Virtualizing IoT
with Code Coverage Guided Fuzzing

HITB2018DXB, November 2018

NGUYEN Anh Quynh, aquynh -at- gmail.com
KaiJern LAU, kj -at- theshepherd.io

About NGUYEN Anh Quynh

- Nanyang Technological University, Singapore
- PhD in Computer Science
- Operating System, Virtual Machine, Binary analysis, etc
- Usenix, ACM, IEEE, LNCS, etc
- Blackhat USA/EU/Asia, DEFCON, Recon, HackInTheBox, Syscan, etc
- Capstone disassembler: http://capstone-engine.org
- Unicorn emulator: http://unicorn-engine.org
- Keystone assembler: http://keystone-engine.org

About kaijern.xwings.L

Founder

Stays in the lab 24/7 by hoping making the world a better place

- IoT Research
- Blockchain Research
- Fun Security Research

Badge Maker

HACKERSBADGE.COM

Electronic fan boy, making toys from hacker to hacker

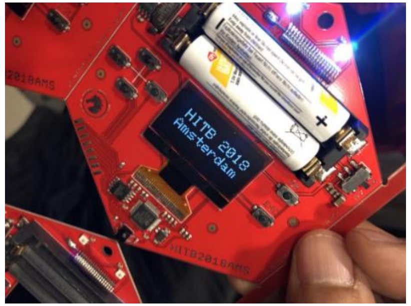
- Reversing Binary
- Reversing IoT Devices
- Part Time CtF player

Broker

HITB

Crew since 2008, from Kuala Lumpur till now AMS, SG, BEIJING and DXB

- 2006 (ctf) till end of time
- Core Crew
- Review Board



- 2005, HITB CTF, Malaysia, First Place /w 20+ Intl. Team
- 2010, Hack In The Box, Malaysia, Speaker
- 2012, Codegate, Korean, Speaker
- 2015, VXRL, Hong Kong, Speaker
- 2015, HITCON Pre Qual, Taiwan, Top 10 /w 4K+ Intl. Team
- 2016, Codegate PreQual, Korean, Top 5 /w 3K+ Intl. Team
- 2016, Qcon, Beijing, Speaker
- 2016, Kcon, Beijing, Speaker
- 2016, Intl. Antivirus Conference, Tianjin, Speaker
- 2017, Kcon, Beijing, Trainer
- 2017, DC852, Hong Kong, Speaker
- 2018, KCON, Beijing, Trainer
- 2018, DC010, Beijing, Speaker
- 2018, Brucon, Brussel, Speaker
- 2018, H2HC, San Paolo, Brazil, Speaker
- 2018, HITB, Beijing/Dubai, Speaker
- 2018, beVX, Hong Kong, Speaker

- MacOS SMC, Buffer Overflow, suid
- GDB, PE File Parser Buffer Overflow
- Metasploit Module, Snort Back Oriffice
- Linux ASLR bypass, Return to EDX

Agenda

Coverage Guided Fuzzer vs Embedded Systems

Emulating Firmware

Skorpio Dynamic Binary Instrumentation

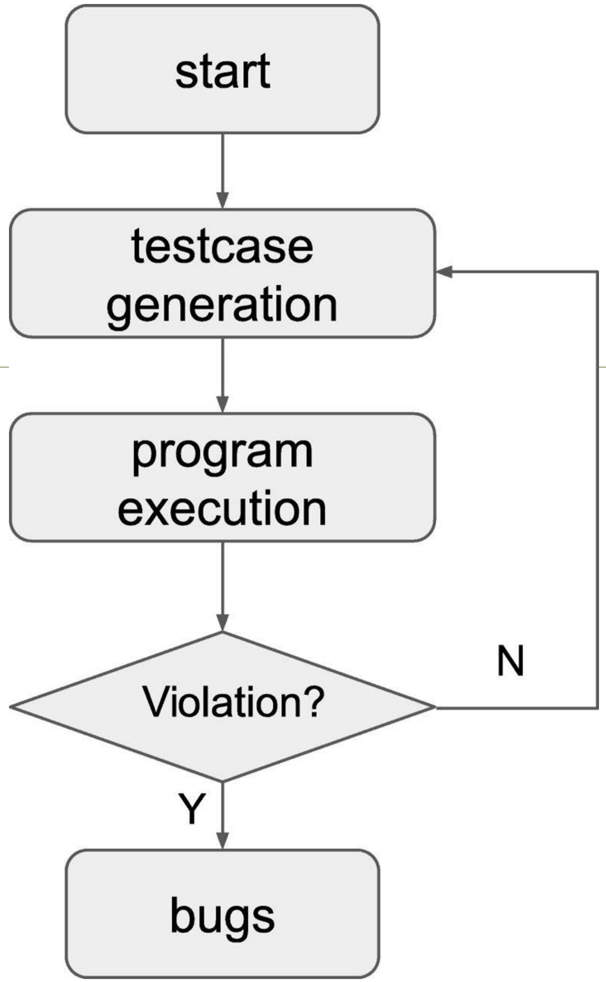
Guided Fuzzer for Embedded

DEMO

Conclusions

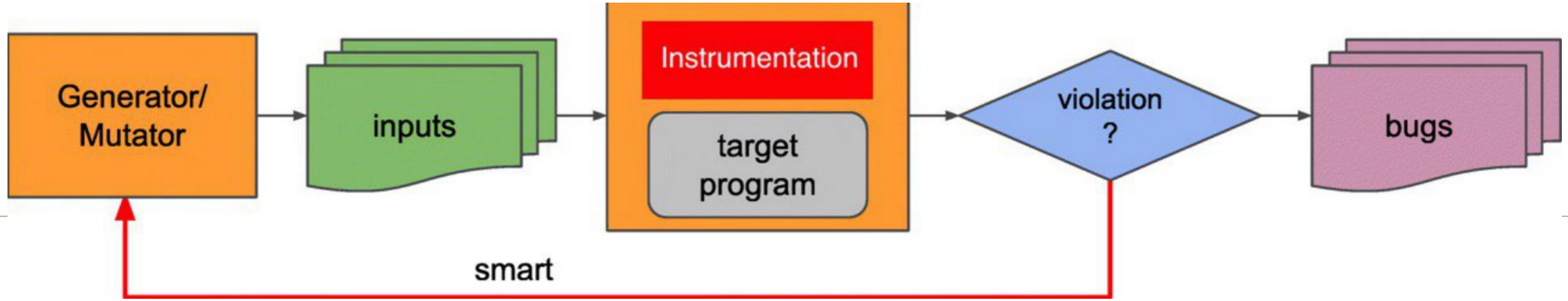
Secret Menu

Fuzzing



- Automated software testing technique to find bugs
  - Feed craft input data to the program under test
  - Monitor for errors like crash/hang/memory leaking
  - Focus more on exploitable errors like memory corruption, info leaking
- Maximize code coverage to find bugs
- Blackbox fuzzing
- Whitebox fuzzing
- Graybox fuzzing, or Coverage Guided Fuzzing

Coverage-guided Fuzzer



- Instrument target binary to collect coverage info
- Mutate the input to maximize the coverage
- Repeat above steps to find bugs
    - Proved to be very effective
        - Easier to use/setup & found a lot of bugs
    - Trending in fuzzing technology
        - American Fuzzy Lop (AFL) really changed the game

Guided Fuzzer for Embedded



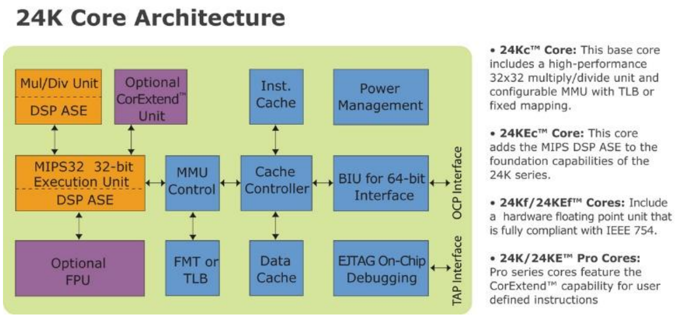


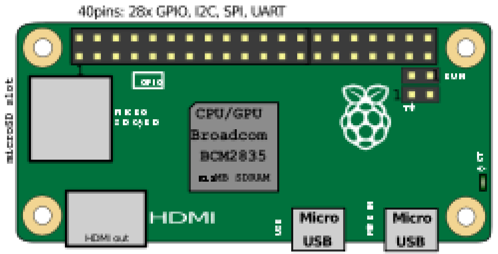


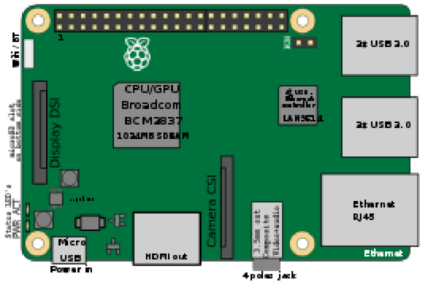


- Guided fuzzer was introduced for powerful PC systems
- Bring over to embedded world?
  - No support for introducing new tools
  - Not open source
  - Lack support for embedded hardware

Issues







Restricted System	Closed System	Lack Support for Embedded

- Without built-in shell access for user interaction
- Without developement facilities required for building new tools
  - Compiler
  - Debugger
  - Analysis tools

- Binary only - without source code
  - Existing guided fuzzers rely on source code available
    - Source code is needed for branch instrumentation to feedback fuzzing progress
    - Emulation such as QEMU mode support in AFL is slow & limited in capability
    - Same issue for other tools based on Dynamic Binary Instrumentation

- Most fuzzers are built for X86 only
  - Embedded systems based on Arm, Arm64, Mips, PPC
- Existing DBIs are poor for non-X86 CPU
  - Pin: Intel only
  - DynamoRio: experimental support for Arm

Agenda

Coverage Guided Fuzzer vs Embedded Systems

Emulating Firmware

Skorpio Dynamic Binary Instrumentation

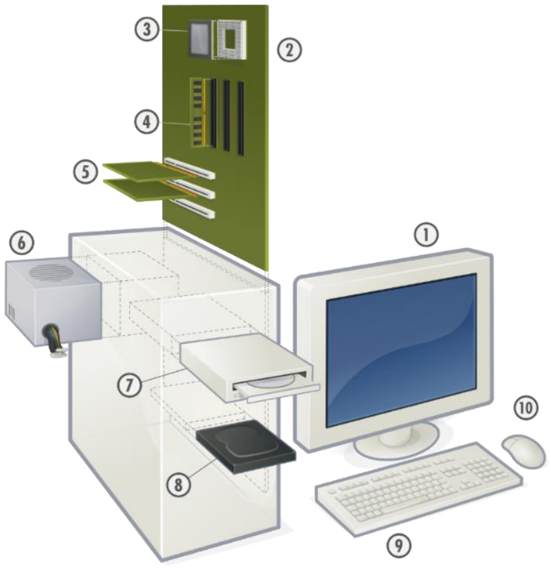
Guided Fuzzer for Embedded

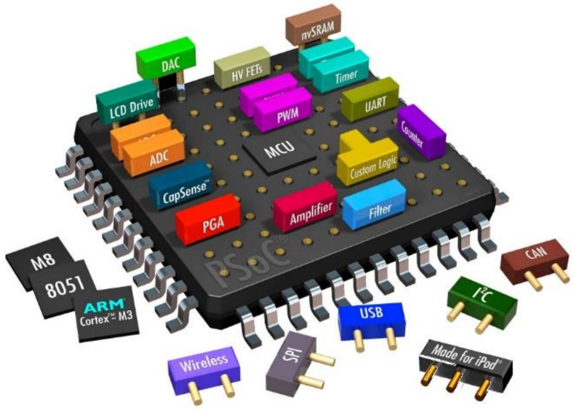
DEMO

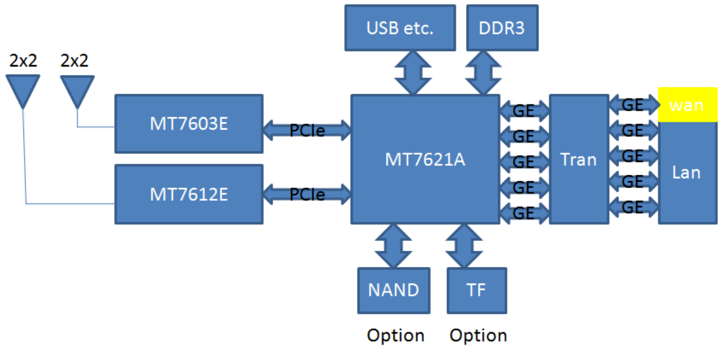
Conclusions

Secret Menu

The SoC



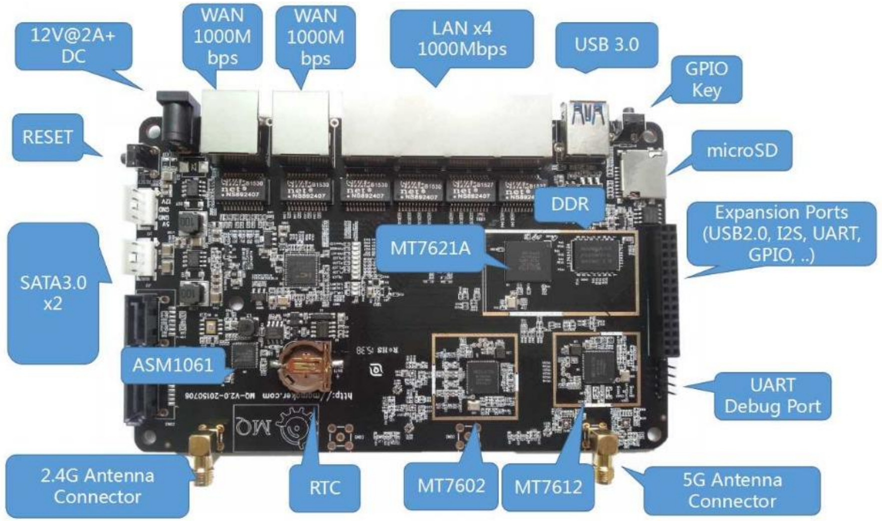


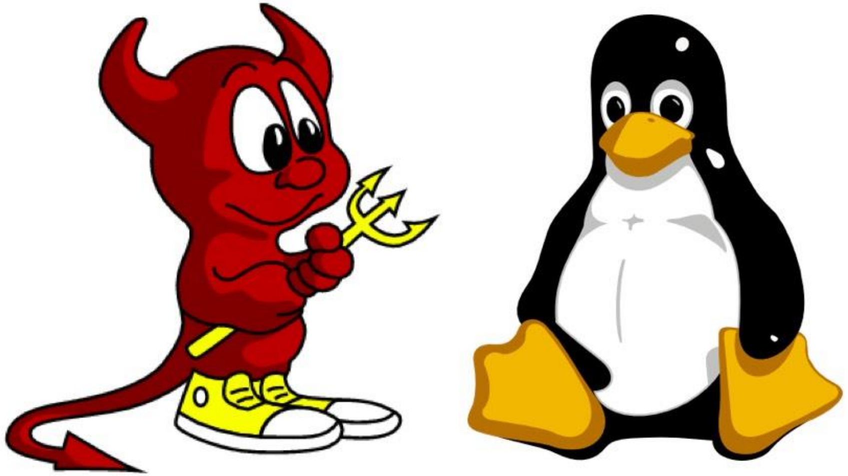


- Scale Down from PC
- System on Chip
- A chip with all the PCI-e slot and card in it

- Pinout to different parts
- Wifi, Lan, Bluetooth and etc
- Low power device

Requirement





Hardware + GNU Command

also

love hardware and not only hardware hacking

Once you cross over, there are things in the darkness that can keep your heart from feeling the light again

Lets Get Started

Device Limited Bug

Netgear : Security Vulnerabilities

CVSS Scores Greater Than: 0 1 2 3 4 5 6 7 8 9

Sort Results By : CVE Number Descending CVE Number Ascending CVSS Score Descending Number Of Exploits Descending

