users, their source codes are leaked (sometimes in grandiose fashion).

Distribution: “Spots”

The primary source of revenue is selling botnet “spots”, which are user accounts with a set number of “credits” that can be used to launch DDoS attacks against different IPs.

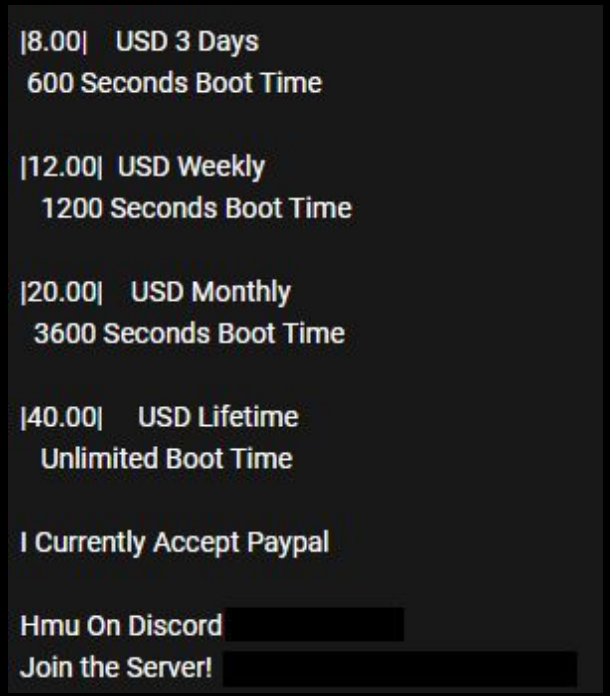
These spots are typically sold in plans, such as:

- Weekly
- Monthly
- Lifetime

Lifetime typically means for the duration of the botnet’s lifetime, which may or may not last beyond the other two plans.

More enterprising operators may have a full on UI, known as a webstresser, which can be accessed via a browser, rather than through telnet.

Some people act as resellers, and get a cut of the sales of botnet spots / booter time.



Who runs a botnet?

IoT botnet operators typically (based purely on observation):

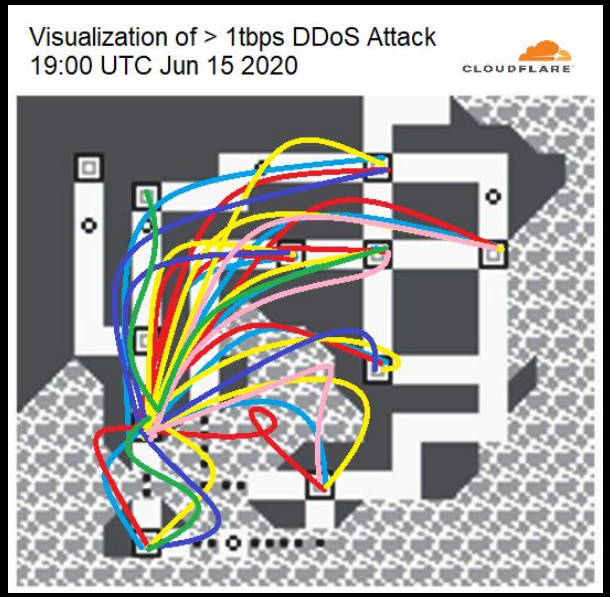
- Are young, somewhat experienced with computers
- Learn through YouTube and text files
- Have no clue what they're doing

More sophisticated operators might:

- Have a webstresser or an API for their botnets
- Use cryptocurrency for transactions
- In some cases, use their botnets for additional purposes like proxying traffic and selling that

Pictured: Botnet operator putting the finishing touches on their new setup.

Why Run An IoT Botnet?



Money: Operators can likely earn decent money for renting botnet spots.

Attention: Botnet operators typically seek attention for their botnets, which can come back to bite them.

Supply & Demand: "Booting" has been steadily increasing in popularity since the rise of IoT botnets.

Revenge: Botnet operators can get revenge for supposed wrongdoing of others.

Inspired By Past Attacks: There have been so many with global impact, people want a piece of the pie.

Also, it's *easy AF*.

