

TALOS



Finding Vulns in Embedded Systems

Carlos Pacho

\$whoami

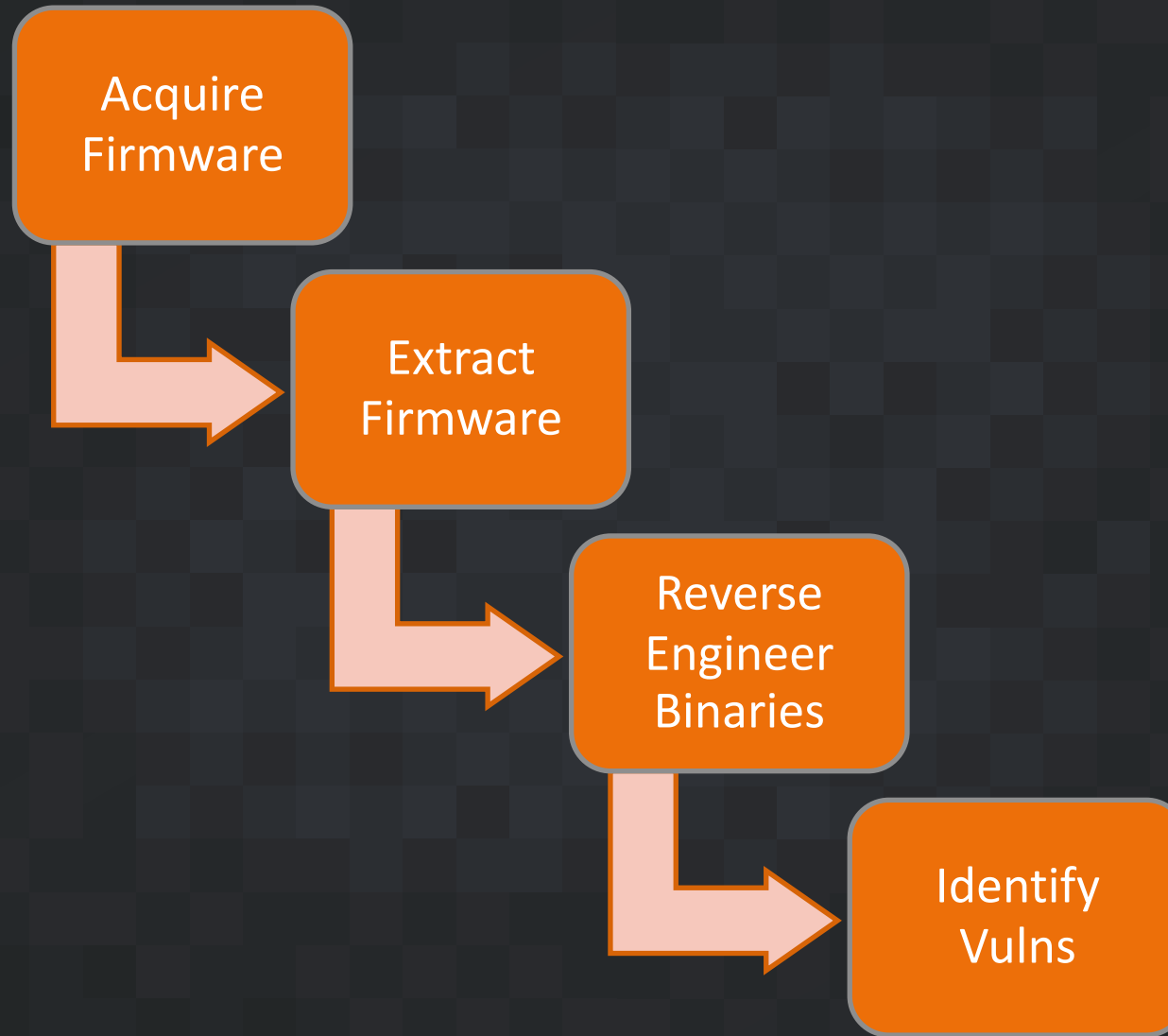
- Carlos Pacho
- Been with Cisco Talos for the past 5 years
- Security researcher for the ARES (Advanced Research Embedded System) team
 - We built an ICS kegerator
- Embedded system reverse engineering focused