Total number of vulnerabilities : **75** Page : 1 (This Page) 2

Copy Results Download Results

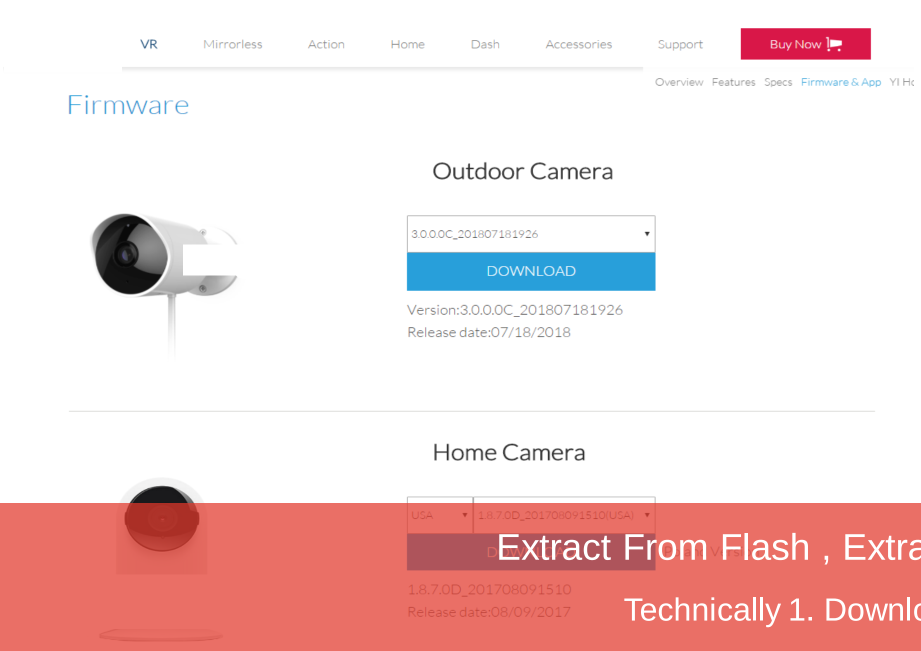
#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained Access Level	Access	Complexity	Authentication	Conf.	Integ.	Avail.
1	CVE-2017-6862	119		Exec Code Overflow Bypass	2017-05-26	2017-07-17	7.5	None	Remote	Low	Not required	Partial	Partial	Partial
NETGEAR WNR2000v3 devices before 1.1.2.14, WNR2000v4 devices before 1.0.0.66, and WNR2000v5 devices before 1.0.0.42 allow authentication bypass and remote code execution via a buffer overflow that uses a parameter in the administration webapp. The NETGEAR ID is PSV-2016-0261.														
2	CVE-2017-6366	352		Exec Code CSRF	2017-03-15	2017-03-29	6.8	None	Remote	Medium	Not required	Partial	Partial	Partial
Cross-site request forgery (CSRF) vulnerability in NETGEAR DGN2200 routers with firmware 10.0.0.20 through 10.0.0.50 allows remote attackers to hijack the authentication of users for requests that perform DNS lookups via the host_name parameter to dnslookup.cgi. NOTE: this issue can be combined with CVE-2017-6334 to execute arbitrary code remotely.														
3	CVE-2017-6334	264		Exec Code	2017-03-05	2017-08-31	9.0	None	Remote	Low	Single system	Complete	Complete	Complete
dnslookup.cgi on NETGEAR DGN2200 devices with firmware through 10.0.0.50 allows remote authenticated users to execute arbitrary OS commands via shell metacharacters in the host_name field of an HTTP POST request, a different vulnerability than CVE-2017-6077.														
4	CVE-2017-6077	78		Exec Code	2017-02-22	2017-03-01	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
ping.cgi on NETGEAR DGN2200 devices with firmware through 10.0.0.50 allows remote authenticated users to execute arbitrary OS commands via shell metacharacters in the ping_IPAddr field of an HTTP POST request.														
5	CVE-2017-5521	200		+Info	2017-01-17	2017-08-31	4.3	None	Remote	Medium	Not required	Partial	None	None
An issue was discovered on NETGEAR R8500, R8300, R7000, R6400, R7300, R7100LG, R6300v2, WNDR3400v3, WNR3500Lv2, R6250, R6700, R6900, and R8000 devices. They are prone to password disclosure via simple crafted requests to the web management server. The bug is exploitable remotely if the remote management option is set, and can also be exploited given access to the router over LAN or WLAN. When trying to access the web panel, a user is asked to authenticate; if the authentication is canceled and password recovery is not enabled, the user is redirected to a page that exposes a password recovery token. If a user supplies the correct token to the page /passwordrecovered.cgi?id=TOKEN (and password recovery is not enabled), they will receive the admin password for the router. If password recovery is set the exploit will fail, as it will ask the user for the recovery questions that were previously set when enabling that feature. This is persistent (even after disabling the recovery option, the exploit will fail) because the router will ask for the security questions.														
6	CVE-2017-2137	264		Bypass	2017-04-28	2017-05-05	4.3	None	Remote	Medium	Not required	None	Partial	None
ProSAFE Plus Configuration Utility prior to 2.3.29 allows remote attackers to bypass access restriction and change configurations of the switch via SOAP requests.														
7	CVE-2016-10176	20		Exec Code	2017-01-29	2017-09-02	7.5	None	Remote	Low	Not required	Partial	Partial	Partial
The NETGEAR WNR2000v5 router allows an administrator to perform sensitive actions by invoking the apply.cgi URL on the web server of the device. This special URL is handled by the embedded web server (uhttpd) and processed accordingly. The web server also contains another URL, apply_noauth.cgi, that allows an unauthenticated user to perform sensitive actions on the device. This functionality can be exploited to change the router settings (such as the answers to the password-recovery questions) and achieve remote code execution.														
8	CVE-2016-10175	200		+Info	2017-01-29	2017-09-02	5.0	None	Remote	Low	Not required	Partial	None	None
The NETGEAR WNR2000v5 router leaks its serial number when performing a request to the /BRS_netgear_success.html URI. This serial number allows a user to obtain the administrator username and password, when used in combination with the CVE-2016-10176 vulnerability that allows resetting the answers to the password-recovery questions.														
9	CVE-2016-10174	119		Exec Code Overflow	2017-01-29	2017-09-02	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
The NETGEAR WNR2000v5 router contains a buffer overflow in the hidden_lang_avi parameter when invoking the URL /apply.cgi?/lang_check.html. This buffer overflow can be exploited by an unauthenticated attacker to achieve remote code execution.														
10	CVE-2016-10116	264			2017-01-04	2017-01-11	9.3	None	Remote	Medium	Not required	Complete	Complete	Complete
NETGEAR Arlo base stations with firmware 1.7.5_6178 and earlier, Arlo Q devices with firmware 1.8.0_5551 and earlier, and Arlo Q Plus devices with firmware 1.8.1_6094 and earlier use a pattern of adjective, noun, and three-digit number for the customized password, which makes it easier for remote attackers to obtain access via a dictionary attack.														
11	CVE-2016-10115	798			2017-01-04	2017-01-11	10.0	None	Remote	Low	Not required	Complete	Complete	Complete
NETGEAR Arlo base stations with firmware 1.7.5_6178 and earlier, Arlo Q devices with firmware 1.8.0_5551 and earlier, and Arlo Q Plus devices with firmware 1.8.1_6094 and earlier have a default password of 12345678, which makes it easier for remote attackers to obtain access after a factory reset or in a factory configuration.														
12	CVE-2016-10106	22		Dir. Trav.	2017-01-03	2017-07-26	4.0	None	Remote	Low	Single system	Partial	None	None
Directory traversal vulnerability in scgi-bin/platform.cgi on NETGEAR FVS336Gv3, FVS318N, FVS318Gv2, and SRX5308 devices with firmware before 4.3.3-8 allows remote authenticated users to read arbitrary files via a .. (dot dot) in the thispage parameter, as demonstrated by reading the /etc/shadow file.														

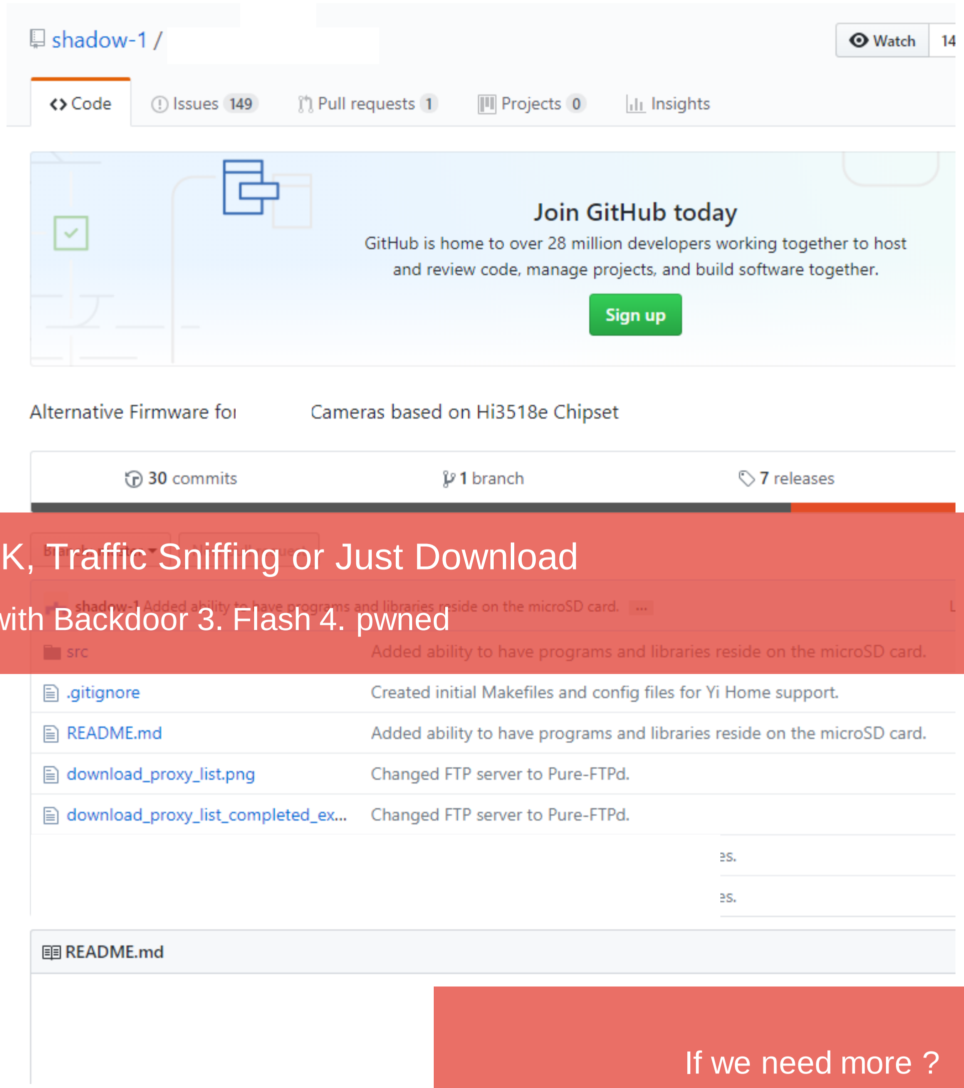
If all_model = one_firmware

In The Beginning:
We Need Firmware

Getting Firmware

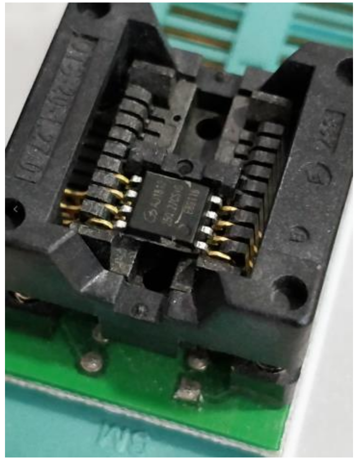
Firmware and Hardware





Extract From Flash , Extract From APK, Traffic Sniffing or Just Download

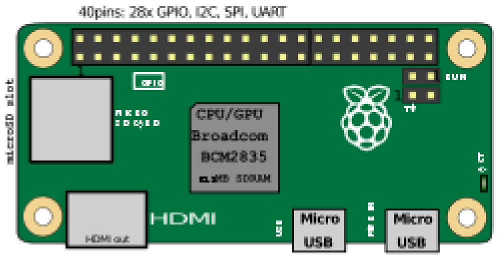
Technically 1. Download 2. Patch with Backdoor 3. Flash 4. pwned

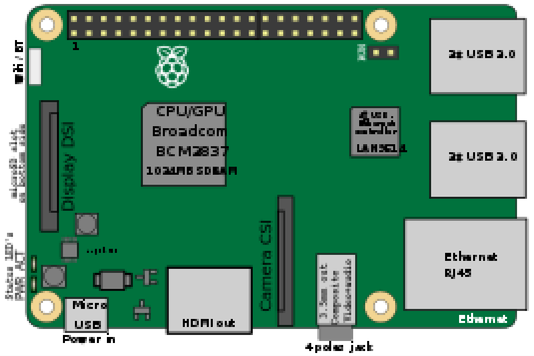


If we need more ?
1. RCE 2. Fuzz

The Easy Way

Complete Kit to Success





MIPS

ARM

AARCH64

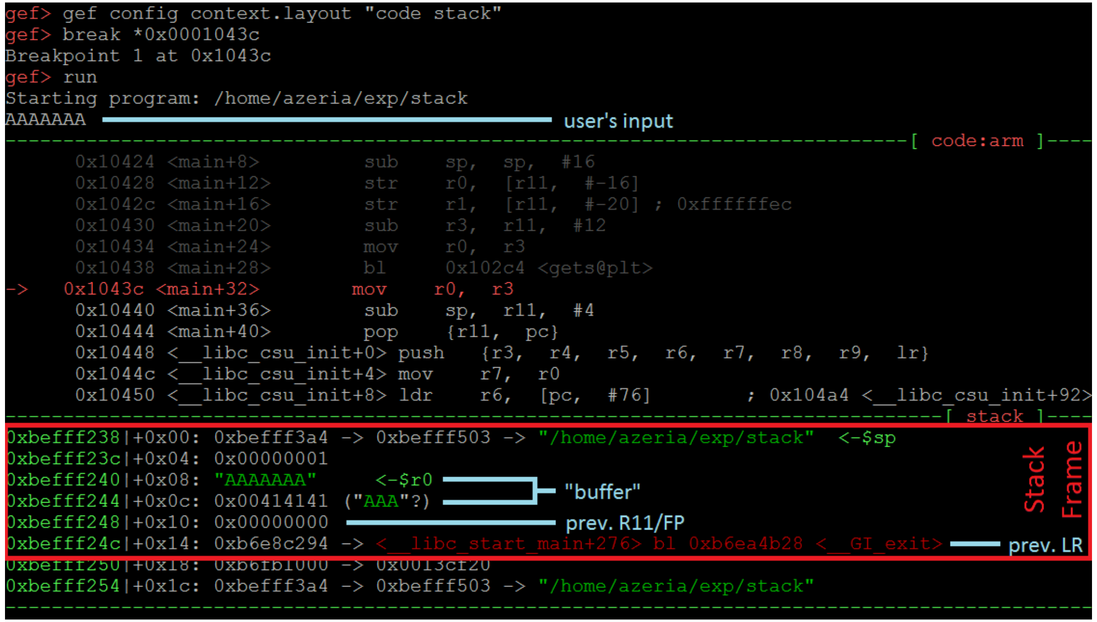
How Many Dev Board

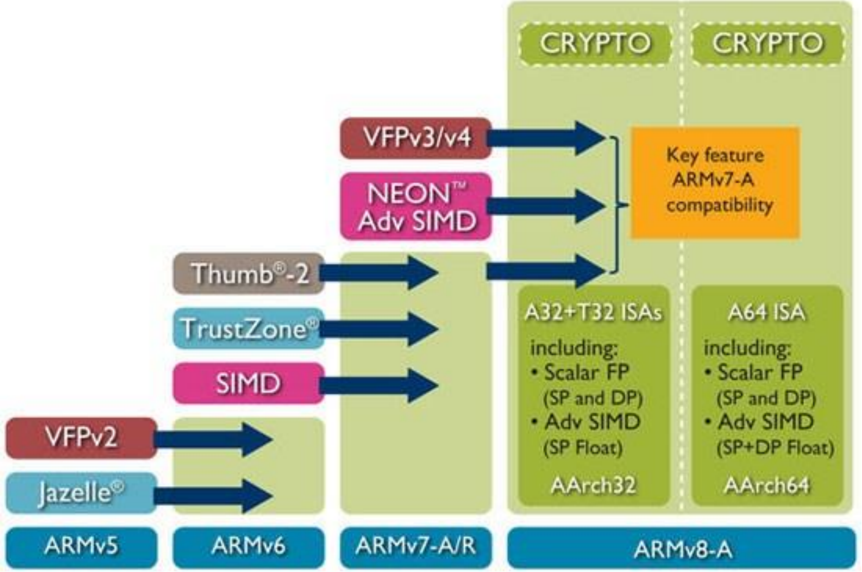
Classic LIBC Issue

```
fstat(3, {st_mode=S_IFREG|0644, st_size=35112, ...}) = 0
mmap(NULL, 99840, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xffff8b54d000
mprotect(0xffff8b554000, 65536, PROT_NONE) = 0
mmap(0xffff8b564000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x7000) = 0xffff8b564000
close(3)                                = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffff8b54b000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffff8b549000
mprotect(0xffff8bee3000, 16384, PROT_READ) = 0
mprotect(0xffff8b564000, 4096, PROT_READ) = 0
mprotect(0xffff8b585000, 4096, PROT_READ) = 0
mprotect(0xffff8b708000, 16384, PROT_READ) = 0
mprotect(0xffff8b738000, 4096, PROT_READ) = 0
mprotect(0xffff8c2fb000, 4096, PROT_READ) = 0
mprotect(0xffff8bbf9000, 4096, PROT_READ) = 0
mprotect(0xffff8b839000, 45056, PROT_READ) = 0
mprotect(0xffff8bcea000, 4096, PROT_READ) = 0
mprotect(0xffff8b8c0000, 4096, PROT_READ) = 0
mprotect(0xffff8b941000, 4096, PROT_READ) = 0
mprotect(0xffff8b9c7000, 4096, PROT_READ) = 0
mprotect(0xffff8b985000, 4096, PROT_READ) = 0
mprotect(0xffff8ba0a000, 4096, PROT_READ) = 0
mprotect(0xffff8bb68000, 53248, PROT_READ) = 0
mprotect(0xffff8bb8c000, 4096, PROT_READ) = 0
mprotect(0xffff8bbaf000, 4096, PROT_READ) = 0
mprotect(0xffff8bf0f000, 4096, PROT_READ) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffff8b547000
mprotect(0xffff8c08e000, 40960, PROT_READ) = 0
mprotect(0xffff8bd51000, 155648, PROT_READ|PROT_WRITE) = 0
mprotect(0xffff8bd51000, 155648, PROT_READ|PROT_EXEC) = 0
mprotect(0xffff8c1df000, 32768, PROT_READ) = 0
mprotect(0xffff8c59f000, 4096, PROT_READ) = 0
munmap(0xffff8c596000, 19536)           = 0
set_tid_address(0xffff8b549500)         = 3637
set_robust_list(0xffff8b549510, 24)     = 0
rt_sigaction(SIGRTMIN, {sa_handler=0xffff8c2da768, sa_mask=[], sa_flags=SA_SIGINFO}, NULL, 8) = 0
rt_sigaction(SIGRT_1, {sa_handler=0xffff8c2da838, sa_mask=[], sa_flags=SA_RESTART|SA_SIGINFO}, NULL, 8) = 0
rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
--- SIGILL {si_signo=SIGILL, si_code=ILL_ILLOPC, si_addr=0xffff8c574338} ---
+++ killed by SIGILL +++
Illegal instruction
```

Hardware is not “down gradable”