Architecture

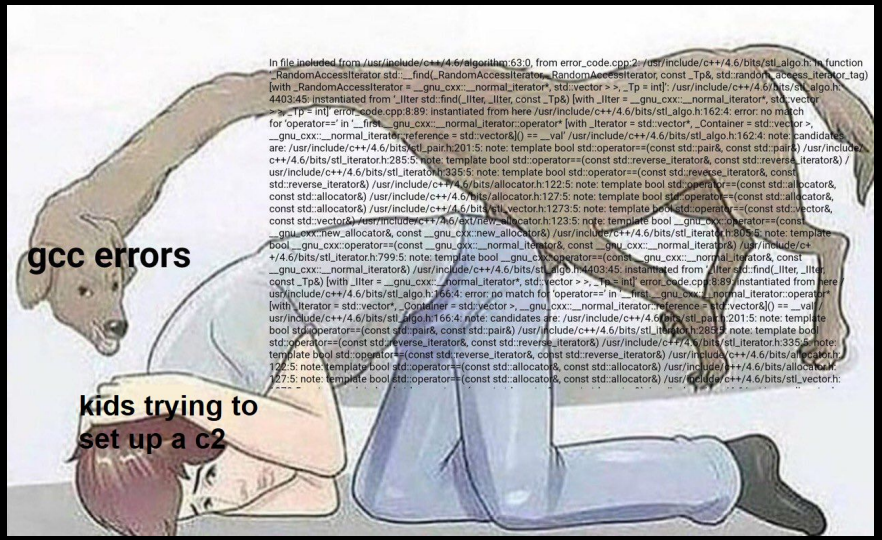
Architecture over Time

Early botnets used standalone bot files, and C2 files. (QBot/P2P botnets)

Some used older methodology like IRC for Command and Control.

Mirai modernized the C2 with a Golang based server, using MySQL for a backend.

Web stresser front ends also added a bit of abstraction that modernized the approach.



Lifecycle of an IoT Botnet

Setup and Usage

- Person sets up a C2 on a more lax VPS host
- Scans for vuln devices or uses a list of known vuln devices
- Gets a few bots
- Advertises spots on their botnet
- People use it and abuse it

Takedown

- Bots eventually get noticed and their C2 gets taken down

  OR
- Someone else's botnet starts kicking their bots from systems
- Eventually the botnet loses it's power

Then the cycle continues

King of the Hill Game

Botnets are a “King of the Hill” game, very territorial and ephemeral.

Most of the time, everyone who touches the device has root access, with no real way to re-configure the device.

This means that every bot only lasts as long as it can before it is inevitably kicked off.

No real repercussions.

Pictured: Operator watching their bot count drop.

Evasion

```
char *mynameis = "/usr/sbin/dropbear";
strncpy(argv[0],"",strlen(argv[0]));
argv[0] = "/usr/sbin/dropbear";
prctl(PR_SET_NAME, (unsigned long) mynameis, 0, 0, 0);
```

There are varying levels of simplistic evasion coded into bots, but these are typically not to hide from things like AV or firewalls.

Mainly used to evade other botnet authors.

Techniques:

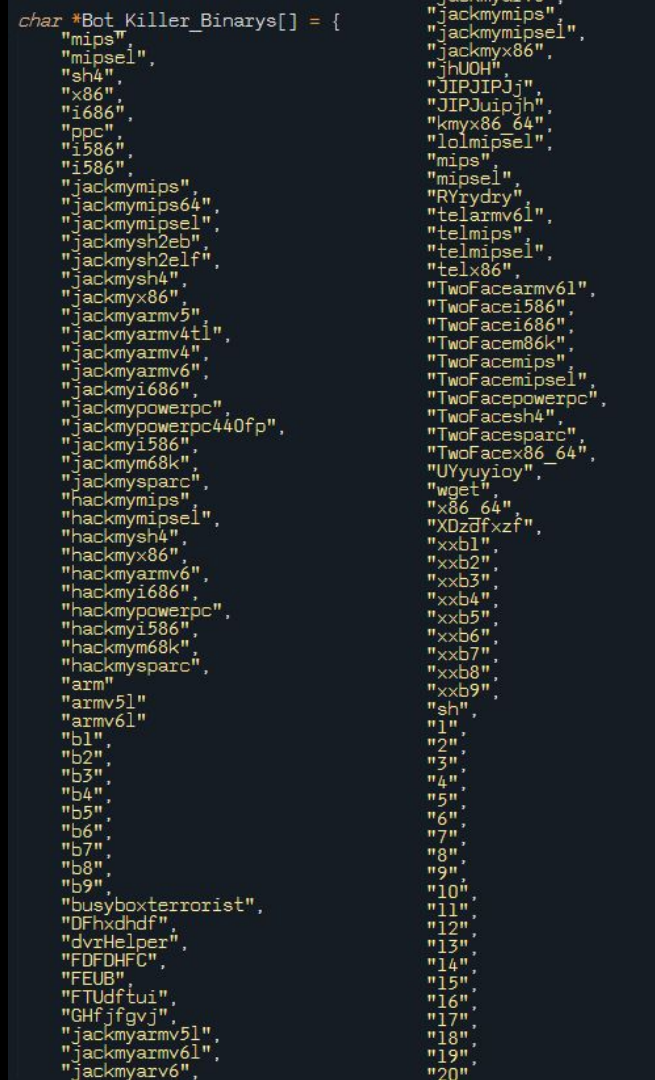
- Process Masking (eg. Pretend to be Dropbear or some other system process)
- Hide in lesser known areas of the file system
- Hiding backup bots in other locations

Bot Killing

Bots will sometimes have hardcoded lists of known bot names, which they use to attempt to remove existing bots on a given system.