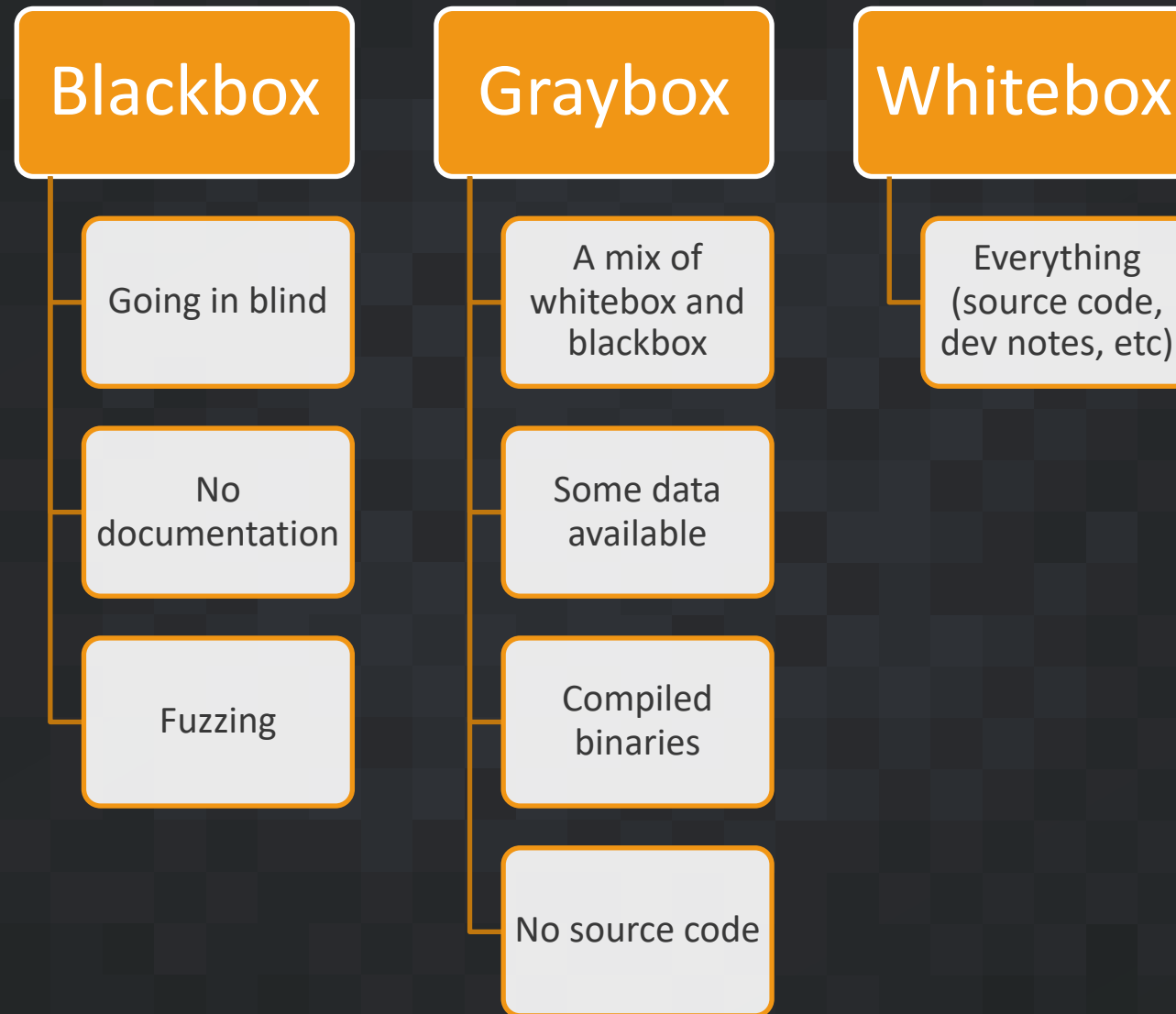
Why Find Vulnerabilities

- Protect our customers
- Make the internet a safer place
- Detection content
- Notify affected vendor of newly discovered vulnerability
 - Vendor (hopefully) patches the vulnerability

Vulnerability Discovery Overview



Types of Vulnerability Assessments





Firmware

Firmware Overview

- Firmware typically contains a device's file system, which has:
 - Binaries
 - User account info (/etc/passwd, /etc/shadow ...)
 - Notes/test scripts from the developers
 - Device startup information
- Acquire firmware from
 - Vendor websites
 - Off of the hardware
 - Hopefully not encrypted