Assembly Instruction Compatibility



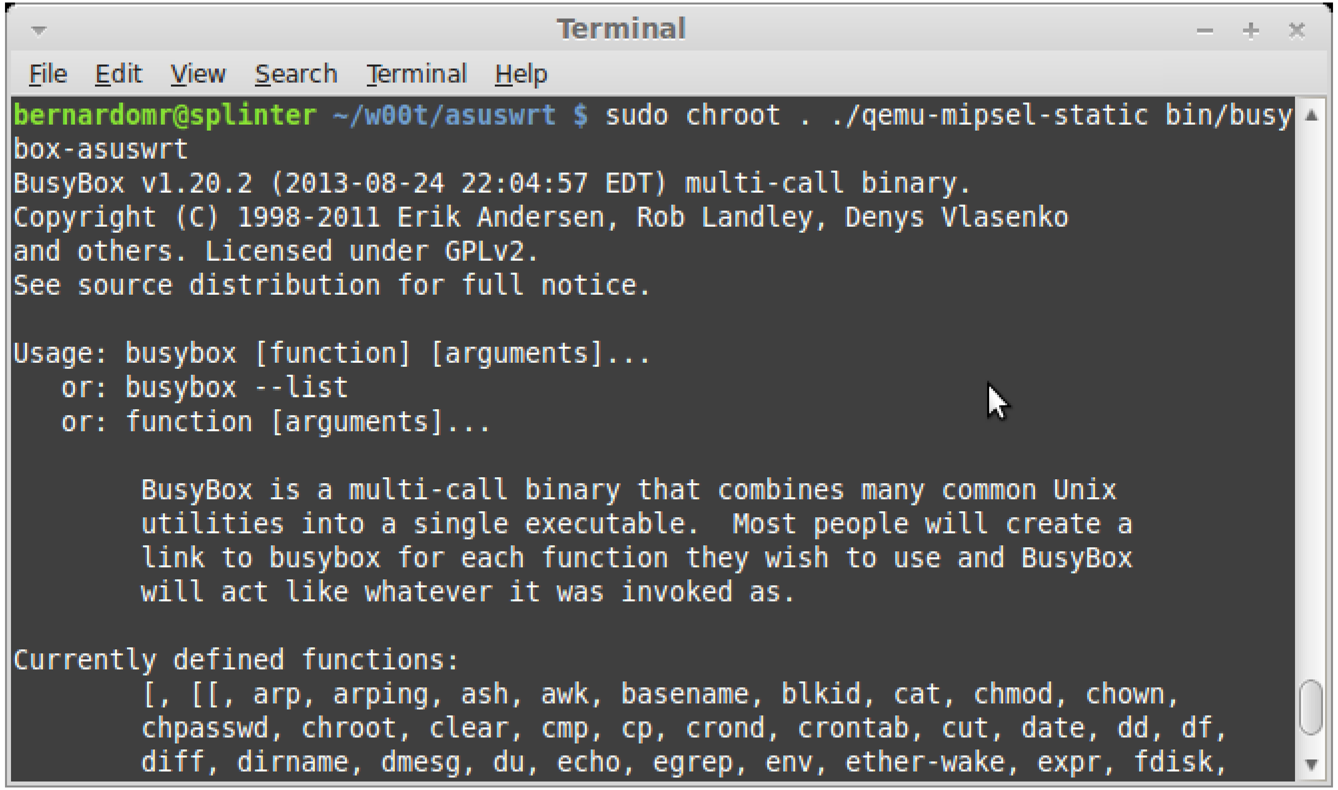


ARM

AARCH64

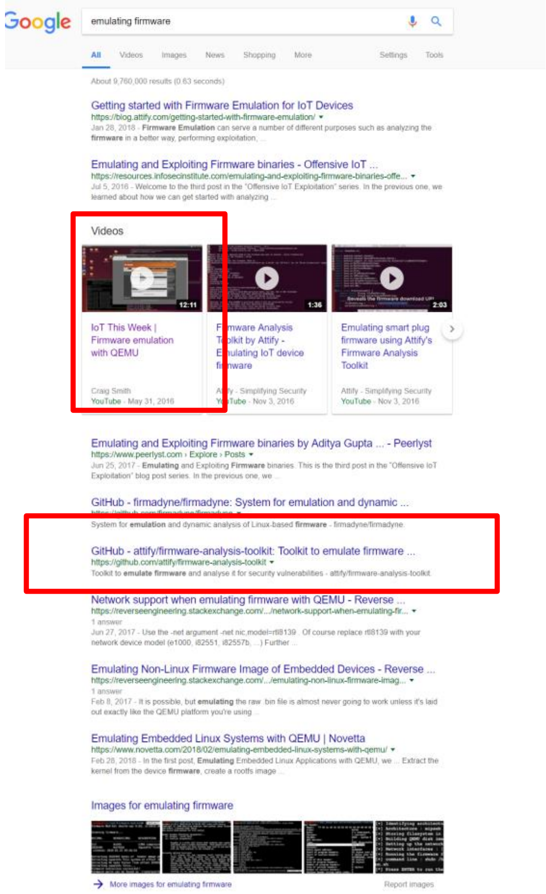
Current Work Around

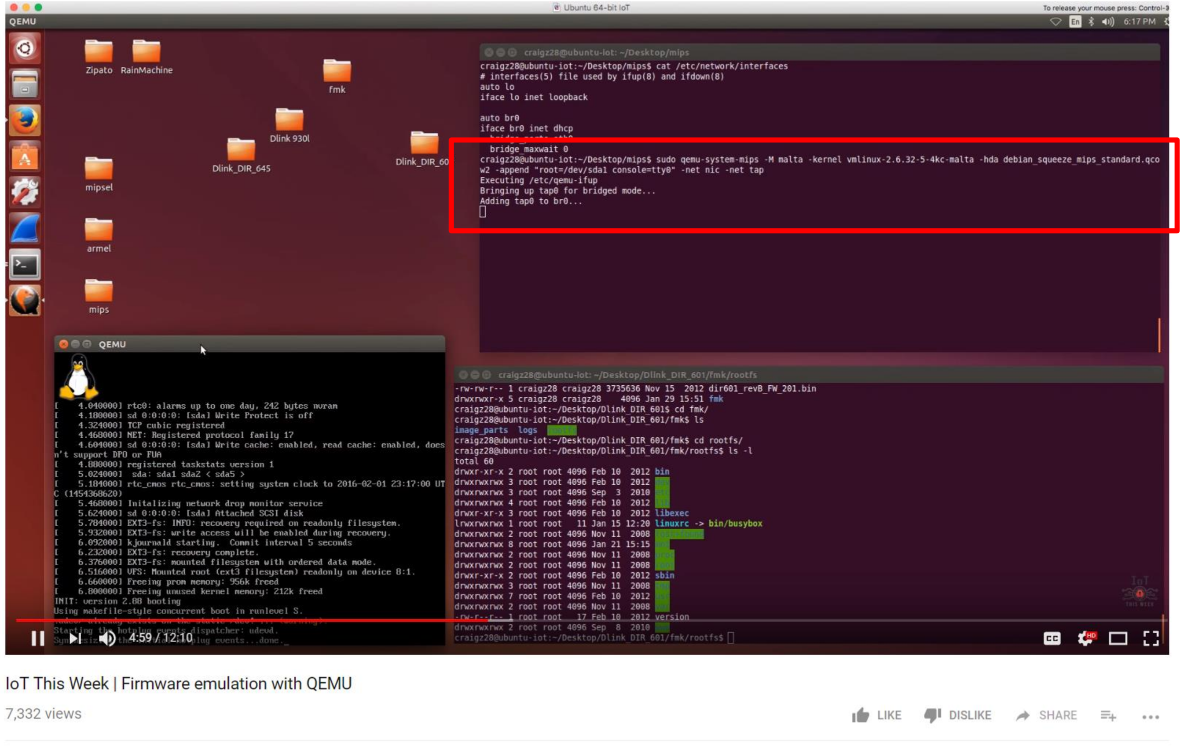
Qemu Static



QEMU-Static is good for binary execution without additional software or hardware interection

Current Primitive Firmware Emulation





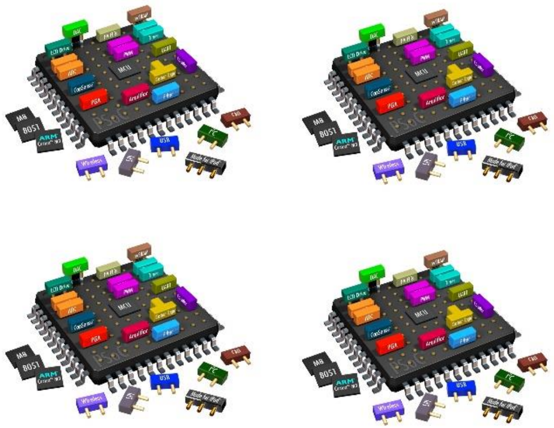
Leaving squashfs and going into a unknown world

Its not easy after 2016

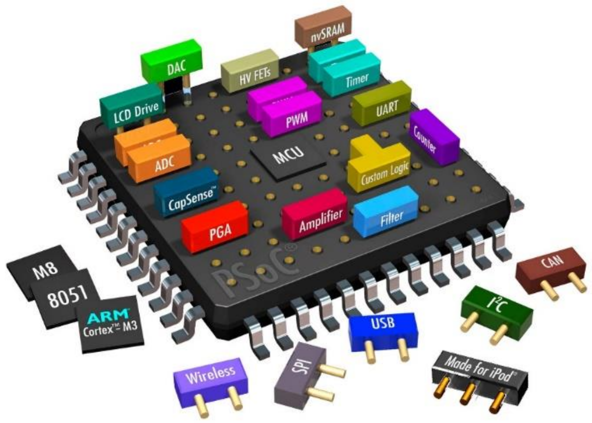
Why Firmware Emulation

More Resources = More Power

Multicore



MAX RAM



MAX Space

Processor

Normally 1-2 Core

RAM

Normally
256MB/512MB

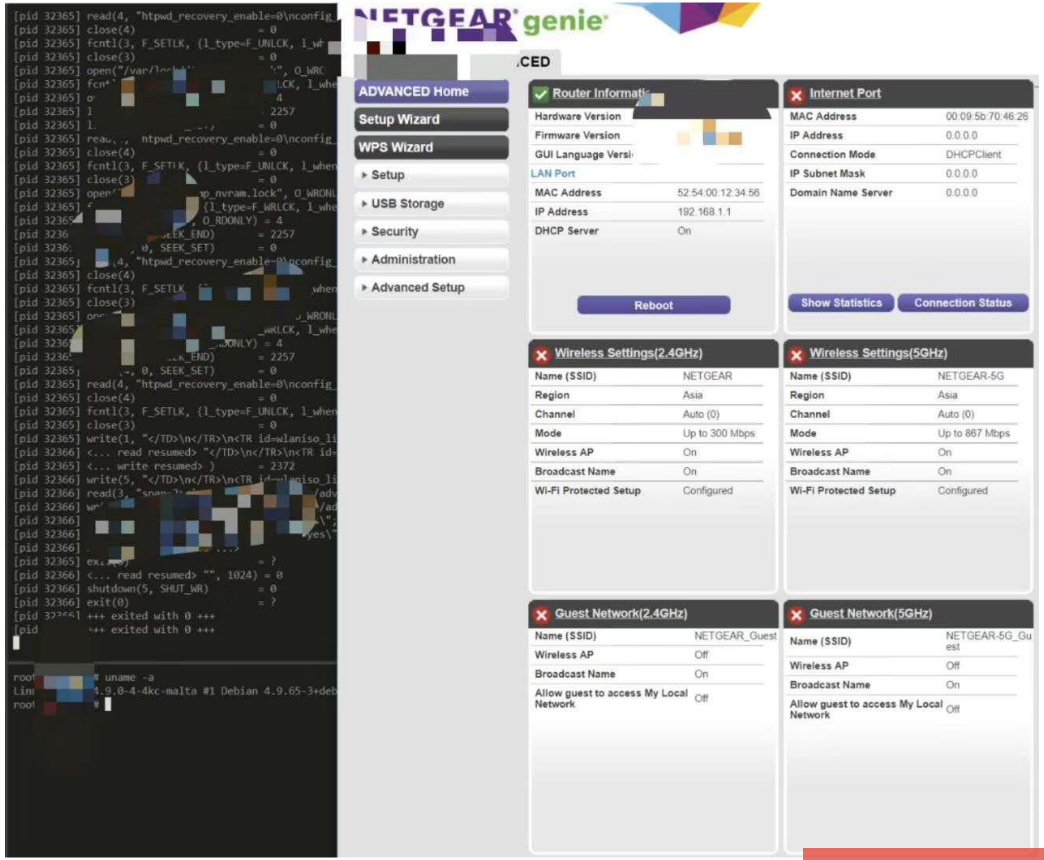
FLASH

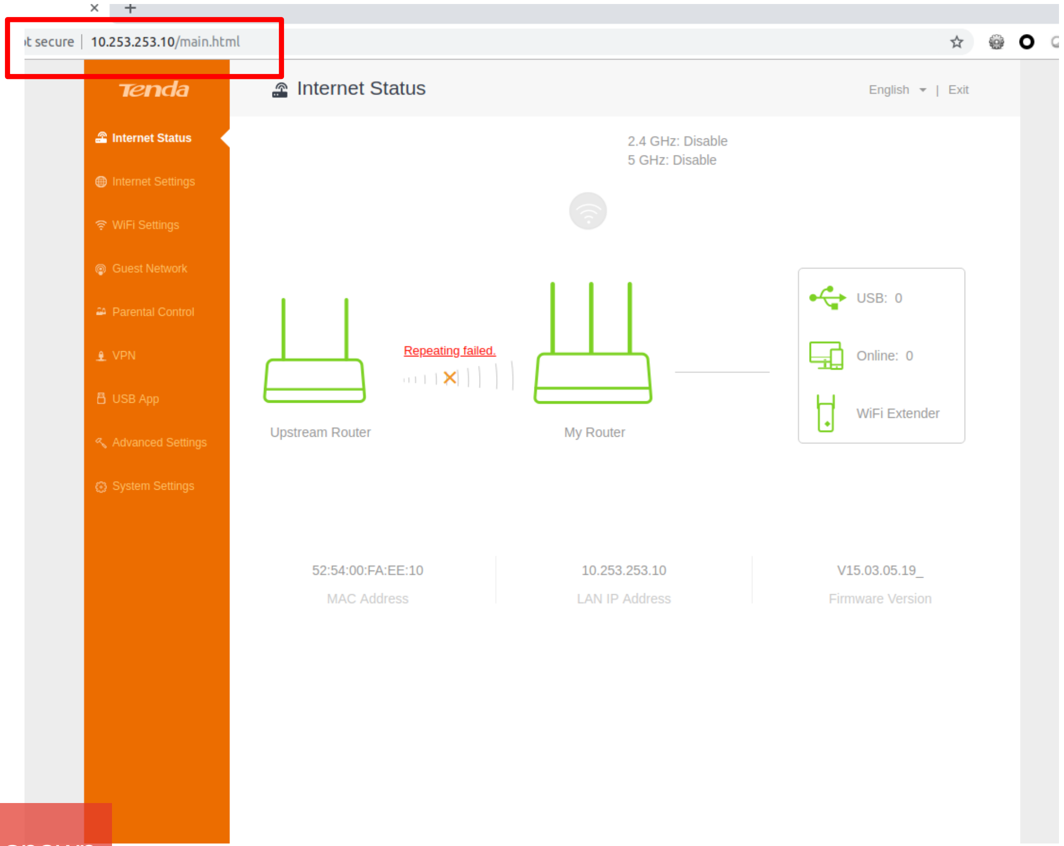
Normally
8MB/16MB/32MB/256MB

Most Important, we got apt-get

Objectives

Only One Process with Interaction





Hunt for the one that spawn listener port

most of the devices comes with one big binary

Booting Up

Distro and Kernel Mix and Match

script to boot arm

```
#!/bin/bash

sudo tunctl -d tap0

sudo screen -dm /opt/qemu/bin/qemu-system-arm -m 2048 -M virt -cpu cortex-a15 -smp cpus=
4,maxcpus=4 -kernel boot.stretch.armhf.virt/vmlinuz-4.9.0-6-armmp-lpae -initrd boot.stre
tch.armhf.virt/initrd.img-4.9.0-6-armmp-lpae -append "root=/dev/vda2" -drive file=debian
-stretch.armhf_virt.qcow2,if=none,format=qcow2,id=hd0 -device virtio-blk-device,drive=hd
0 -netdev type=tap,id=net0 -device virtio-net-device,netdev=net0,mac=52:54:00:fa:ee:10 -
nographic

sudo sysctl -w net.ipv4.ip_forward=1

echo "Stopping firewall and allowing everyone..."
sudo iptables -F
sudo iptables -X
sudo iptables -t nat -F
sudo iptables -t nat -X
sudo iptables -t mangle -F
sudo iptables -t mangle -X
sudo iptables -P INPUT ACCEPT
sudo iptables -P FORWARD ACCEPT
sudo iptables -P OUTPUT ACCEPT

sudo iptables -t nat -A POSTROUTING -o ens33 -j MASQUERADE
sudo iptables -I FORWARD 1 -i tap0 -j ACCEPT
sudo iptables -I FORWARD 1 -o tap0 -m state --state RELATED,ESTABLISHED -j ACCEPT


sudo iptables -t nat -A PREROUTING -i ens33 -p tcp --dport 1022 -j DNAT --to-destination
 10.253.253.10:22
sudo iptables -t nat -A PREROUTING -i ens33 -p tcp --dport 1080 -j DNAT --to-destination
 10.253.253.10:80
sudo iptables -t nat -A PREROUTING -i ens33 -p tcp --dport 10443 -j DNAT --to-destinatio
n 10.253.253.10:443

echo "Booting VM, eta 10 seconds"
sleep 10
sudo ifconfig tap0 10.253.253.254 netmask 255.255.255.0
```