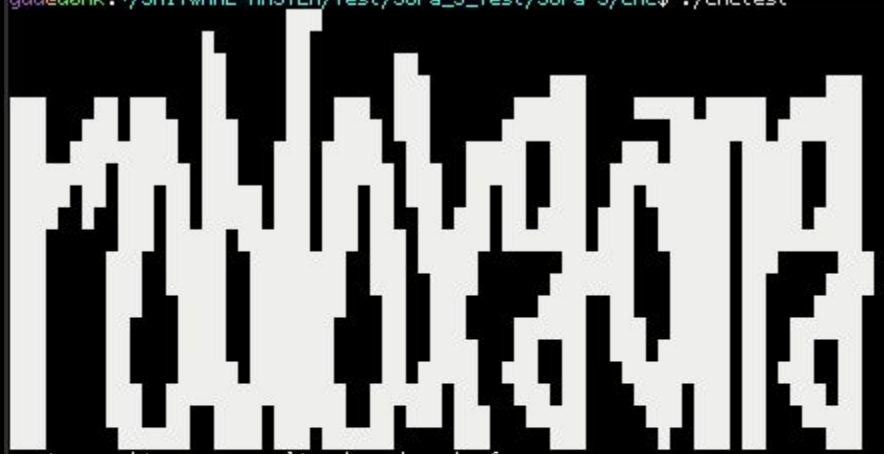
~~Some~~ ~~most~~ *nearly-all* bots and c2s have silly vulnerabilities that make them really easy to knock offline.

These techniques are largely under utilized.



Non-Live Demo: C2 Killer

```
yuu@d0nk:~/SHITWARE-MASTER/Test/Sora_3_Test/Sora 3/cnc$ ./cnctest
```



```
panic: runtime error: slice bounds out of range

goroutine 34 [running]:
main.(*Admin).ReadLine(0xc000108000, 0xc000116000, 0x22, 0x22, 0x0, 0x6f15c0)
        /home/yuu/SHITWARE-MASTER/Test/Sora_3_Test/Sora 3/cnc/admin.go:339 +0x5dc
main.(*Admin).Handle(0xc000108000)
        /home/yuu/SHITWARE-MASTER/Test/Sora_3_Test/Sora 3/cnc/admin.go:34 +0x1d5
main.initialHandler(0x6f6dc0, 0xc0000f8000)
        /home/yuu/SHITWARE-MASTER/Test/Sora_3_Test/Sora 3/cnc/main.go:68 +0x447
created by main.main
        /home/yuu/SHITWARE-MASTER/Test/Sora_3_Test/Sora 3/cnc/main.go:30 +0x143
yuu@d0nk:~/SHITWARE-MASTER/Test/Sora_3_Test/Sora 3/cnc$
```

```
yuu@d0nk:~/SHITWARE-MASTER/Test/Sora_3_Test$ python miraikill.py
see ya <3
yuu@d0nk:~/SHITWARE-MASTER/Test/Sora_3_Test$ cat miraikill.py
import socket

# Mirai C2 Admin Port DOS

rip = "127.0.0.1"
rport = 42069

print "see ya <3"
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect = s.connect((rip,rport))
s.send('\x00\x00'+"A"*2600+'\r\n')
s.close()
yuu@d0nk:~/SHITWARE-MASTER/Test/Sora_3_Test$
```

Non-Live Demo: Inhale

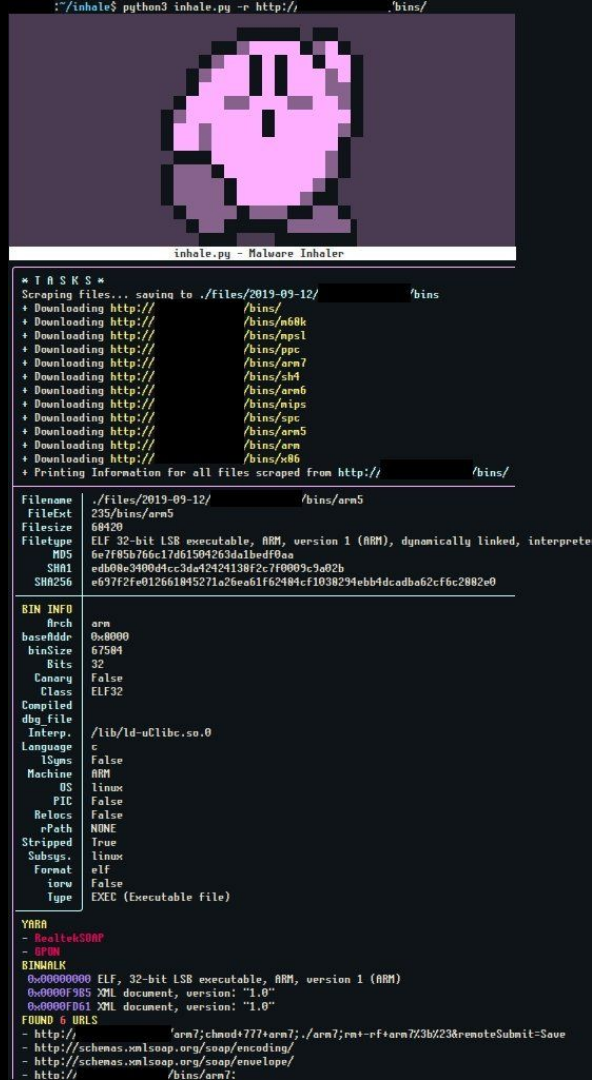
Created this tool to help track botnet binaries

