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

Reverse Engineer Badge Maker
Founder of hackersbadge.com, RE &amp; CTF fan

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

- Reversing Binary
- Reversing IoT Devices
- Part Time CTF player
- Part 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 Orffice
- 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 crafted input data to the program under test
  - Monitor for errors like crash/hang/memory leaking
  - Focus more on exploitable errors like memory corruption, info leaking
- Maximize code coverage to find bugs
- Blackbox fuzzing
- Whitebox fuzzing
- Graybox fuzzing, or **Coverage Guided Fuzzing**
Coverage-guided Fuzzer

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

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

**Restricted System**
- 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 available
- 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

**Closed System**
- 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

**Lack Support for Embedded**
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
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

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

If we need more?
1. RCE 2. Fuzz
The Easy Way
Complete Kit to Success

MIPS
How Many Dev Board

ARM

AARCH64
Classic LIBC Issue
The Hackers Way: Virtualization
More Resources = More Power

Processor
- Normally 1-2 Core

RAM
- Normally 256MB/512MB

FLASH
- Normally 8MB/16MB/32MB/256MB

Most Important, we got apt-get
Objectives
Since only one binary, do we really need qemu-system or just use good old qemu-static
Booting Up
Current Solution

Leaving squashfs and going into a unknown world
Old vs New

argument: running new or old distro + kernel + hypervisor

2016 way

script to boot mips
chroot is easy (still hardware dependent), but we will have issue with tools
Classic Case: File Not Found
Now You See It

We found you

```
root@rpi3:/opt/
```

```
../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]=22e2854c58b1814825b95c8a103ac658d371f5b0, stripped
```

We Missed You

```
chdir("/")
```

```
execve("/bin/bash", ["/bin/bash", "-i"], 0xfffffca14f650 /* 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/

```

We Missed You

```
chdir("/")
execve("/bin/bash", ["/bin/bash", ",", 0xffffffca14f650 /* 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)
write2(2, "chroot: ", chroot: ) = 8
write2(2, "failed to run command '/bin/bash'", 33) failed to run command '/bin/bash' = 33
write2(2, ": No such file or directory", 27: No such file or directory = 27
write2(2, "\n", 0)
close(1) = 0
close(2) = 0
exit_group(127) = ?
```
The missing .SO and binary Issue
Out from chroot, we need feeding

Feeding all the required so and binary with “ln –s”
Out from chroot, we need feeding

“segfault” without clear error. strace come to rescue
The Secretive NVRAM
Dark Side of NVRAM

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

Dark Side of the main process, we ignore and can't to next step.
A Fake NVRAM

IF interactor is the medium, can we fake it?

Custom Interactor
Wireless Device
Faking *wpa_supplicant*

```bash
[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_ctrl_808
[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_moni_808
[WIFI_MW] wpaCtrlOpen: bind(), bindRet = 0.
```

making eth0 looks like wlan0 works too
Everything Things Else Fail
jmp, cbz, cbnz and Friends

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

> 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
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
making eth0 looks like wlan0 works too
DBI Techniques

- **Just-in-Time translation**
  - Transparently translate & execute code at runtime
    - Perform on IR: Valgrind
    - Perform directly on native code: DynamoRio
  - Better control on code executed
  - Heavy, super complicated in design & implementation

- **Hooking**
  - Lightweight, much simpler to design & implement
  - Less control on code executed & need to know in advance where to instrument
Hooking Mechanisms - Inline

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

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

![Diagram of Detour Injection](attachment:detour-diagram.png)
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

![Diagram of detour injection mechanism]

Original code:
1 2 3 4

Detour instrumentation:
1 2 3 4

A
B
Jump-trampoline Technique

- instruction
  ...

- save context
  JUMP
  ...

- restore context
  save context
  CALL
  restore context
  reloc instruction
  JUMP

- callback
Jump-callback Technique

- saving the context
- jumping to the callback
- restoring the context
Call-trampoline Technique

instruction

... 

original

save context

CALL

... 

instrumented

restore context

save context

CALL

restore context

reloc instruction

RET

callback
Call-callback Technique

- instruction
- save context
- CALL
- restore context
- reloc instruction
- RET
- callback

Making eth0 looks like wlan0 works too.
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

making eth0 looks like wlan0 works too
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()
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
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>"
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 "ld 
  - Arm64 CALL: "bl <addr>" or "ld x16, .+8; br x16"
  - 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"

```c
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;
}
```
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
- 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 ars
- Patch hooks in batch, or individual
- User decide when to write/unwrite memory protect
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: 505351525756554150415415241549c48c7c77b0000006a00c70424321091a7c74424041a7f00006a00c70424800c4000c39d415c415a415941585d5e5f5a595b584883ec08b9800c4000baa0c400068ae0c4000c3
Patch size: 14
Patched code: ff2500000000001091a71a7f0000
Hook original code size: 14
Hook original code: 4883ec08b9800c4000baa0c4000

--- 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

Skorpio DBI

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
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
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

```
[16:55:44]
(31)  username = 
Linux xiangyu 4.15.0-34-generic #37-Ubuntu SMP Mon Aug 27 13:21:48 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
(52)  username = 
Linux xiangyu 4.15.0-34-generic #37-Ubuntu SMP Mon Aug 27 13:21:48 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
(52)  telnet 10.253.253.10 4444
Trying 10.253.253.10...
telnet: Unable to connect to remote host: Connection refused
(52)  telnet 10.253.253.10 4444
Trying 10.253.253.10...
Connected to 10.253.253.10.
Escape character is '^]'.
```

```
(52)  quit
Connection closed by foreign host.
(52)  cat exp_route_intermaional.py | grep 4444
```

```
(52)  python exp_route_intermaional.py
```

```
Traceback (most recent call last):
  File "exp_route_intermaional.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 1226, in http_open
    return self._open(req, http_response, timeout)
  File "/usr/lib/python2.7/urllib2.py", line 1201, in do_open
    r = http_response.getrinfo(True)
  File "/usr/lib/python2.7/urllib.py", line 1121, in getresponse
    raise BadStatusLine(line)
http.client.BadStatusLine: '':
```

```
(56)  telnet 10.253.253.10 4444
```

```
Trying 10.253.253.10...
Connected to 10.253.253.10.
Escape character is '^]'.
```

```
# username = 
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

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
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