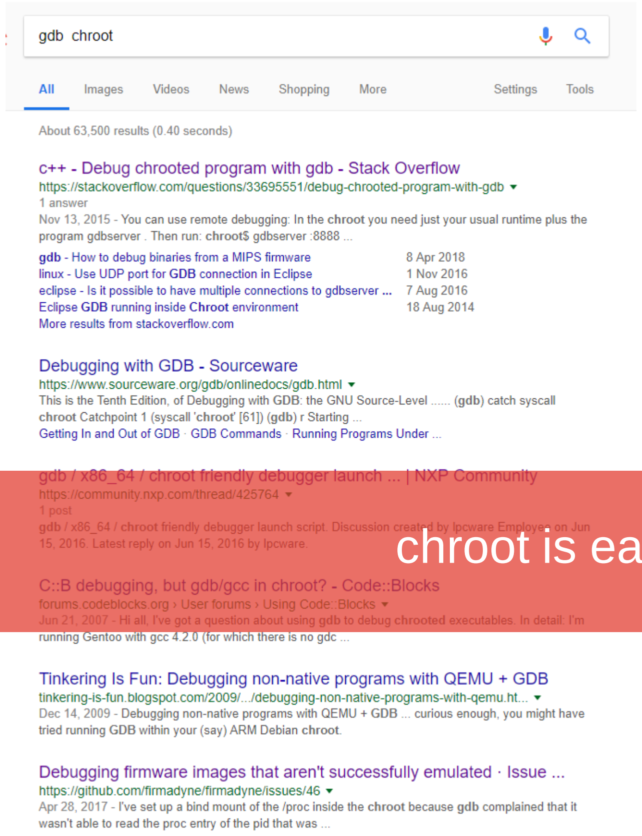
script to boot mips

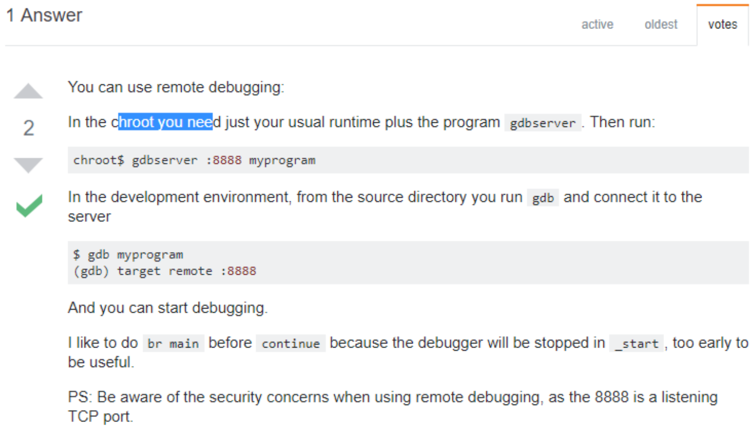
```
#!/bin/bash

sudo screen -dm /opt/qemu/bin/qemu-system-mipsel -m 512 -M malta -kernel boot.stretch.mi
psel/vmlinux-4.9.0-4-4kc-malta -initrd boot.stretch.mipsel/initrd.img-4.9.0-4-4kc-malta
-append "root=/dev/sda1  net.ifnames=0 biosdevname=0 nokaslr" -hda debian-stretch.mipsel
.qcow2 -net nic -net tap,ifname=tap0,script=no,downscript=no -net nic -net tap,ifname=ta
p1,script=no,downscript=no -nographic

sudo tunctl -t tap0 -u xwings
sudo ifconfig tap0 10.253.253.254 netmask 255.255.255.0

sudo sysctl -w net.ipv4.ip_forward=1

echo "Stopping firewall and allowing everyone..."
sudo iptables -F
sudo iptables -X
sudo iptables -t nat -F
sudo iptables -t nat -X
sudo iptables -t mangle -F
sudo iptables -t mangle -X
sudo iptables -P INPUT ACCEPT
sudo iptables -P FORWARD ACCEPT
sudo iptables -P OUTPUT ACCEPT

sudo iptables -t nat -A POSTROUTING -o ens33 -j MASQUERADE
sudo iptables -I FORWARD 1 -i tap0 -j ACCEPT
sudo iptables -I FORWARD 1 -o tap0 -m state --state RELATED,ESTABLISHED -j ACCEPT


sudo iptables -t nat -A PREROUTING -i ens33 -p tcp --dport 1122 -j DNAT --to-destination
 10.253.253.11:22
sudo iptables -t nat -A PREROUTING -i ens33 -p tcp --dport 1180 -j DNAT --to-destination
 10.253.253.11:80
sudo iptables -t nat -A PREROUTING -i ens33 -p tcp --dport 11443 -j DNAT --to-destinatio
n 10.253.253.11:443
```

argument: running new or old distro + kernel

chroot

Easy Way Out, chroot





chroot is easy (still hardware dependent), but we will have issue with tools

Running without chroot

Classic Case: File Not Found

The File Missing Trick

We Missed You

```
chdir("/")                                = 0
execve("/bin/bash", ["/bin/bash", "-i"], 0xffffca14f650 /* 18 vars */) = -1 ENOENT (No such file or d
irectory)
openat(AT_FDCWD, "/usr/lib/aarch64-linux-gnu/charset.alias", O_RDONLY|O_NOFOLLOW) = -1 ENOENT (No suc
h file or directory)
write(2, "chroot: ", 8chroot: )                   = 8
write(2, "failed to run command '/bin/bash'", 33failed to run command '/bin/bash') = 33
write(2, ": No such file or directory", 27: No such file or directory) = 27
write(2, "\n", 1
)                        = 1
close(1)                                  = 0
close(2)                                  = 0
exit_group(127)                           = ?
```

We found you

```
root@rpi3:/opt/[redacted]/lib64# file ../bin/bash
../bin/bash: ELF 64-bit LSB executable, ARM aarch64, version 1 (SYSV), dynamically linked, interprete
r /lib64/ld-linux-aarch64.so.1, for GNU/Linux 3.14.0, BuildID[sha1]=22e2854c58b1814825b95cba103ac658d
371f5b0, stripped
```

The missing .SO and binary Issue

Out from chroot, we need feeeding

```
[pid  2680] close(4)                        = 0
[pid  2680] write(1, "<dhcpc script>no udhcpc pid can be killed, but udhcpc id is ", 60) = 60
[pid  2680] newfstatat(AT_FDCWD, "/usr/local/sbin/ps", 0xfffffe081a30, 0) = -1 ENOENT (No such file
r directory)
[pid  2680] newfstatat(AT_FDCWD, "/usr/local/bin/ps", 0xfffffe081a30, 0) = -1 ENOENT (No such file o
 directory)
[pid  2680] newfstatat(AT_FDCWD, "/usr/sbin/ps", 0xfffffe081a30, 0) = -1 ENOENT (No such file or dir
ctory)
[pid  2680] newfstatat(AT_FDCWD, "/usr/bin/ps", 0xfffffe081a30, 0) = -1 ENOENT (No such file or dire
tory)
[pid  2680] newfstatat(AT_FDCWD, "/sbin/ps", 0xfffffe081a30, 0) = -1 ENOENT (No such file or directo
y)
[pid  2680] newfstatat(AT_FDCWD, "/bin/ps", {st_mode=S_IFREG|0755, st_size=535832, ...}, 0) = 0
[pid  2680] pipe2([4, 7], 0)                = 0
[pid  2680] clone(strace: Process 2681 attached
```

```
Usage: unzip [-lnopq] FILE[.zip] [FILE]... [-x FILE...] [-d DIR]
root@aarch64:/opt/[redacted]12/bin# ln -s busybox.nosuid unzip
root@aarch64:/opt/[redacted]12/bin# ./busybox.nosuid sync
root@aarch64:/opt/[redacted]12/bin# ./busybox.nosuid syn
syn: applet not found
root@aarch64:/opt/[redacted]12/bin# ln -s busybox.nosuid sync
root@aarch64:/opt/[redacted]12/bin#
```

```
root@[redacted]2/usr/lib64# ln -s libgnutls.so.30.9.0 libgnutls.so.30
root@[redacted]2/usr/lib64# ln -s libidn.so.11.6.16 libidn.so.11
root@[redacted]2/usr/lib64# ln -s libnettle.so.6.2 libnettle.so.6
root@[redacted]2/usr/lib64# ln -s libhogweed.so.4.2 libhogweed.so.4
root@[redacted]2/usr/lib64# ln -s libgmp.so.10.3.1 libgmp.so.10
root@[redacted]2/usr/lib64# ln -s libpcre.so.1.2.7 libpcre.so.1
root@[redacted]2/usr/lib64# ln -s libexpat.so.1.6.2 libexpat.so.1
root@[redacted]2/usr/lib64#
```

Feeding all the required so and binary with “ln –s”

Out from chroot, we need feeding

```
bash-3.2# /usr/bin/appmainprog
<appmain>*********************************************
<appmain>child process id is 3931
<appmain>Appcliation Init Begin
<appmain>Audio Mas process Init
[Aud][PPC] AudioPPCControl constructor
[Aud][PPC] AudioPPCControl getInstance
[Aud][PPC] AudioPPCControl freeInstance
[Aud][PPC] AudioPPCControl destructor
[Aud][PPC][deInit] PPC deinit begin.
[Aud][PPC][ppcStructUnalloc] ppc_destroy_info begin.
Segmentation fault
bash-3.2#
```

```
close(3)                                = 0
write(1, "<appmain>Appcliation Init Begin\n", 32<appmain>Appcliation Init Begin
) = 32
write(1, "<appmain>Audio Mas process Init\n", 32<appmain>Audio Mas process Init
) = 32
umask(000)                              = 022
faccessat(AT_FDCWD, "/data/log_all", F_OK) = -1 ENOENT (No such file or directory)
socket(AF_UNIX, SOCK_DGRAM|SOCK_CLOEXEC, 0) = 3
connect(3, {sa_family=AF_UNIX, sun_path="/dev/log"}, 110) = -1 ENOENT (No such file or directory)
close(3)                                = 0
write(1, "[Aud][PPC] AudioPPCControl constructor\n", 39[Aud][PPC] AudioPPCControl constructor
) = 39
write(1, "[Aud][PPC] AudioPPCControl getInstance\n", 39[Aud][PPC] AudioPPCControl getInstance
) = 39
faccessat(AT_FDCWD, "/tmp/ppcfifo", F_OK) = -1 ENOENT (No such file or directory)
[illegible]DCWD, "/tmp/ppcfifo", S_IFIFO|0777) = -1 ENOENT (No such file or directory)
```

Classical file not found error

"segfault" without clear error. strace come to rescue

The Secretive NVRAM

Dark side of NVRAM

main process

```
2750] openat(AT_FDCWD, "/data/nvram/APCFG/APRDEB/BT_Addr", O_RDONLY) = 5
2750] flock(5, LOCK_SH)               = 0
2750] read(5, "\0\0F\201g\1`\0#\20\0\0\7\200\0\6\5\7\3@\37@\37\0\4\200\0\377\
)\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 64) = 64
2750] close(5)                        = 0
2750] openat(AT_FDCWD, "/dev/disk/by-partlabel/NVRAM", O_RDWR) = -1 ENOENT (N
)
2750] openat(AT_FDCWD, "/dev/mtd1", O_RDWR) = -1 ENOENT (No such file or dire
2750] openat(AT_FDCWD, "/data/nvram/APCFG/APRDEB/PRODUCT_INFO", O_RDONLY) = 5
2750] close(5)                        = 0
2750] newfstatat(AT_FDCWD, "/data/nvram/APCFG/APRDCL/FILE_VER", {st_mode=S_IF
...}, 0) = 0
2750] openat(AT_FDCWD, "/data/nvram/APCFG/APRDCL/FILE_VER", O_RDONLY) = 5
2750] read(5, "NVRAM_VER_INFO\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 70) = 70
2750] lseek(5, 3626, SEEK_SET)        = 3626
2750] read(5, "PRODUCT_INFO\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
```

ask for nvram info

Relationship between main binary is so intimate, but in actual fact. Is just a hit and run

reply with nvram info

```
root@rpi3:/opt/[redacted]# strace -f -s 256 chroot /opt/[redacted]/ /usr/bin/appmainprog
/abc 2>&1
^Croot@rpi3:/opt/[redacted]# ^C
root@rpi3:/opt/[redacted]# ^C
root@rpi3:/opt[redacted]# cat /tmp/abc | grep nvram
openat(AT_FDCWD, "/lib64/libnvram.so", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib64/libnvram_custom.so", O_RDONLY|O_CLOEXEC) = 3
```

interactor

Dark Side of NVRAM

main process

```
2750] close(5)                         = 0
2750] openat(AT_FDCWD, "/data/nvram/APCFG/APRDEB/BT_Addr", O_RDONLY) = 5
2750] flock(5, LOCK_SH)                = 0
2750] read(5, "\0\0F\201g\1`\0#\20\0\0\7\200\0\6\5\7\3@\37@\37\0\4\200\0\377\
)\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 64) = 64
2750] close(5)                         = 0
2750] openat(AT_FDCWD, "/dev/disk/by-partlabel/NVRAM", O_RDWR) = -1 ENOENT (N
)
2750] openat(AT_FDCWD, "/dev/mtd1", O_RDWR) = -1 ENOENT (No such file or dire
2750] openat(AT_FDCWD, "/data/nvram/APCFG/APRDEB/PRODUCT_INFO", O_RDONLY) = 5
2750] close(5)                         = 0
2750] newfstatat(AT_FDCWD, "/data/nvram/APCFG/APRDCL/FILE_VER", {st_mode=S_IF
...}, 0) = 0
2750] openat(AT_FDCWD, "/data/nvram/APCFG/APRDCL/FILE_VER", O_RDONLY) = 5
2750] read(5, "NVRAM_VER_INFO\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 70) = 70
2750] lseek(5, 3626, SEEK_SET)     = 3626
2750] read(5, "PRODUCT_INFO\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
```

ask for nvram info

Relationship between main binary is so intimate, but in actual fact. Is just a hit and run

reply with nvram info

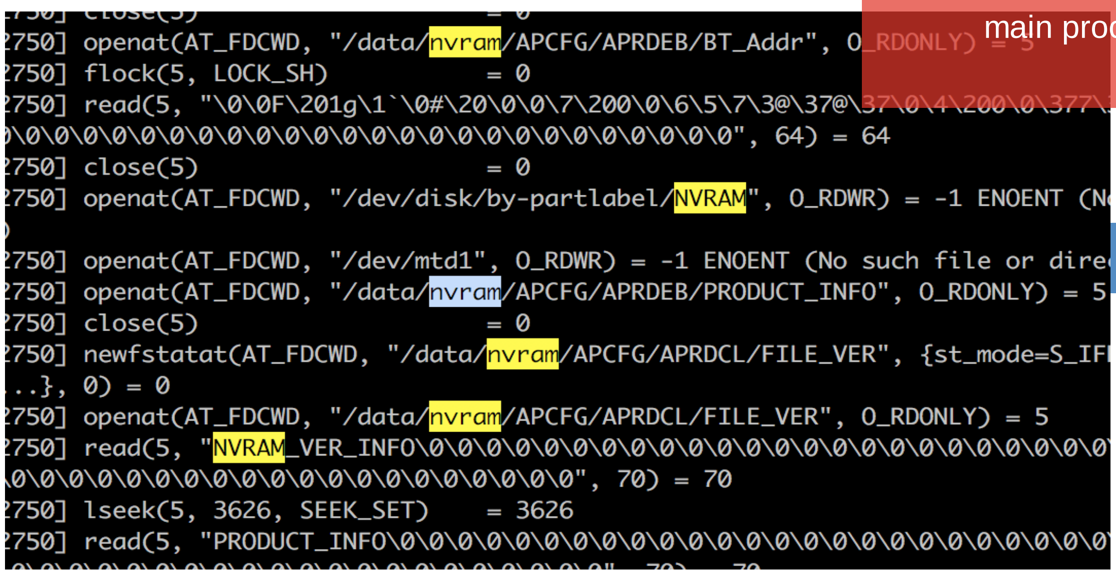
```
root@rpi3:/opt/[redacted]# strace -f -s 256 chroot /opt/[redacted]/ /usr/bin/appmainprog
/abc 2>&1
^Croot@rpi3:/opt/[redacted]# ^C
root@rpi3:/opt/[redacted]# ^C
root@rpi3:/opt[redacted]# cat /tmp/abc | grep nvram
openat(AT_FDCWD, "/lib64/libnvram.so", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib64/libnvram_custom.so", O_RDONLY|O_CLOEXEC) = 3
```

interactor

Dark Side of the main process, we ignore and con't to next step

```
efused)
[pid  3088] close(5)                    = 0
[pid  3088] write(1, "[08-28 20:45:32][utils/SNManager.cpp:26][D] : Read NVRAM Failed\n", 64[08-28 2
:45:32][utils/SNManager.cpp:26][D] : Read NVRAM Failed
) = 64
[pid  3088] write(1, "<AST>[RegisterCmdHandler:113]:Cmd [22] Registered Handler!\n", 59<AST>[Registe
```

A fake NVRAM



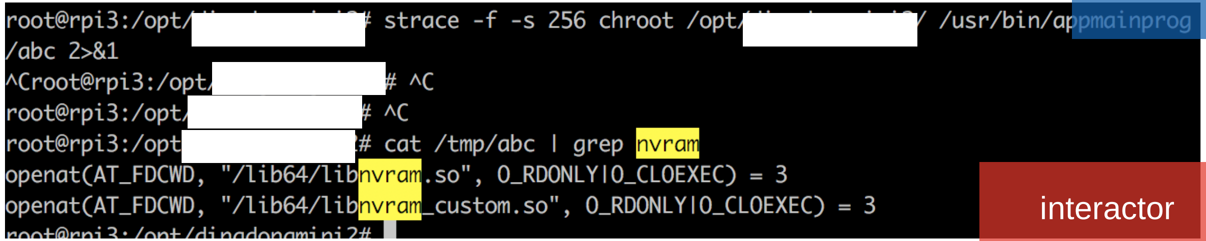
main process

ask for nvram info

IF interactor is the medium,

can we fake it ?

reply with nvram info



interactor

A fake NVRAM

```
2750] openat(AT_FDCWD, "/data/nvram/APCFG/APRDEB/BT_Addr", O_RDONLY) = 5
2750] flock(5, LOCK_SH)                = 0
2750] read(5, "\0\0F\201g\1`\0#\20\0\0\7\200\0\6\5\7\3@\37@\37\0\4\200\0\377\
)\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 64) = 64
2750] close(5)                         = 0
2750] openat(AT_FDCWD, "/dev/disk/by-partlabel/NVRAM", O_RDWR) = -1 ENOENT (N
)
2750] openat(AT_FDCWD, "/dev/mtd1", O_RDWR) = -1 ENOENT (No such file or dire
2750] openat(AT_FDCWD, "/data/nvram/APCFG/APRDEB/PRODUCT_INFO", O_RDONLY) = 5
2750] close(5)                         = 0
2750] newfstatat(AT_FDCWD, "/data/nvram/APCFG/APRDCL/FILE_VER", {st_mode=S_IF
..}, 0) = 0
2750] openat(AT_FDCWD, "/data/nvram/APCFG/APRDCL/FILE_VER", O_RDONLY) = 5
2750] read(5, "NVRAM_VER_INFO\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 70) = 70
2750] lseek(5, 3626, SEEK_SET)     = 3626
2750] read(5, "PRODUCT_INFO\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
```

main process

ask for nvram info

IF interactor is the medium,

can we fake it ?