Used for fast static analysis and classification

Updates coming soon (API, telfhash, key extraction)

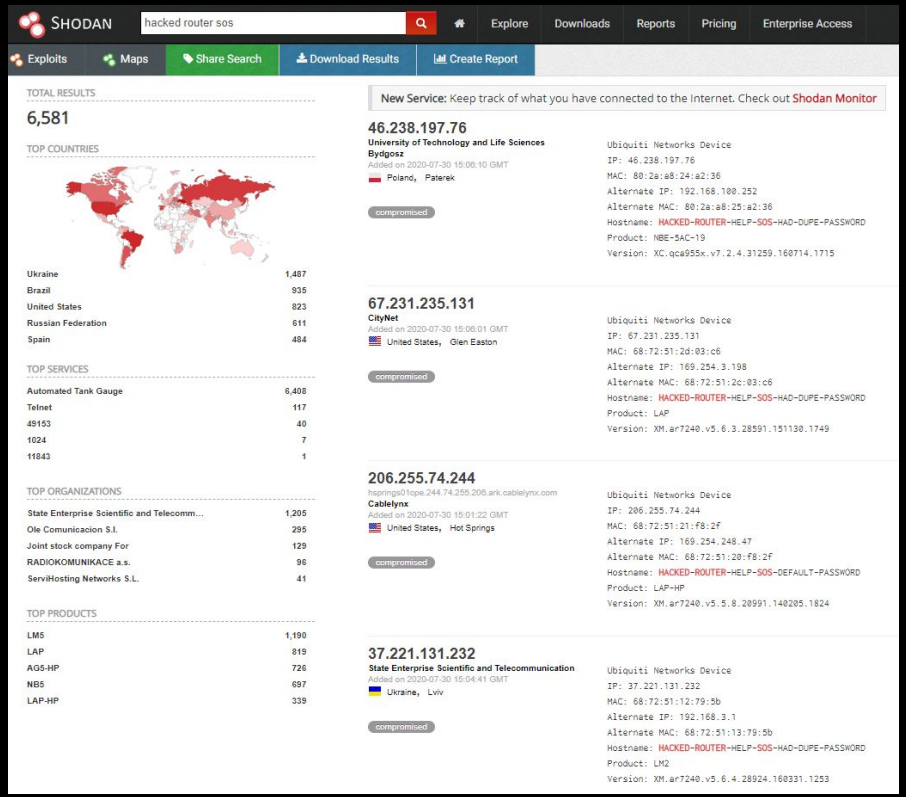
https://github.com/netspooky/inhale

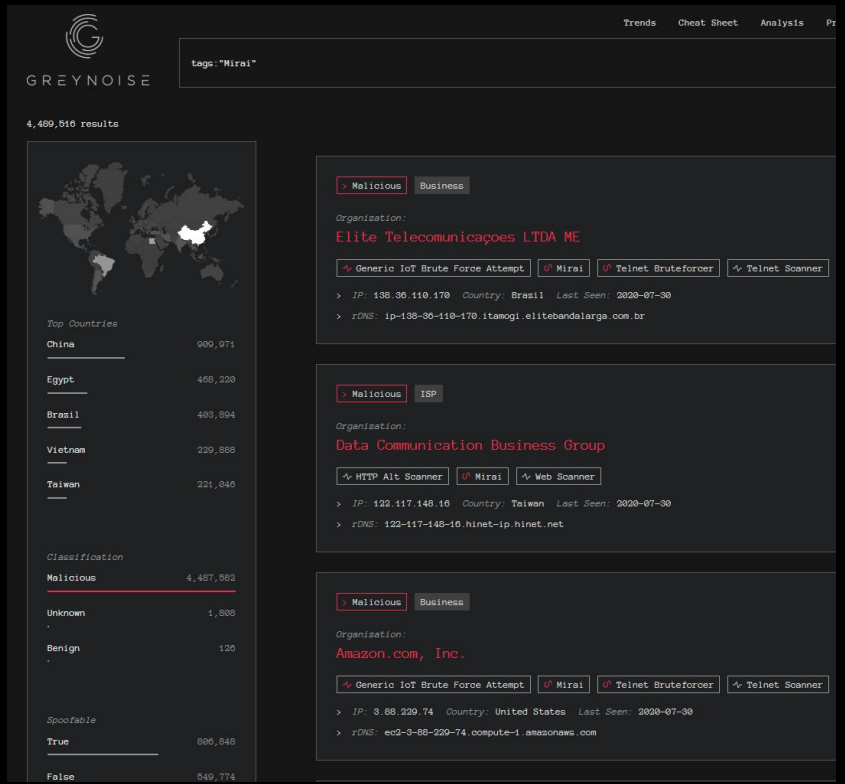
https://github.com/threatland/tl-bots



Vulnerabilities

Peering Into The Void





What types of vulns are exploited?