reply with nvram info

```
root@rpi3:/opt/[redacted]# strace -f -s 256 chroot /opt/[redacted] /usr/bin/appmainprog
/abc 2>&1
^Croot@rpi3:/opt/[redacted]# ^C
root@rpi3:/opt/[redacted]# ^C
root@rpi3:/opt[redacted]# cat /tmp/abc | grep nvram
openat(AT_FDCWD, "/lib64/libnvram.so", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib64/libnvram_custom.so", O_RDONLY|O_CLOEXEC) = 3
```

interactor

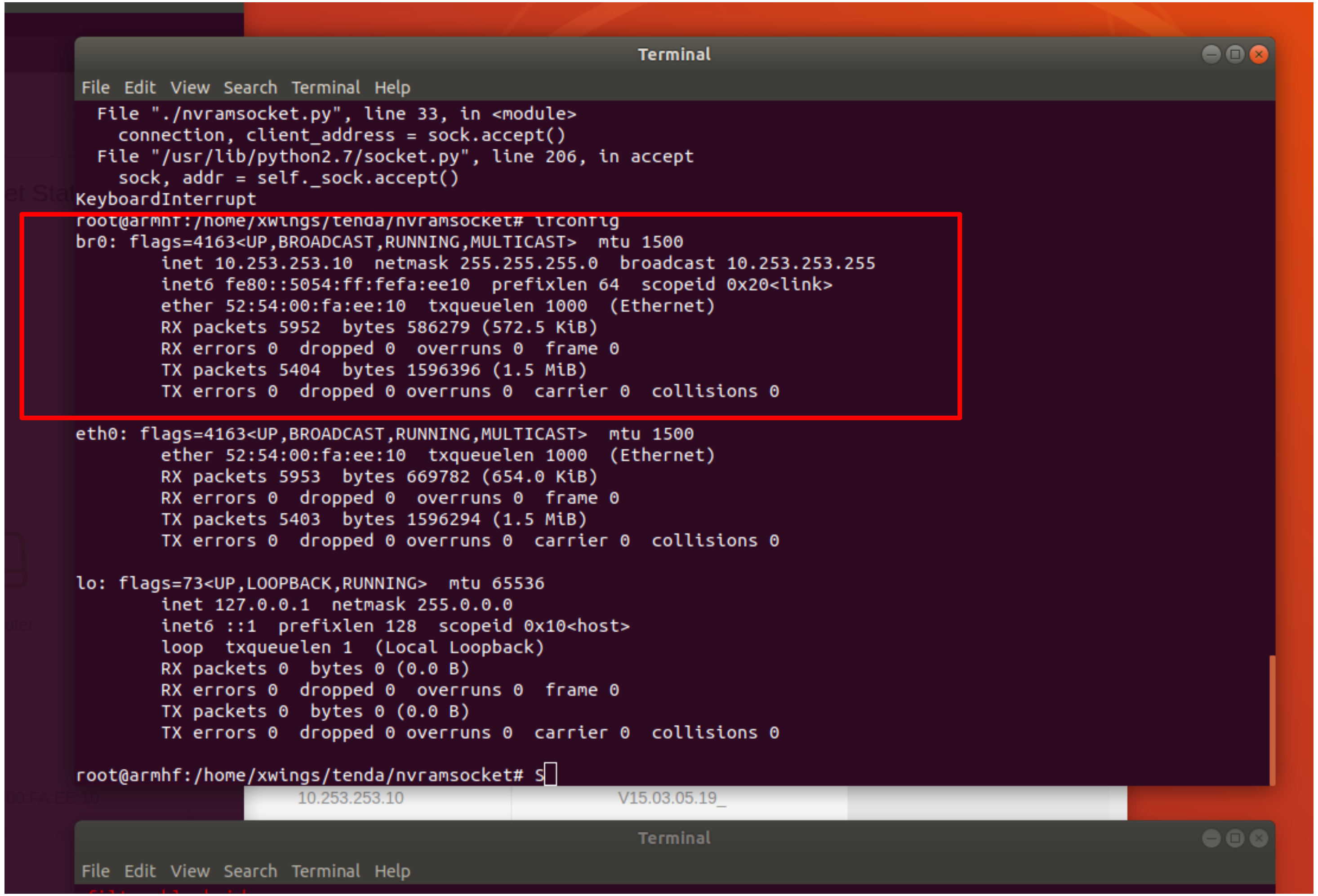
nvramsocket.py 2.4 KB

```python
#!/usr/bin/python

# For [redacted]ulation
# This code suppose to replace cfmd
# cfmd suppose to be the bridge between nvram and httpd and othe
# so far only httpd works will find out more`

import socket
import sys
import os

server_address = '/opt/[redacted]socket'
data = ''

# Make sure the socket does not already exist
try:
        os.unlink(server_address)
except OSError:
        if os.path.exists(server_address):
                raise
# Create a UDS socket
sock = socket.socket(socket.AF_UNIX,socket.SOCK_STREAM)
# Bind the socket to the port
print >>sys.stderr, 'starting up on %s' % server_address
sock.bind(server_address)

# Listen for incoming connections
sock.listen(1)

while True:
    # Wait for a connection
    #print >>sys.stderr, 'waiting for a connection'
    connection, client_address = sock.accept()
    try:
        #print >>sys.stderr, 'connection from', client_address
        while True:
            data += connection.recv(1024)
            data = str(data)
            #data = data.decode('utf-8')
```

Custom Interactor

br0

The bridge trick



```
Terminal
File  Edit  View  Search  Terminal  Help
  File "./nvramsocket.py", line 33, in <module>
    connection, client_address = sock.accept()
  File "/usr/lib/python2.7/socket.py", line 206, in accept
    sock, addr = self._sock.accept()
KeyboardInterrupt
root@armhf:/home/xwings/tenda/nvramsocket# ifconfig
br0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.253.253.10  netmask 255.255.255.0  broadcast 10.253.253.255
        inet6 fe80::5054:ff:fefa:ee10  prefixlen 64  scopeid 0x20<link>
        ether 52:54:00:fa:ee:10  txqueuelen 1000  (Ethernet)
        RX packets 5952  bytes 586279 (572.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5404  bytes 1596396 (1.5 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether 52:54:00:fa:ee:10  txqueuelen 1000  (Ethernet)
        RX packets 5953  bytes 669782 (654.0 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5403  bytes 1596294 (1.5 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@armhf:/home/xwings/tenda/nvramsocket# S
```


The switch looking device

Wireless Device

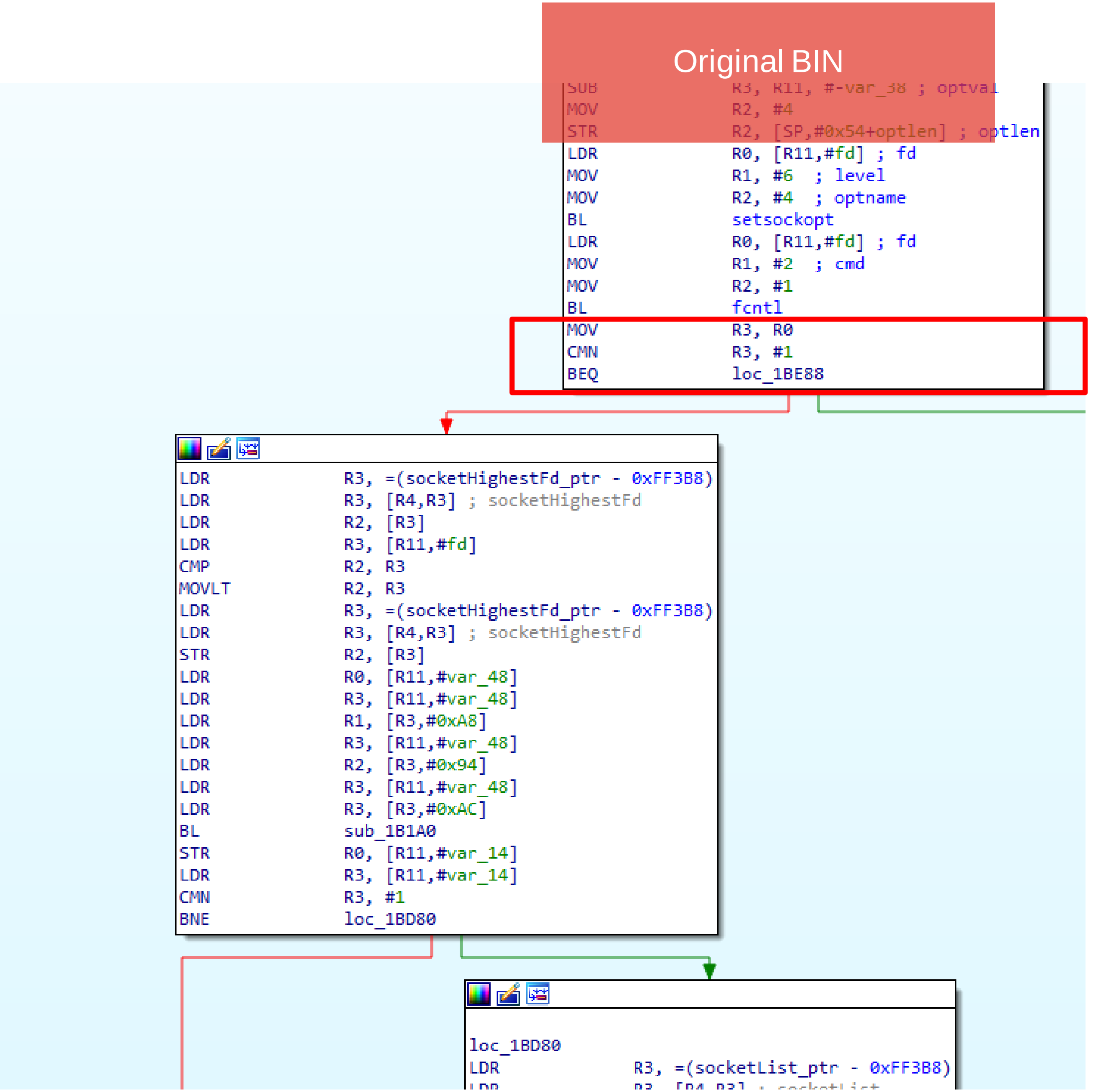
Faking wpa_supplicant

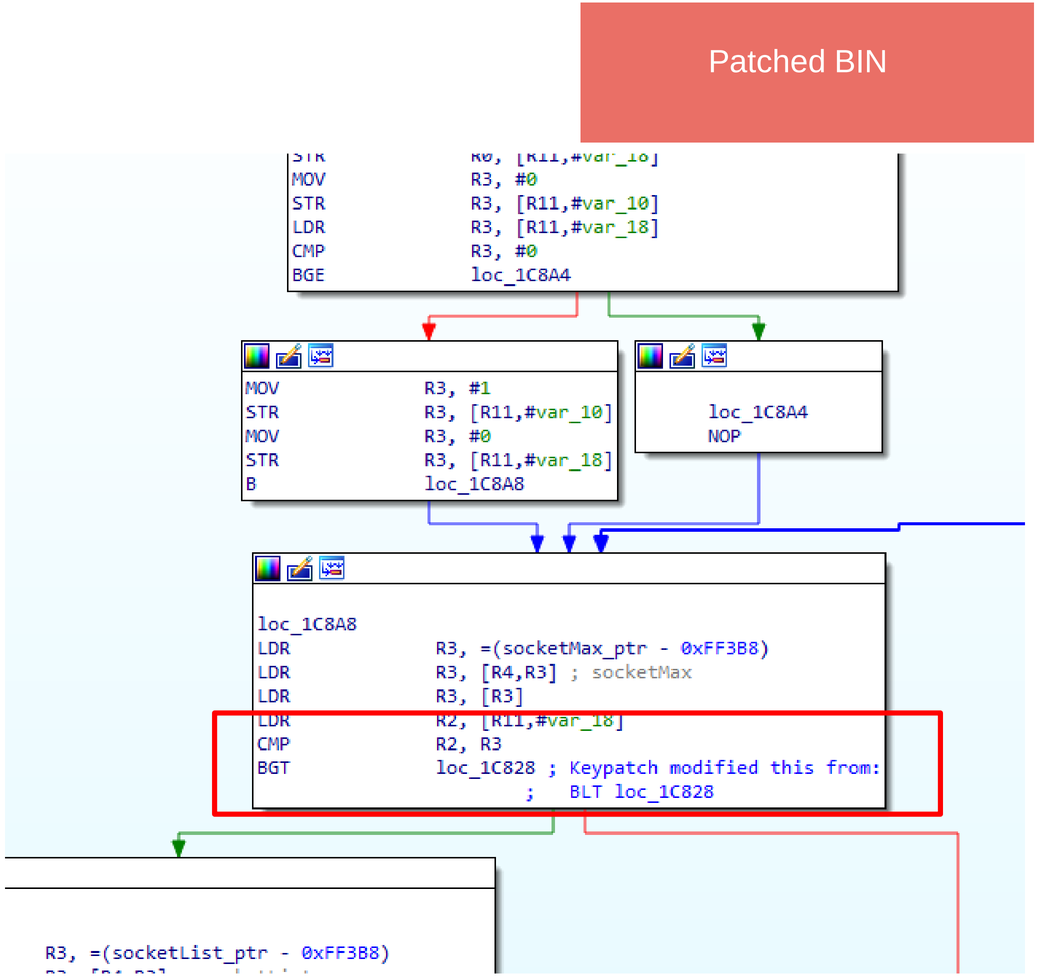
```
[WIFI_MW] Current PID=808

[WIFI_MW]
 control interface dir: /tmp/wpa_supplicant/
 wpa control client path: /tmp/wpa_supplicant/wpa_ctrl_808
 wpa monitor client path: /tmp/wpa_supplicant/wpa_moni_808
 p2p control client path: /tmp/wpa_supplicant/p2p_ctrl_808
 p2p monitor client path: /tmp/wpa_supplicant/p2p_moni_808

[WIFI_MW] [WPA_CTRL] Enter wpaCtrlOpen: ctrl_path = /tmp/wpa_supplicant/wlan0.
[WIFI_MW] wpaCtrlOpen: unlink(), ctrl->s: 11, ctrl->mLocal.sun_path: /tmp/wpa_supplicant/wpa_ct
[WIFI_MW] wpaCtrlOpen: bind(), bindRet = 0.
[WIFI_MW] wpaCtrlOpen: connect(), ctrl->s: 11, ctrl->dest.sun_path: /tmp/wpa_supplicant/wlan0
[WIFI_MW] [WPA_CTRL] Leave wpaCtrlOpen(), conn = 0.
[WIFI_MW] [WPA_CTRL] Enter wpaCtrlOpen: ctrl_path = /tmp/wpa_supplicant/wlan0.
[WIFI_MW] wpaCtrlOpen: unlink(), ctrl->s: 12, ctrl->mLocal.sun_path: /tmp/wpa_supplicant/wpa_mo
[WIFI_MW] wpaCtrlOpen: bind(), bindRet = 0.
```

making eth0 looks like wlan0 works too

Everything Things Else Fail

BL, BNE, BEQ and friends





Agenda

Coverage Guided Fuzzer vs Embedded Systems

Emulating Firmware

Skorpio Dynamic Binary Instrumentation

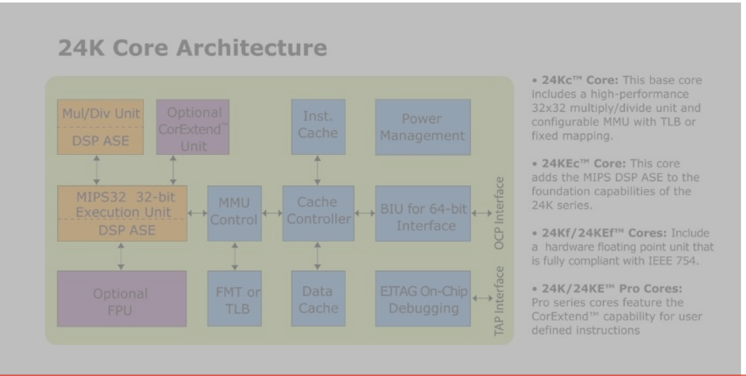
Guided Fuzzer for Embedded

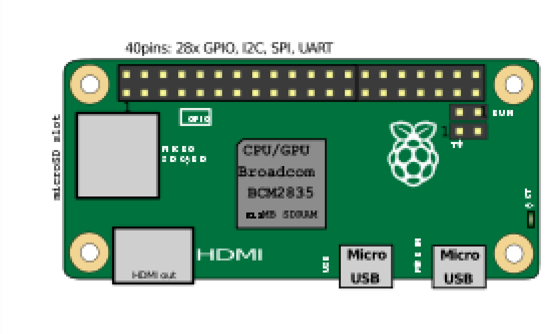
DEMO

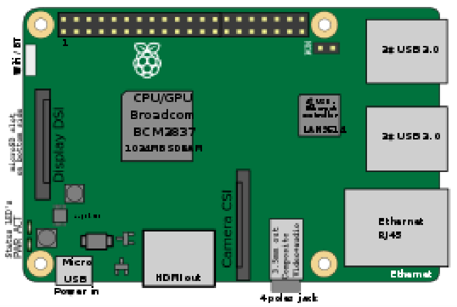
Conclusions

Secret Menu

Issues







Firmware Emulation	Closed System	Lack Support for Embedded

Firmware Emulation

- Without built-in shell access for user interaction
- Without developement facilities required for building new tools
  - Compiler
  - Debugger
  - Analysis tools

Closed System

- Binary only - without source code
  - Existing guided fuzzers rely on source code available
    - Source code is needed for branch instrumentation to feedback fuzzing progress
    - Emulation such as QEMU mode support in AFL is slow & limited in capability
    - Same issue for other tools based on Dynamic Binary Instrumentation

Lack Support for Embedded

- Most fuzzers are built for X86 only
  - Embedded systems based on Arm, Arm64, Mips, PPC
- Existing DBIs are poor for non-X86 CPU
  - Pin: Intel only
  - DynamoRio: experimental support for Arm

Dynamic Binary Instrumentation (DBI)

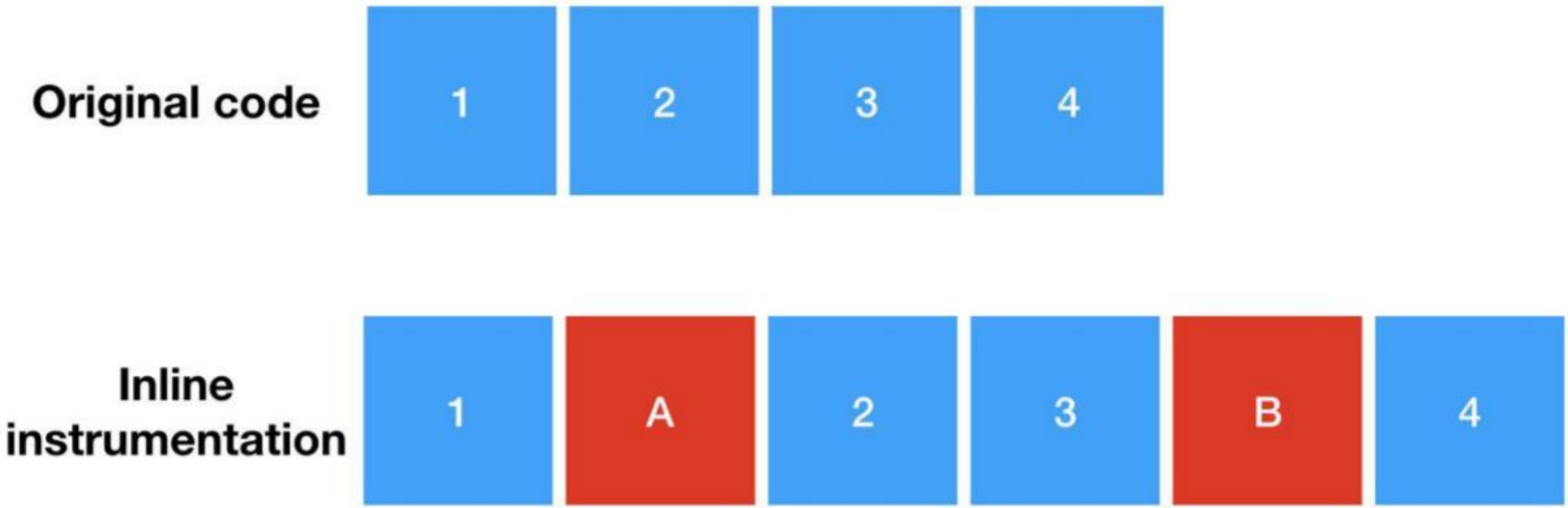
Definition

- A method of analyzing a binary application at runtime through injection of instrumentation code.
  - Extra code executed as a part of original instruction stream
  - No change to the original behavior
- Framework to build apps on top of it

Applications

- Code tracing/logging
- Debugging
- Profiling
- Security enhancement/mitigation

DBI Illustration

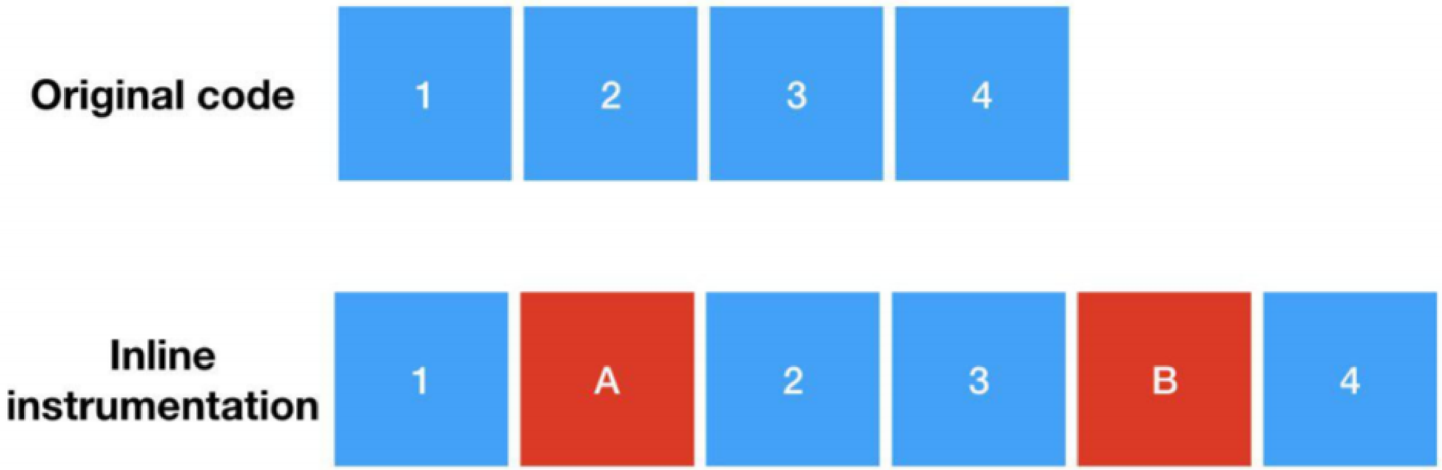


DBI Techniques

- Just-in-Time translation
  - Transparently translate & execute code at runtime
    - Perform on IR: Valgrind
    - Perform directly on native code: DynamoRio
  - Better control on code executed
  - Heavy, super complicated in design & implementation
- Hooking
  - Lightweight, much simpler to design & implement
  - Less control on code executed & need to know in advance where to instrument

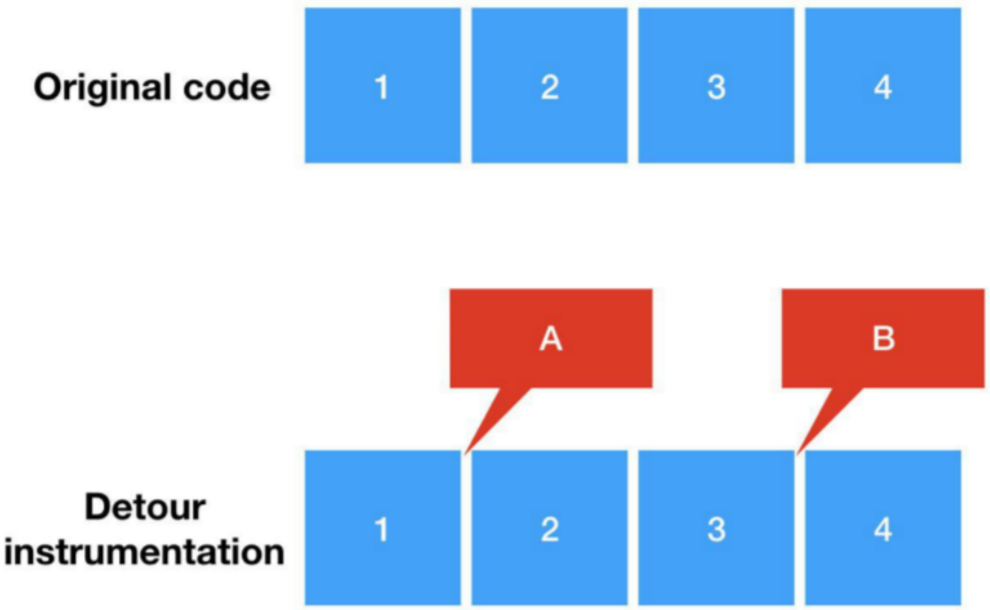
Hooking Mechanisms - Inline

- Inline code injection
  - Put instrumented code inline with original code
  - Can instrument anywhere & unlimited in extra code injected
  - Require complicated code rewrite



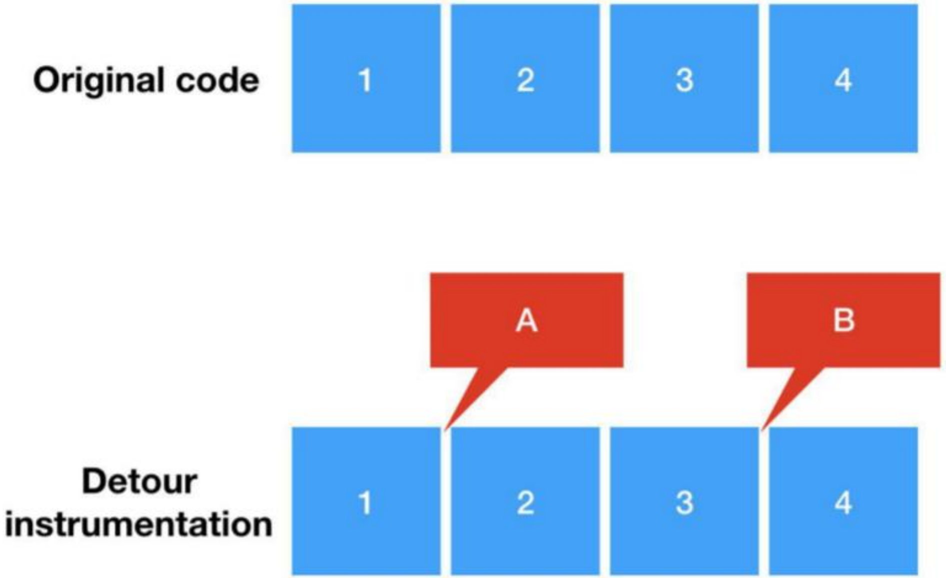
Hooking Mechanisms - Detour

- Detour injection
  - Branch to external instrumentation code
    - User-defined **CALLBACK** as instrumented code
    - **TRAMPOLINE** memory as a step-stone buffer
  - Limited on where to hook
    - Basic block too small?
  - Easier to design & implement

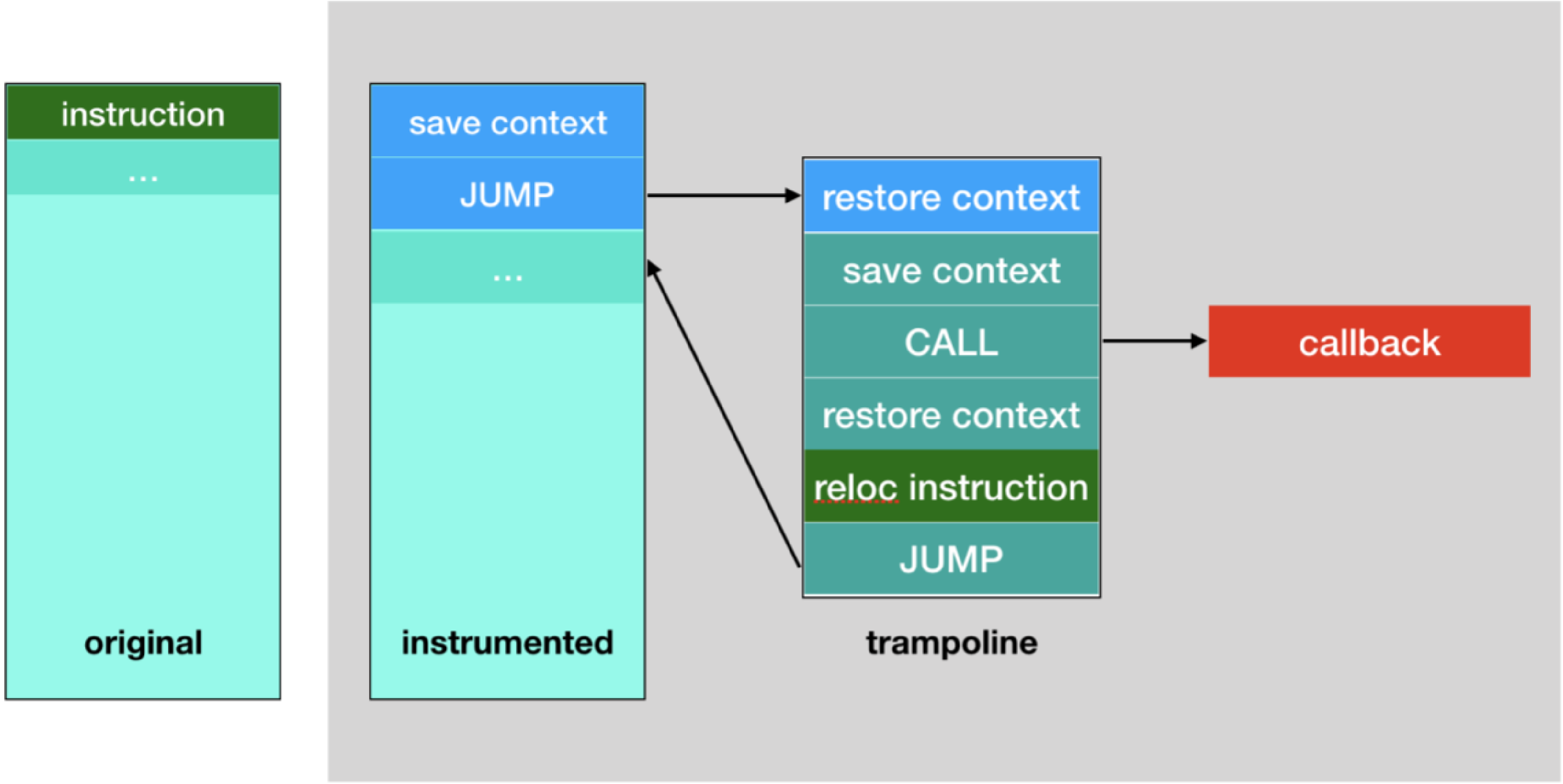


Detour Injection Mechanisms

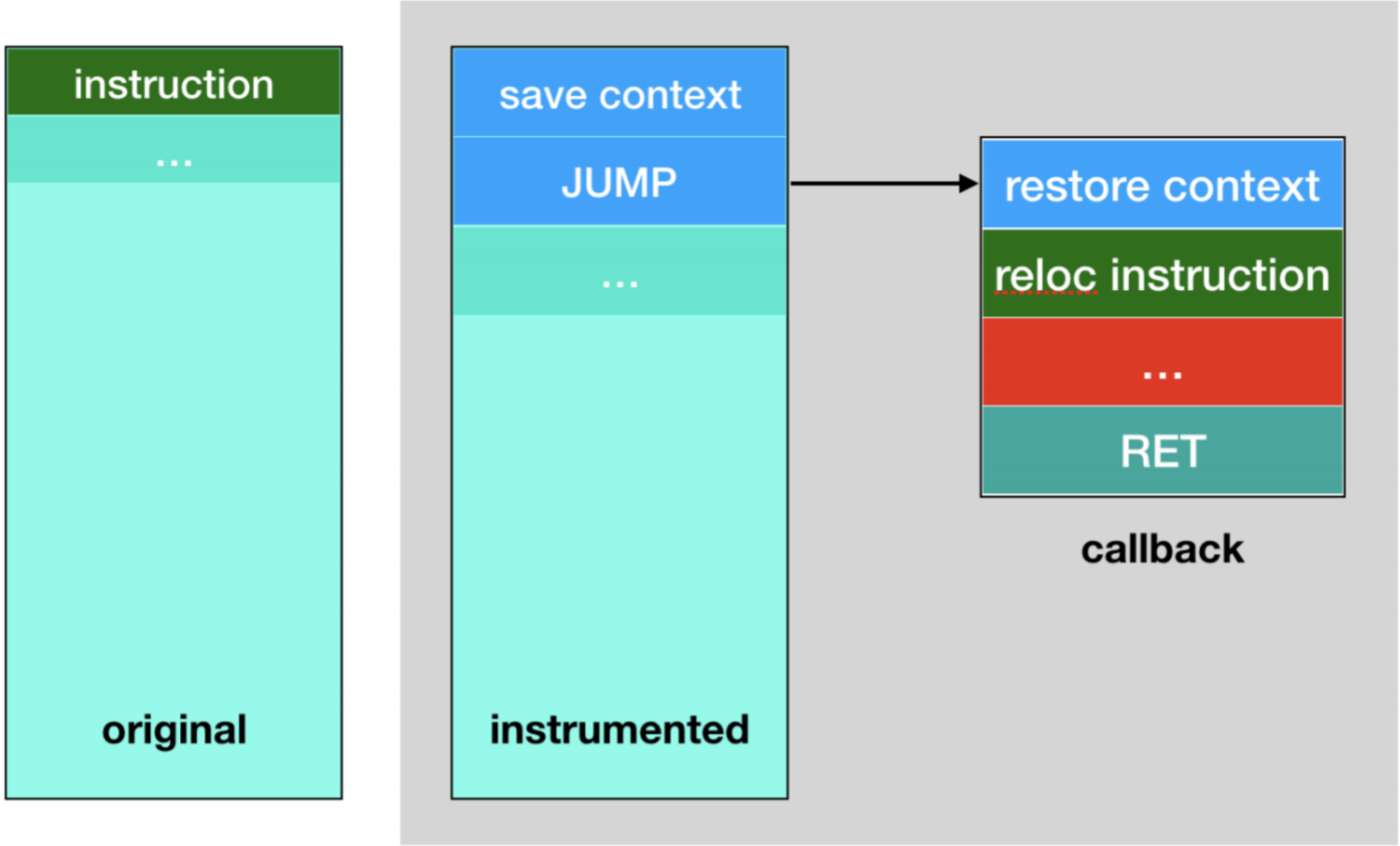
- Branch from original instruction to instrumented code
- Branch to trampoline, or directly to callback
  - Jump-trampoline technique
  - Jump-callback technique
  - Call-trampoline technique
  - Call-callback technique