- Weak Auth / Auth Bypass
- Command Injection
- Common Exploits in specific services and libraries (Realtek uPNP, GoAhead, ThinkPHP)
- More Rare: Binary exploits

Other vectors include previously compromised devices, eg. scanning with recovered creds for bots

Most Targeted Devices

Looked at vulns that were leveraged by various botnet sources.

Not Included: Telnet/SSH Bruteforce, Non-IoT Vulns

Many vulns aren't even properly tracked, eg CVE or vendor acknowledgement.

When a new exploit comes out, bot scanners start up shortly after and attempt to use it to load bots.

Unofficial Name	CVE	Vuln Class
AVTech	NONE	Default Creds, Unauthed Command Injection in URL
BCMLoad	NONE	Default Creds
CCTV-DVR	NONE	Command injection in URL `language/Swedish`
Dasan.GPON	CVE-2018-10561:CVE-2018-10562	Auth bypass/command injection
Dasan.H640X	CVE-2017-18046	Buffer Overflow
DLink.Command	NONE	Unauthenticated command interface
DLink.DCS-7410	CVE-2013-1599:CVE-2013-1603	Command injection in URL
DLink.uPNP	CVE-2014-8361	Command Injection telnetd uPNP SOAP soap.cgi "NewInternalClient"
EnGenius	NONE	RCE via usbinteract.cgi
GoAhead	CVE-2017-8225	Pre-Auth Info Leak, Auth RCE, Unauth RCE
Grandstream UCM62XX	NONE	Command injection
Huawei.HG532	CVE-2017-17215	Default Creds, Command injection in SOAP /ctrlt/DeviceUpgrade_1
JAWS/MVPower DVR	NONE	Command injection in URL, possible backdoor
libupnp_ssdp	CVE-2012-5958	RCE
Linksys.Eseries	NONE	Command injection in Headers to tmUnblock.cgi
Mikrotik.SSH	NONE	Default SSH Creds
Netgear.DGN1000	NONE	Command injection in URL
Netgear.R7000	CVE-2016-6277	Command injection in URL
Netis	NONE	Hardcoded pw - Buffer Overflow on UDP port
R4IX	CVE-2017-8224	Default creds for FTP, Authenticated RCE
Realtek.uPNP	CVE-2014-8361	Command Injection on uPNP SOAP picsdesc.xml "NewInternalClient"
ThinkPHP	CVE-2018-20062	Command injection in URL
UPNP.HNAP	NONE	Command Injection on uPNP SOAP /HNAP1 SOAPAction Header
Vacron	NONE	Command injection
ZyXEL.D1000	NONE	Command Injection on uPNP SOAP /UD/act?1 "NewNTPServer1"
Zyxel.SecuManager	CVE-2020-15312:CVE-2020-15348	RCE

Infection Spillover

IoT Malware families, particularly Mirai, run on the most diverse array of architectures. Due to extensive cross compiling.

This means that they can also infect other hosts that aren't IoT just the same.

Commonly exploited vulns include Drupalgeddon, Apache Struts, Android ADB, and various database exploits.

Why are these devices so easy to exploit?