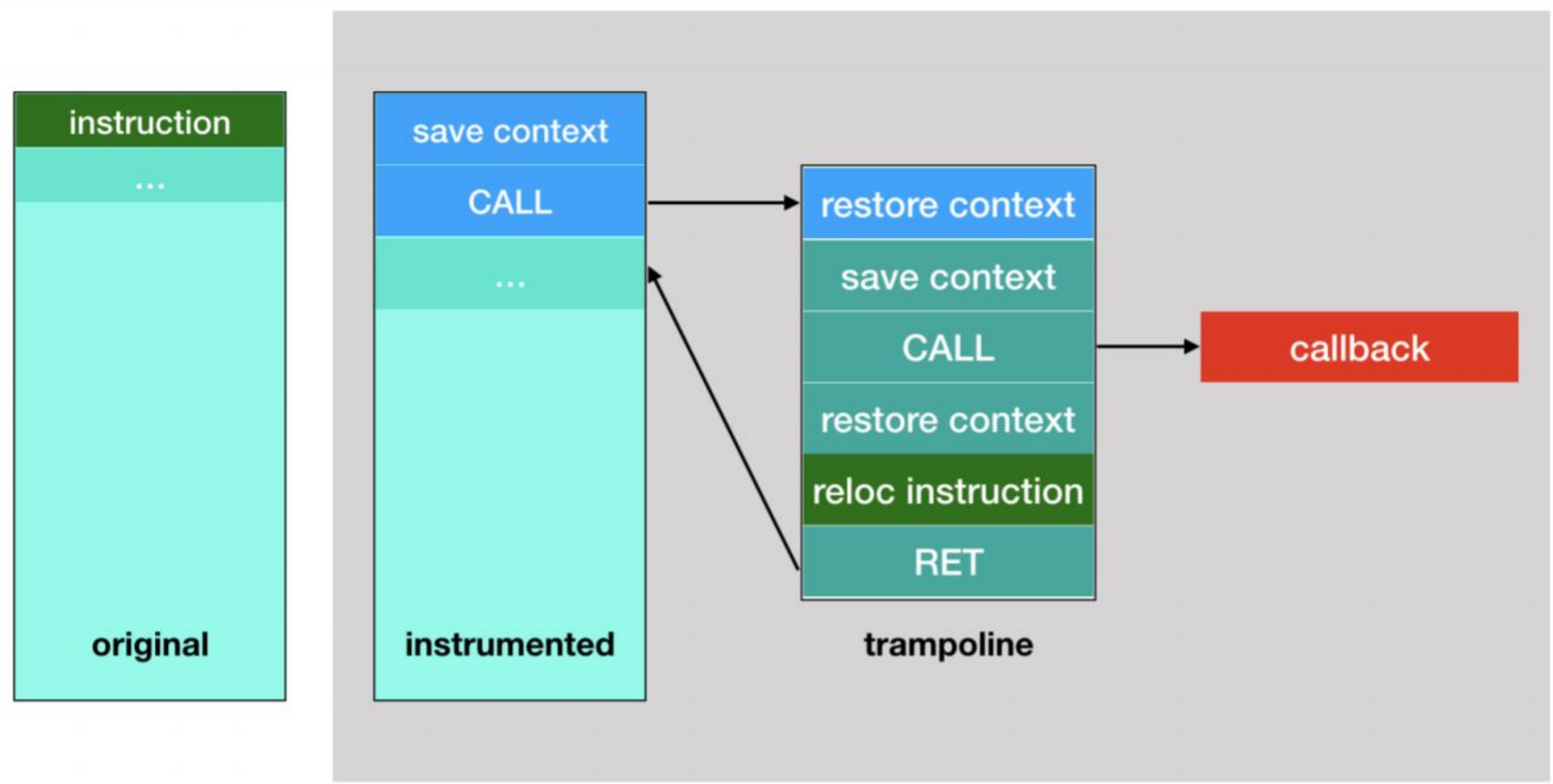
Jump-trampoline Technique



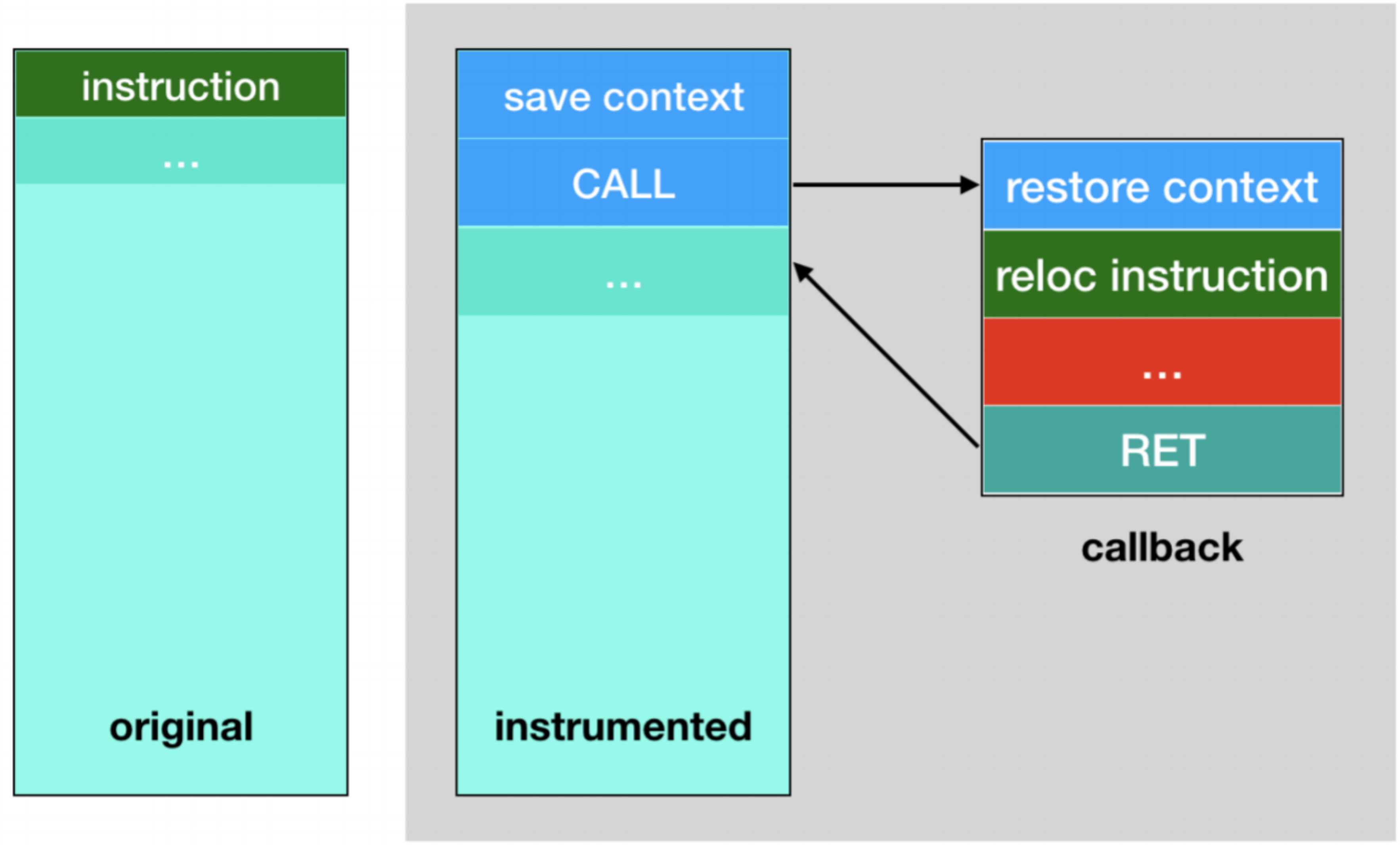
Jump-callback Technique



Call-trampoline Technique



Call-callback Technique



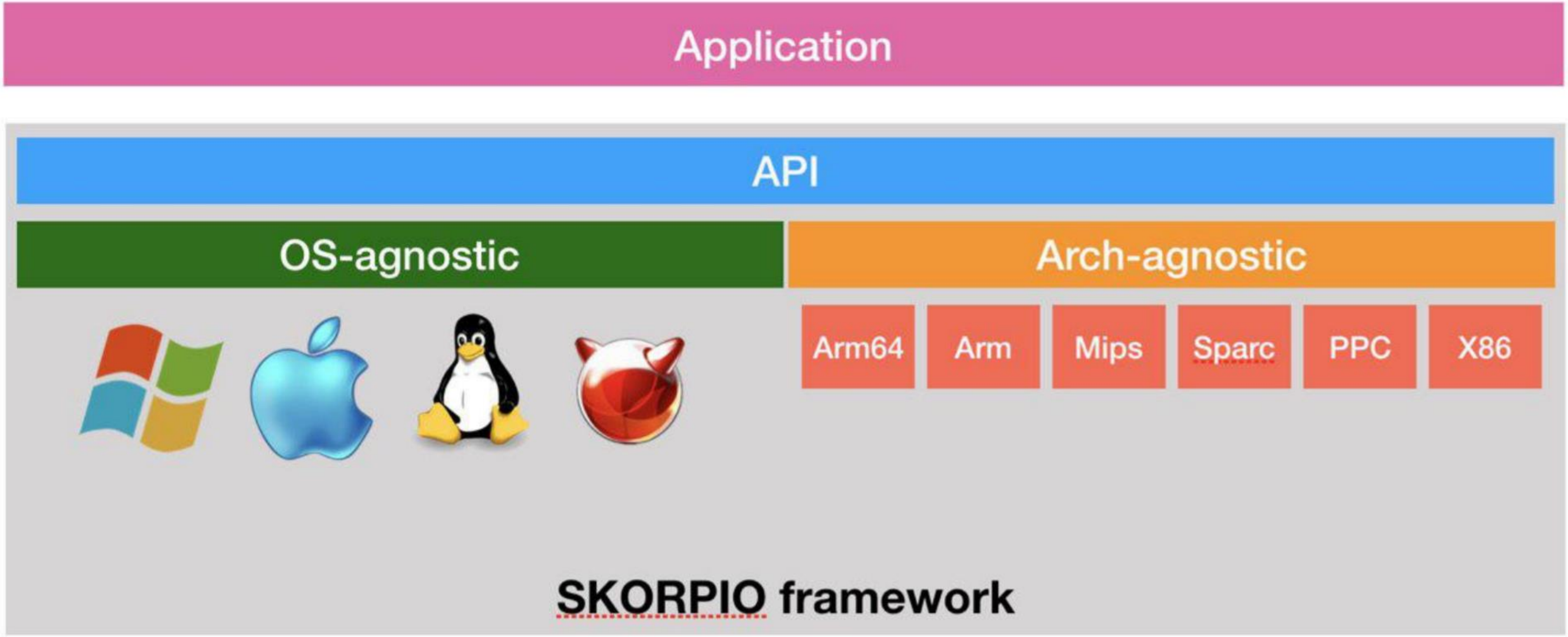
Problems of Existing DBI

- Limited on platform support
- Limited on architecture support
- Limited on instrumentation techniques
- Limited on optimization

SKORPIO Framework

- Low level framework to build applications on top
  - App typically designed as dynamic libraries (DLL/SO/DYLIB)
- Cross-platform-architecture
  - Windows, MacOS, Linux, BSD, etc
  - X86, Arm, Arm64, Mips, Sparc, PowerPC
- Allow all kind of instrumentations
  - Arbitrary address, in any privilege level
- Designed to be easy to use, but support all kind of optimization
  - Super fast (100x) compared to other frameworks, with proper setup
- Support static instrumentation, too!

SKORPIO Architecture



Cross Platform - Memory

- Thin layer to abstract away platform details
- Different OS supported in separate plugin
  - Posix vs Windows
- Trampoline buffer
  - Allocate memory: malloc() vs VirtualAlloc()
  - Memory privilege RWX: mprotect() vs VirtualAlloc()
  - Trampoline buffer as close as possible to code to reduce branch distance
- Patch code in memory
  - Unprotect -> Patch -> Re-protect
  - mprotect() vs VirtualProtect()

Cross architecture - Save/Restore Context

- Save memory/registers modified by initial branch & callback
- Keep the code size as small as possible
- Depend on architecture + mode
  - X86-32: PUSHAD; PUSHFD & POPFD; POPAD
  - X86-64 & other CPUs: no simple instruction to save all registers :-(
    - Calling convention: cdecl, optlink, pascal, stdcall, fastcall, safecall, thiscall, vectorcall, Borland, Watcom
    - SystemV ABI vs Windows ABI
- Special API to customize code to save/restore context

Cross Architecture - Callback argument

- Pass user argument to user-defined callback
- Depend on architecture + mode & calling convention
  - SysV/Windows x86-32 vs x86-64
    - Windows: cdecl, optlink, pascal, stdcall, fastcall, safecall, thiscall, vectorcall, Borland, Watcom
  - X86-64: "mov rcx, <value>" or "mov rdi, <value>. Encoding depends on data value
  - Arm: "ldr r0, [pc, 0]; b .+8; <4-byte-value>"
  - Arm64: "movz x0, <lo16>; movk x0, <hi16>, lsl 16"
  - Mips: "li $a0, <value>"
  - PPC: "lis %r3, <hi16>; ori %r3, %r3, <lo16>"

Cross Architecture - Branch distance

- Distance from hooking place to callback cause nightmare :-(
  - Some architectures have no explicit support for far branching
    - X86-64 JUMP: "push <addr>; ret" or "push 0; mov dword ptr [rsp+4], <addr>" or "jmp [rip]"
    - X86-64 CALL: "push <next-addr>; push <target>; ret"
    - Arm JUMP: "b <addr>" or "ldr pc, [pc, #-4]"
    - Arm CALL: "bl <addr>" or "add lr, pc, #4; ldr pc, [pc, #-4]"
    - Arm64 JUMP: "b <addr>" or "ldr x16, .+8; br x16"
    - Arm64 CALL: "bl <addr>" or "ldr x16, .+12; blr x16; b .+12"
    - Mips JUMP: "li $t0, <addr>; jr $t0"
    - Mips CALL: "li $t0, <addr>; move $t9, $t0; jalr $t0"
    - Sparc JUMP: "set <addr>, %l4; jmp %l4; nop"
    - Sparc CALL: "set <addr>, %l4; call %l4; nop"

Cross Architecture - Branch for PPC

- PPC has no far jump instruction :-(
  - copy LR to r23, save target address to r24, then copy to LR for BLR
  - restore LR from r23 after jumping back from trampoline
  - "mflr %r23; lis %r24, <hi16>; ori %r24, %r24, <lo16>; mtlr %r24; blr"
- PPC has no far call instruction :-(
  - save r24 with target address, then copy r24 to LR
  - point r24 to instruction after BLR, so later BLR go back there from callback
  - "lis %r24, <target-hi16>; ori %r24, %r24, <target-lo16>; mtlr %r24; lis %r24, <ret-hi16>; ori %r24, %r24, <ret-lo16>; blr"

```
SK_INLINE_NO static void bbb_hook(size_t v)
{
    // restore LR from R24
    __asm__("mtlr %r24");

    printf("== in callback, userdata = %zu\n", v);

    return;
}
```

Cross Architecture - Scratch Register

- Scratch registers used in initial branching
  - Arm64, Mips, Sparc & PPC do not allow branch to indirect target in memory
  - Calculate branch target, or used as branch target
  - Need scratch register(s) that are unused in local context
    - Specified by user via API, or discovered automatically by engine

Cross Architecture - Flush Code Cache

- Code patching need to be reflected in i-cache
- Depend on architecture
  - X86: no need
  - Arm, Arm64, Mips, PowrPC, Sparc: special syscalls/instructions to flush/invalidate i-cache
  - Linux/GCC has special function: cacheflush(begin, end)

Code Boudary & Relocation

- Need to extract instructions overwritten at instrumentation point
  - Determine instruction boundary for X86
  - Use Capstone disassembler
- Need to rewrite instructions to work at relocated place (trampoline)
  - Relative instructions (branch, memory access)
  - Use Capstone disassembler to detect instruction type
  - Use Keystone assembler to recompile

Code Analysis

- Avoid overflow to next basic block
  - Analysis to detect if basic block is too small for patching
- Reduce number of registers saved before callback
- Registers to be choosen as scratch registers

Customize on Instrumentation

- API to setup calling convention
- User-defined callback
- User-defined trampoline
- User-defined scratch registers
- User-defined save-restore context
- User-defined code to setup callback ars
- Patch hooks in batch, or individual
- User decide when to write/unwrite memory protect

Skorpio Sample C Code

```
Sample for Skorpio engine

--- Original code
BBB code = 0x400ca0, callback = 0x400c80

Hook info:
Hook type:                  2
Hook address:               0x400ca0
Hook callback:              0x400c80
Hook user_data:             0x7b
Hook trampoline addr:       0x7f1aa7911000
Hook trampoline size:       86
Hook trampoline code:       505351525756554150415141524154 9c48c7c77b0000006a00c70424321091a7c74424041a7f00006a00c70424800c4000c39d415c4
15a415941585d5e5f5a595b584883ec08b9800c4000baa00c400068ae0c4000c3
Patch size:                 14
Patched code:               ff2500000000001091a71a7f0000
Hook original code size:    14
Hook original code:         4883ec08b9800c4000baa00c4000


--- Functions with instrumentation now
== inside callback, userdata = 123
BBB code = 0x400ca0, callback = 0x400c80


--- Restored original code, now without instrumentation
BBB code = 0x400ca0, callback = 0x400c80
```

Agenda

Coverage Guided Fuzzer vs Embedded Systems

Emulating Firmware

Skorpio Dynamic Binary Instrumentation

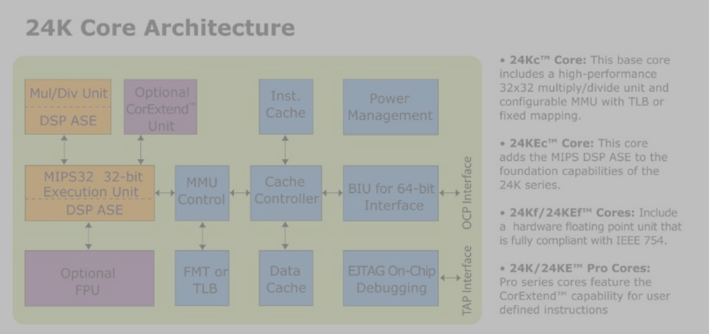
Guided Fuzzer for Embedded

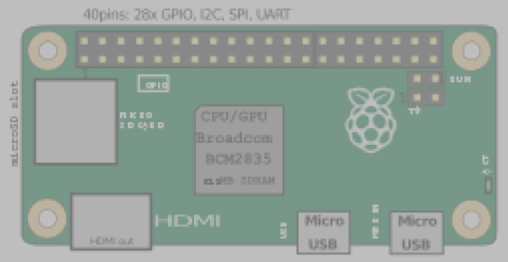
DEMO

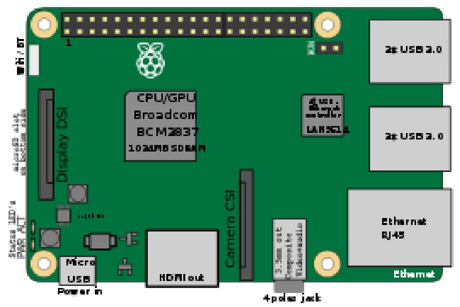
Conclusions

Secret Menu

Issues







Firmware Emulation

- Without built-in shell access for user interaction
- Without developement facilities required for building new tools
  - Compiler
  - Debugger
  - Analysis tools

Skorpio DBI

- Binary only - without source code
  - Existing guided fuzzers rely on source code available
    - Source code is needed for branch instrumentation to feedback fuzzing progress
    - Emulation such as QEMU mode support in AFL is slow & limited in capability
    - Same issue for other tools based on Dynamic Binary Instrumentation

Lack Support for Embedded

- Most fuzzers are built for X86 only
  - Embedded systems based on Arm, Arm64, Mips, PPC
- Existing DBIs are poor for non-X86 CPU
  - Pin: Intel only
  - DynamoRio: experimental support for Arm

Fuzzer Features

- Built on top of AFL fuzzer
- Support closed-source binary for all platforms & architectures
  - Use Skorpio DBI to support all popular embedded CPUs
- Support selective binary fuzzing
- Support persistent mode
- Other enhanced techniques
  - Symbolic Execution to guide fuzzer forward
  - Combine with static analysis for smarter/deeper penetration

Fuzzer Design

- Pure software-based
- Cross-platform/architecture
  - Native compiled on embedded systems
- Binary support
  - Full & selected binary fuzzing + Persistent mode
- Fast & stable
  - Stable & support all kind of binaries
  - Order of magnitude faster than DBI/Emulation approaches

Fuzzer Implementation

- Reuse AFL fuzzer - without changing its core design
- AFL-compatible instrumentation
- Static analysis on target binary beforehand
- Inject Skorpio hooks into selected area in target binary at runtime
- At runtime, hook callbacks update execution context in shared memory, like how source-code based instrumentation do
- Near native execution speed, ASLR / threading compatible

Fuzzer Instrumentation

- LD_PRELOAD to dynamically inject instrumentation
  - Take place before main program runs
  - Linux: shared object file (.so)
- Inject hooks at SO initialisation time
  - Can be 10k hooks, so must do as quickly as possible
- Inject forkserver at program entry-point, or at user-defined point