Vulnerable Libraries / Software

Easy to guess default passwords

Devices by default listening on the open internet

Giant lists of vuln devices are passed around online

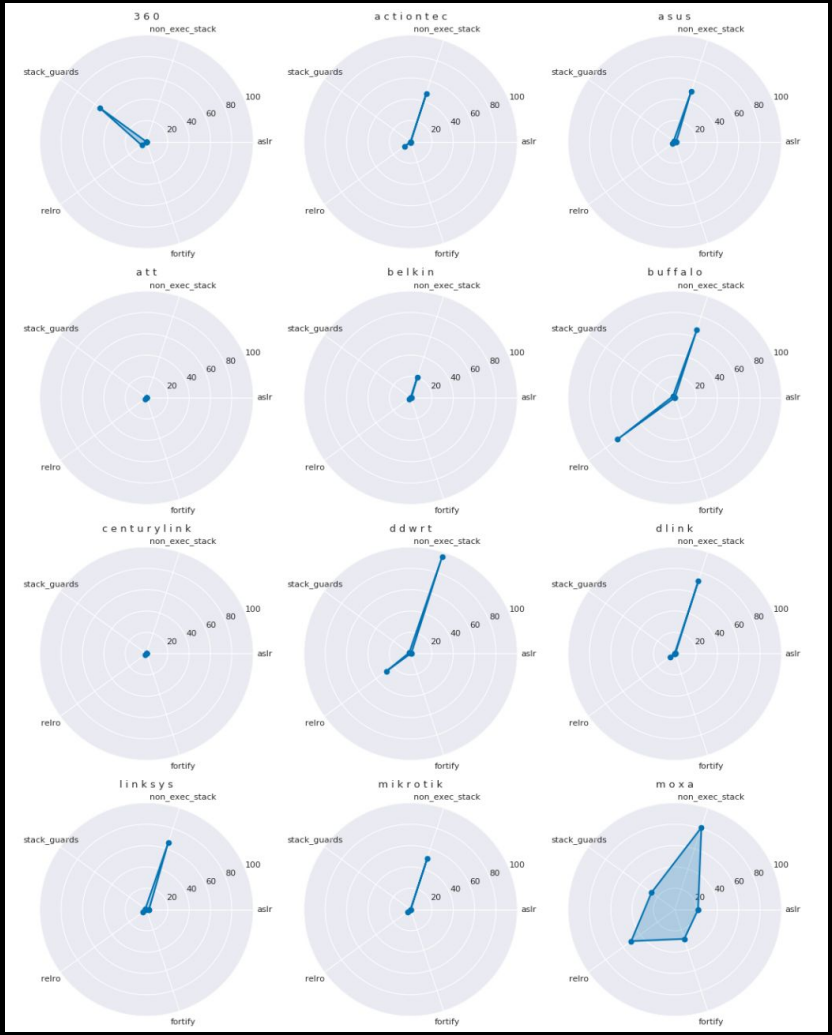
Insufficient or non-existent security practices in development

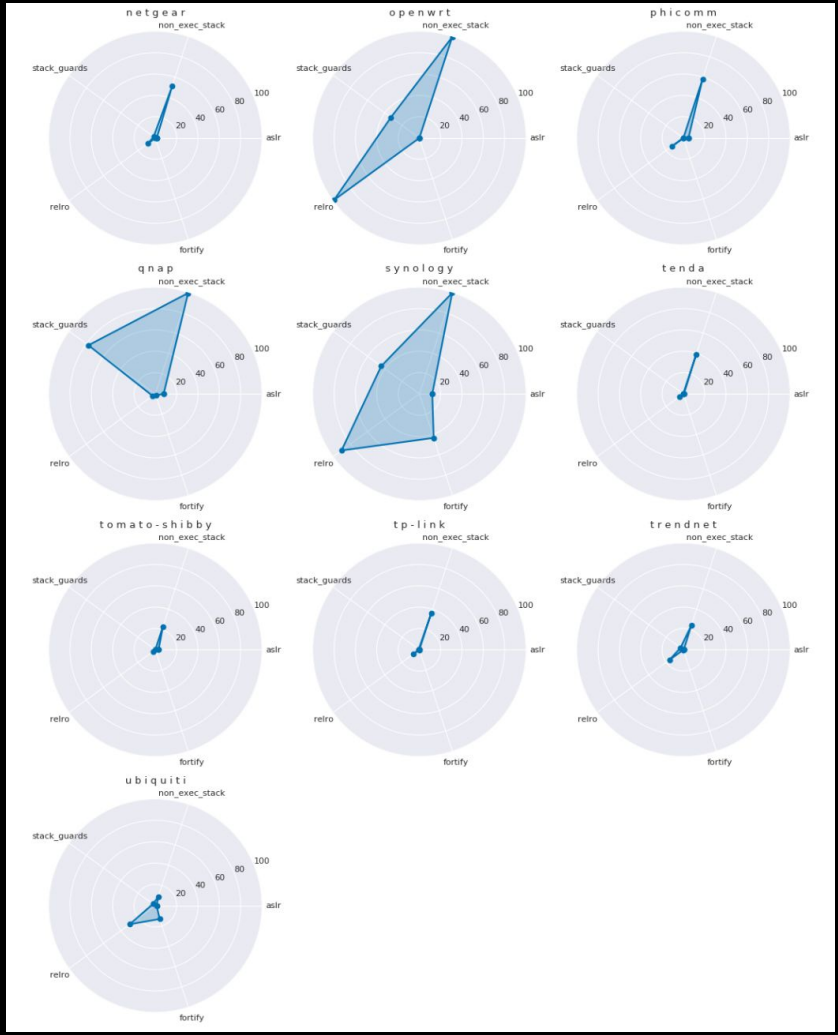
Firmware Vulns

Basic security practices on a binary level aren't being taken by many major vendors.

Regression Analysis shows that firmware overall is not improving from a security perspective, based on data from 2003-2018.

Analysis of Firmware Vulns by CITL [8]


360
actiontec
asus
att
belkin
buffalo
centurylink
ddwrt
dlink
linksys
mikrotik
moxa
non_exec_stack
aslr
fortify
relro
stack_guards
20
40
60
80
100

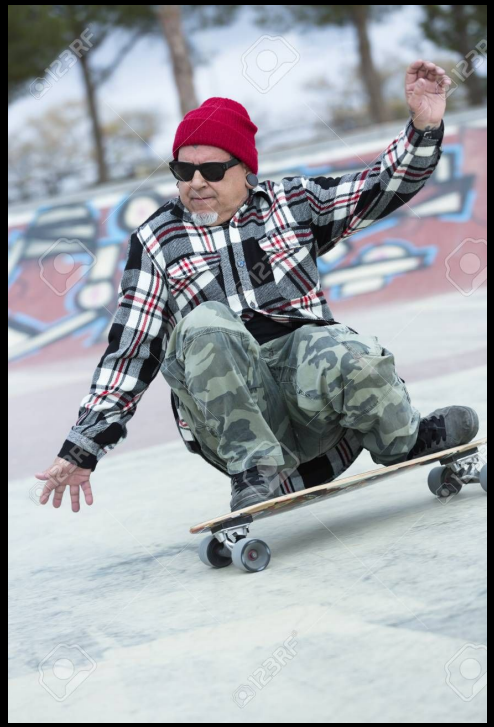

netgear
openwrt
phicomm
qnap
synology
tenda
tomato-shibby
tp-link
trendnet
ubiquiti
non_exec_stack
aslr
fortify
relro
stack_guards
20
40
60
80
100

Why is firmware so difficult to maintain?

- Rearchitecting cost
- Locked into vendor contracts, third party libraries and dependencies
- Unsupported chips and hardware, outdated toolchains
- Hardware constraints
- The need to maintain backward compatibility
- Lack of a dependable update pipeline for end users and devices
- Poor communication channels with end users
- Vendors might not have any security or bug reporting mechanisms in place
- Lack of modern security measures like secure boot, binary hardening, and code signing

Pictured: Firmware dev angry at chip vendor's documentation

Why do we see some of the older stuff still working?



There are still Qbots out there, and they still work! [9]

This phenomenon is rarer in other classes of malware because there's really no patch, so each time there is a new vuln, all it does is add more devices to the pool.

Moving Forward

What can we do?

Only can fix by introducing better firmware practices by meeting developers where they're at

Vendors: Invest in developer training, establish best practices and create security testing pipeline per commit

Encourage researchers to find vulns and disclose them properly

Can mitigate existing vulns by encouraging safer use of IoT devices

Establishing Best Practices

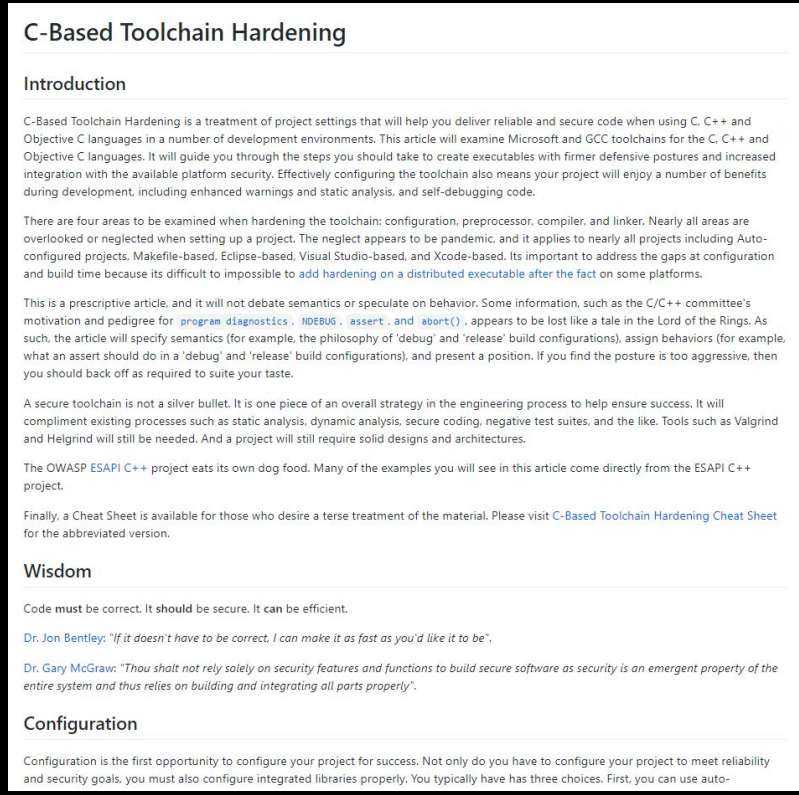
Auditing your development cycle

Depends on what you're building

OWASP/Cheatsheets https://github.com/OWASP/CheatSheetSeries

CIS Benchmarks https://www.cisecurity.org/cis-benchmarks/

Consultants



C-Based Toolchain Hardening

Introduction

C-Based Toolchain Hardening is a treatment of project settings that will help you deliver reliable and secure code when using C, C++ and Objective C languages in a number of development environments. This article will examine Microsoft and GCC toolchains for the C, C++ and Objective C languages. It will guide you through the steps you should take to create executables with firmer defensive postures and increased integration with the available platform security. Effectively configuring the toolchain also means your project will enjoy a number of benefits during development, including enhanced warnings and static analysis, and self-debugging code.