Agenda

Coverage Guided Fuzzer vs Embedded Systems

Emulating Firmware

Skorpio Dynamic Binary Instrumentation

Guided Fuzzer for Embedded

DEMO

Conclusions

Secret Menu

bug disclosed in geekpwn 2018, shanghai

Web Cam Buffer Overflow

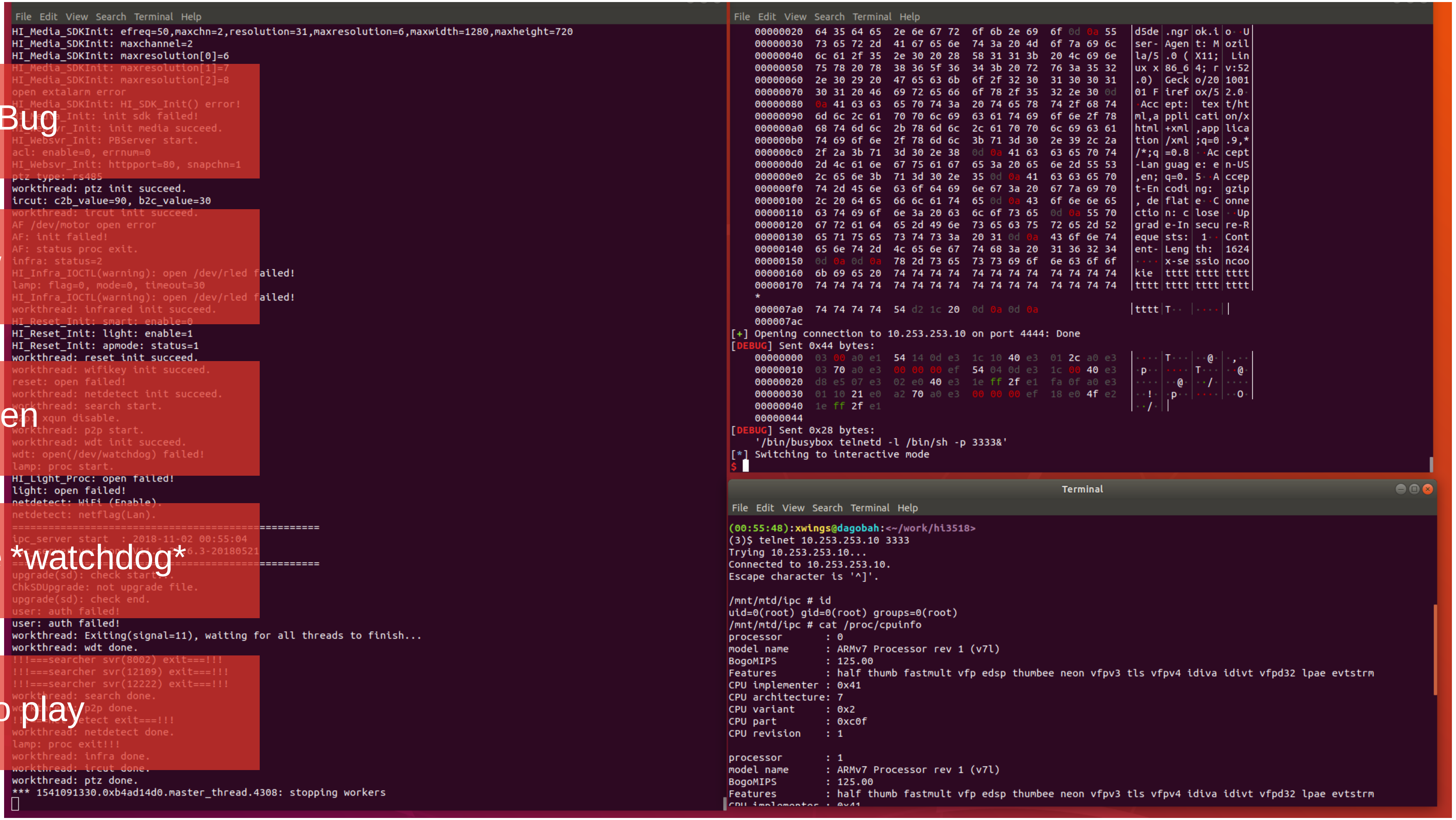
Pre Authentication Bug

Buffer Overflow

Address Overwritten

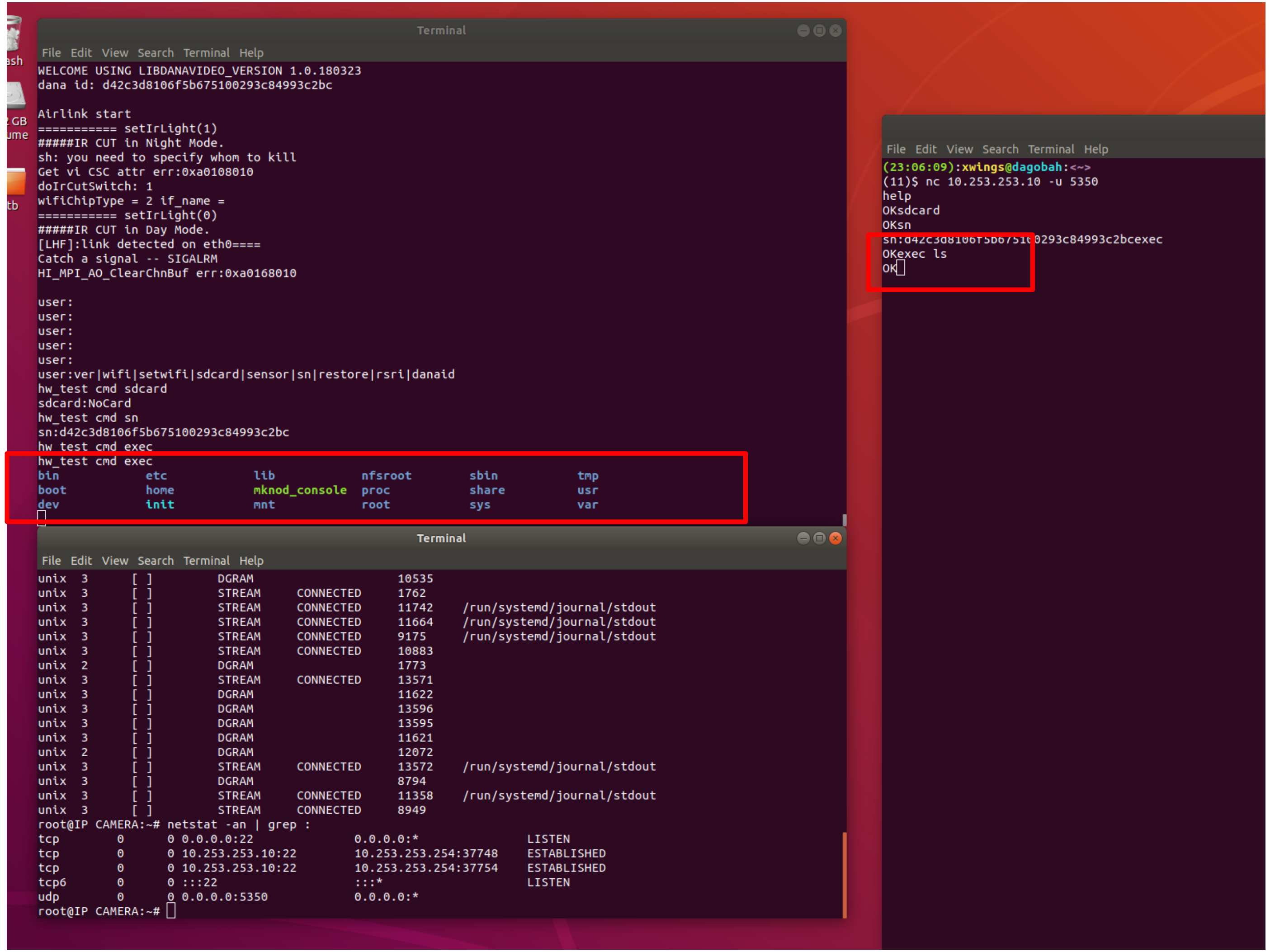
Debug is almost Impossible *watchdog*

Emulation comes into play



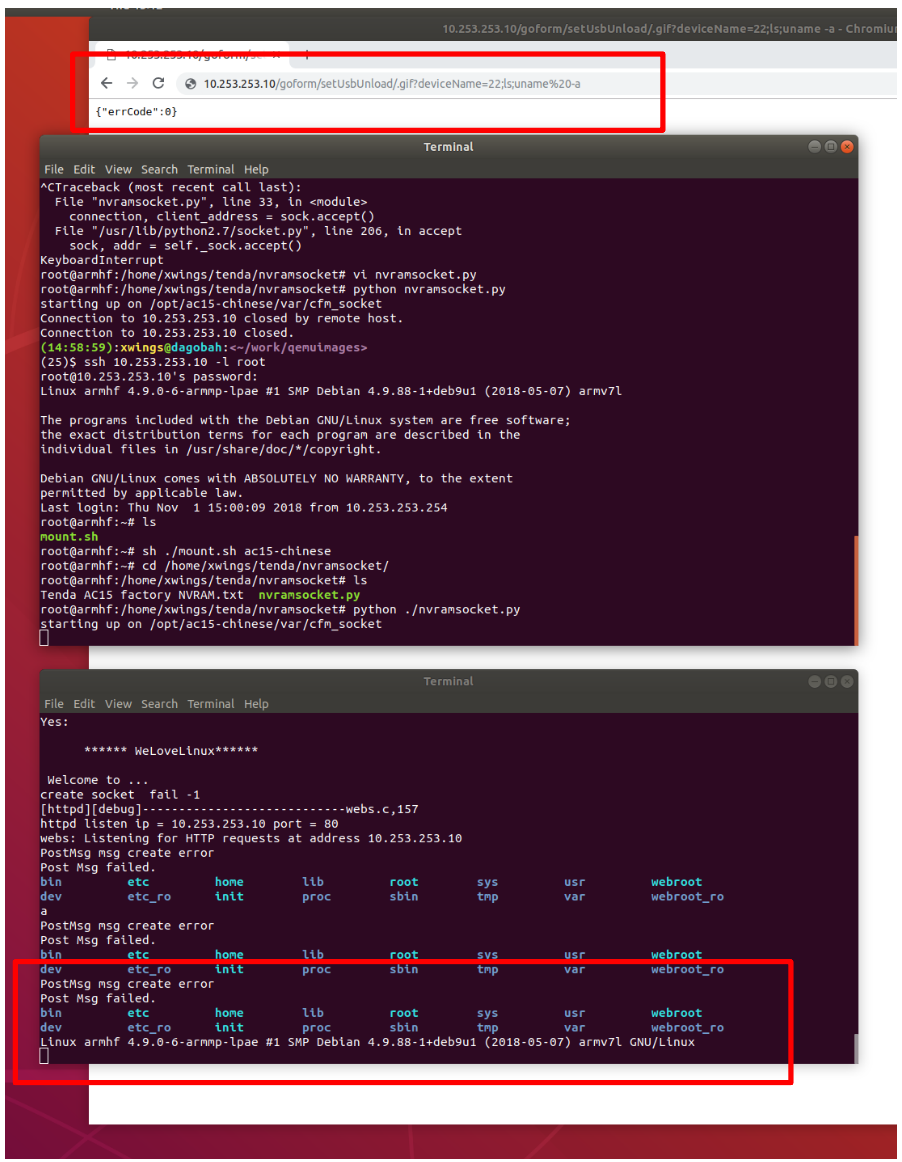
IoT with UDP Access

Web Cam with Motor



Command Execution Injection

Chinese based WiFi Router



Agenda

Coverage Guided Fuzzer vs Embedded Systems

Emulating Firmware

Skorpio Dynamic Binary Instrumentation

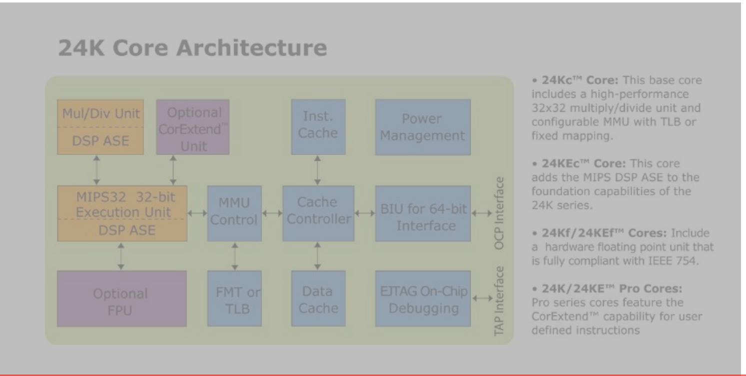
Guided Fuzzer for Embedded

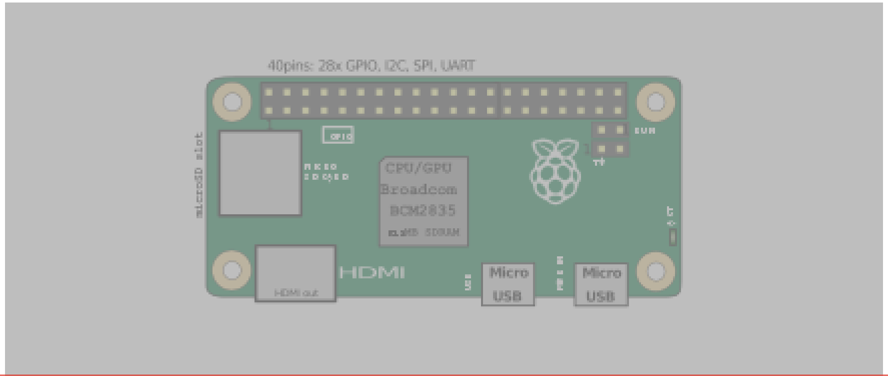
DEMO

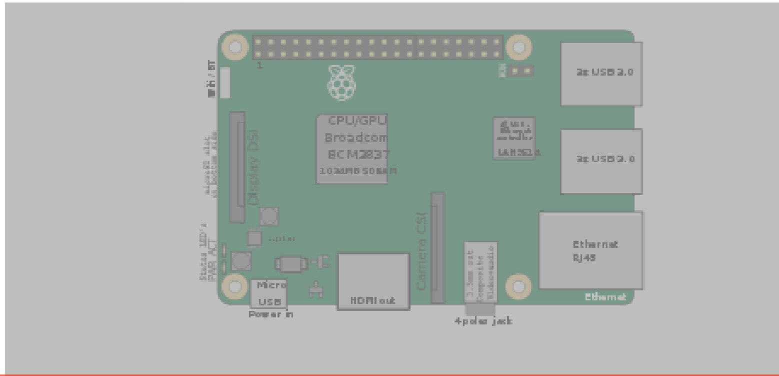
Conclusions

Secret Menu

Issues







Firmware Emulation	Skorpio DBI	Guided Fuzzer for Embedded

Firmware Emulation

- Without built-in shell access for user interaction
- Without developement facilities required for building new tools
  - Compiler
  - Debugger
  - Analysis tools

Skorpio DBI

- Binary only - without source code
  - Existing guided fuzzers rely on source code available
    - Source code is needed for branch instrumentation to feedback fuzzing progress
    - Emulation such as QEMU mode support in AFL is slow & limited in capability
    - Same issue for other tools based on Dynamic Binary Instrumentation

Guided Fuzzer for Embedded

- Most fuzzers are built for X86 only
  - Embedded systems based on Arm, Arm64, Mips, PPC
- Existing DBIs are poor for non-X86 CPU
  - Pin: Intel only
  - DynamoRio: experimental support for Arm

Conclusions

- We built our smart guided fuzzer for embedded systems
  - Emulate firmware
  - Cross platforms/architectures
  - Binary-only support
  - Fast + stable
  - Found real impactful bugs in complicated software

Agenda

Coverage Guided Fuzzer vs Embedded Systems

Emulating Firmware

Skorpio Dynamic Binary Instrumentation

Guided Fuzzer for Embedded

DEMO

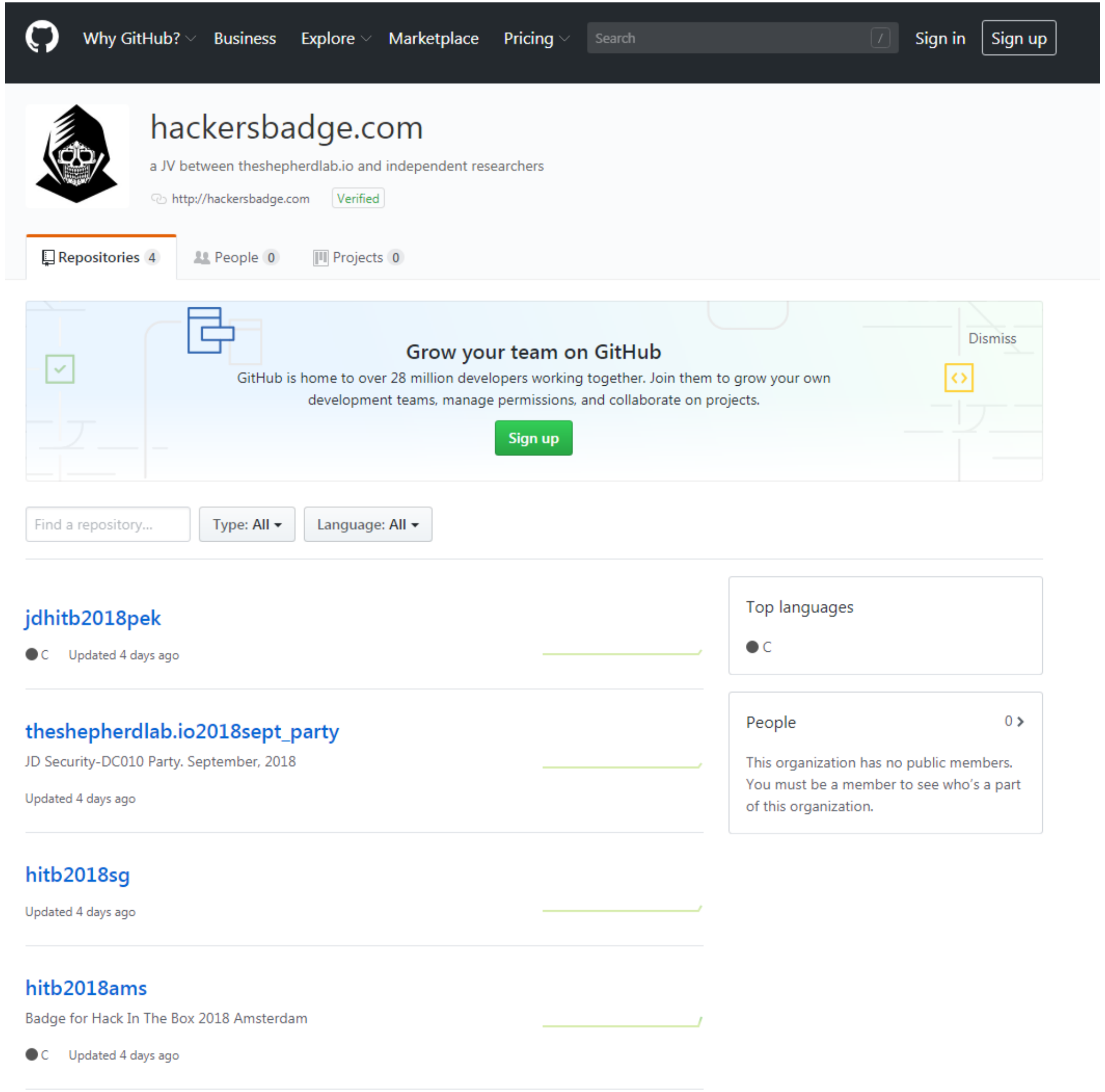
Conclusions

Secret Menu

Capstone 4.0

hackersbadge.com

- Started 2013
- ~160 Contributors
- World Class Disassembler, Industrial Standard
- Used by almost all reverse engineering tools
- Foundation for 400+ opensource/public projects
- Current Release 3.0.5
- In version 3 since 2014
- Dec 2018, Capstone 4.0
- Why take us so long



Questions

Virtualizing IoT
with Code Coverage Guided Fuzzing

NGUYEN Anh Quynh, aquynh -at- gmail.com
KaiJern LAU, kj -at- theshepherd.io