There are four areas to be examined when hardening the toolchain: configuration, preprocessor, compiler, and linker. Nearly all areas are overlooked or neglected when setting up a project. The neglect appears to be pandemic, and it applies to nearly all projects including Auto-configured projects, Makefile-based, Eclipse-based, Visual Studio-based, and Xcode-based. Its important to address the gaps at configuration and build time because its difficult to impossible to add hardening on a distributed executable after the fact on some platforms.

This is a prescriptive article, and it will not debate semantics or speculate on behavior. Some information, such as the C/C++ committee's motivation and pedigree for `program diagnostics`, `NDEBUG`, `assert`, and `abort()`, appears to be lost like a tale in the Lord of the Rings. As such, the article will specify semantics (for example, the philosophy of 'debug' and 'release' build configurations), assign behaviors (for example, what an assert should do in a 'debug' and 'release' build configurations), and present a position. If you find the posture is too aggressive, then you should back off as required to suite your taste.

A secure toolchain is not a silver bullet. It is one piece of an overall strategy in the engineering process to help ensure success. It will compliment existing processes such as static analysis, dynamic analysis, secure coding, negative test suites, and the like. Tools such as Valgrind and Helgrind will still be needed. And a project will still require solid designs and architectures.

The OWASP ESAPI C++ project eats its own dog food. Many of the examples you will see in this article come directly from the ESAPI C++ project.

Finally, a Cheat Sheet is available for those who desire a terse treatment of the material. Please visit C-Based Toolchain Hardening Cheat Sheet for the abbreviated version.

Wisdom

Code **must** be correct. It **should** be secure. It **can** be efficient.

Dr. Jon Bentley: "*If it doesn't have to be correct, I can make it as fast as you'd like it to be*".

Dr. Gary McGraw: "*Thou shalt not rely solely on security features and functions to build secure software as security is an emergent property of the entire system and thus relies on building and integrating all parts properly*".

Configuration

Configuration is the first opportunity to configure your project for success. Not only do you have to configure your project to meet reliability and security goals, you must also configure integrated libraries properly. You typically have has three choices. First, you can use auto-

Vuln Disclosure

Pictured: What hackers really want.

Allow researchers to disclose vulns! Don't sue or ignore! https://disclose.io

Establish a security contact and listen to emails. https://securitytxt.org

Work with researchers who bring issues up, they want to help you.

Have some open channel with your customers to get word out about vulns.

These are elements of a Vulnerability Disclosure Program

Community Suggestions

- Automatic Updates / Better update pipeline
- Vuln disclosure program "Don't sue people who report bugs!"
- Regular audits and code review process
- Have security connections with ODM/OEMs
- *"Make security a named person's problem"*, allocate security budget
- *"Ask yourself if your device truly needs to be on the internet"*
- No default/hardcoded/backdoor credentials
- Put security into your user journey
- Default settings should be sane / with security in mind
- Do risk assessments and threat models for the user, the device, and your company
- Products should have a fair "shelf life" before EOL
- Use modern tool chains to build firmware and applications
- Make use of hardware / chip level security features
- Minimize attack surface
- Sign FW updates
- Follow best practices, Don't reinvent the wheel

https://twitter.com/netspooky/status/1289606589121359872

Final Thoughts

- Make it less easy for people to run botnets.
- The supply is already there, and the demand is great
- Botnet authors are getting smarter, people are using the messy botnet landscape to take control
- New architectures and devices are *always* being targeted, if you don't act soon, your new product will be DOA

Note: Q&A will be done in the Defcon Discord https://discord.gg/defcon or you can talk to me on Twitter @netspooky

Shoutouts

Safari Zone Crew / Threatland / ThugCrowd

Hermit

Andrew Morris / GreyNoise

Mudge/CITL

Ilya - Check out his IoT Village talk on emulating IoT devices and malware with Docker and Qemu https://www.youtube.com/watch?v=ALnOhUxNszI

Oxdade for the theme song

Citations

[1] https://blog.trendmicro.com/trendlabs-security-intelligence/shellshock-vulnerability-downloads-kaiten-source-code/
[2] https://blog.cloudflare.com/inside-mirai-the-infamous-iot-botnet-a-retrospective-analysis/
[3] https://github.com/threatland/TL-BOTS/tree/master/TL.MIRAI/MIRAI.OriginalSource
[4] https://blog.malwaremustdie.org/2020/02/mmd-0065-2021-linuxmirai-fbot-re.html
[5] https://www.bleepingcomputer.com/news/security/brickerbot-author-retires-claiming-to-have-bricked-over-10-million-iot-devices/
[6] https://www.intezer.com/blog/research/kaiji-new-chinese-linux-malware-turning-to-golang/
[7] https://n0.lol/a/miraiexotic.html
[8] https://cyber-itl.org/2019/08/26/iot-data-writeup.html
[9] https://imgur.com/a/CtHlmBE