Extracting Firmware

Grab the
firmware
image

- Download from vendor
- Pull off hardware

Identify data
in the
firmware

- Binwalk
- Radare2 (r2) / hexdump
- Strings

Carve out the
relevant data

- Binwalk -eM
- dd

Extracting Firmware

```
[cpacho@kali target]$ binwalk firmware.bin
```

DECIMAL	HEXADECIMAL	DESCRIPTION

88	0x58	uImage header, header size: 64 bytes, header CRC: 0x6F60D003, created: 2017-09-15 09:44:56, image size: 1103372 bytes, Data Address: 0x80002000, Entry Point: 0x80006230, data CRC: 0xF37B126C, OS: Linux, CPU: MIPS, image type: OS Kernel Image, compression type: lzma, image name:
152	0x98	LZMA compressed data, properties: 0x5D, dictionary size: 16777216 bytes, uncompressed size: 3236352 bytes
1103560	0x10D6C8	Squashfs filesystem, little endian, version 4.0, compression: lzma, size: 5000191 bytes, 713 inodes, blocksize: 131072 bytes, created: 2017-09-15 09:50:49
6104812	0x5D26EC	Squashfs filesystem, little endian, version 4.0, compression: lzma, size: 3020731 bytes, 324 inodes, blocksize: 131072 bytes, created: 2017-09-15 09:50:49

Reverse Engineering Binaries

Assembly vs Source Code

```
#include<stdio.h>

int main()
{
    printf("Hello World");
}
```

Source Code

x86 Assembly (compiled binary)

```
(fcn) sym.main 28
| sym.main ();
| ; DATA XREF from 0x0000055d (entry0)
| 0x0000064a 00:0000 55 push rbp
| 0x0000064b 00:0000 4889e5 mov rbp, rsp
| 0x0000064e 00:0000 488d3d9f0000. lea rdi, str.Hello_World ; 0x6f4 ; "Hello World"
| 0x00000655 00:0000 b800000000 mov eax, 0
| 0x0000065a 00:0000 e8d1feffff call sym.imp.printf ; int printf(const char *format)
| 0x0000065f 00:0000 b800000000 mov eax, 0
| 0x00000664 00:0000 5d pop rbp
| 0x00000665 00:0000 c3 ret
```

Reverse Engineering the Binaries

- Why?
 - Find out what a compiled binary is doing
- Identify important binaries
 - Network communication
 - Run at startup
 - Custom bins
- Tools
 - Radare2, IDA, qemu, gdb

Static vs Dynamic Analysis

- The main difference is code execution
 - Static: No code execution
 - Dynamic: Execute the code and see what happens
- Useful tools for dynamic analysis
 - GDB, QEMU, VirtualBox, sfuzz

Finding Vulns

Vuln Example: Command Injections

- These bugs occur when a programmer makes a unsafe call to system
 - They allow for commands to be passed to system by using shell metacharacters
 - ` & | \$() etc
- I've seen them in
 - Web servers (ping pages, user management, network config)
 - Configuration tools
- Find them by
 - Finding all calls to system
 - Back trace what is being passed to system
 - See if you can control what is being passed to system

Vulnerable Program

```
[cpacho@somewhere target]$ ./hashfinder
**MD5 Hash finder**
Give me a file please...
README.txt
file: README.txt

54a3eaba0dceea4c5cbe4b2d26506fc9  README.txt
```

Vulnerable Program

```
[cpacho@somewhere target]$ ./hashfinder
```

```
**MD5 Hash finder**
```

```
Give me a file please...
```

```
README.txt
```

```
file: README.txt
```

```
54a3eaba0dceea4c5cbe4b2d26506fc9  README.txt
```

```
[cpacho@somewhere target]$ ./hashfinder
```

```
**MD5 Hash finder**
```

```
Give me a file please...
```

```
`cat /etc/passwd`
```

```
HACKER DETECTED
```

Vulnerable Program

```
[cpacho@somewhere target]$ ./hashfinder
```

```
**MD5 Hash finder**
```

```
Give me a file please...
```

```
README.txt
```

```
file: README.txt
```

```
54a3eaba0dceea4c5cbe4b2d26506fc9  README.txt
```

```
[cpacho@somewhere target]$ ./hashfinder
```

```
**MD5 Hash finder**
```

```
Give me a file please...
```

```
`cat /etc/passwd`
```

```
HACKER DETECTED
```

```
[cpacho@somewhere target]$ ./hashfinder
```

```
**MD5 Hash finder**
```

```
Give me a file please...
```

```
$(whoami)
```

```
file: $(whoami)
```

```
md5sum: cpacho: No such file or directory
```

Vulnerable Program

```
10  int main()
11  {
12      char userinput[256];
13      char buf[256];
14      printf("**MD5 Hash finder**\n");
15      printf("Give me a file please...\n");
16      fgets (userinput, 256, stdin);
17      char * pch;
18      pch = strstr(userinput, "\x60"); // Looking for `
19      if (pch) {
20          printf("HACKER DETECTED\n");
21          return 0;
22      } else {
23          printf("file: %s\n", &userinput);
24          snprintf(buf, sizeof(buf), "md5sum %s", userinput);
25          system(buf);
26          return 0;
27      }
28  }
```

← Taking user input

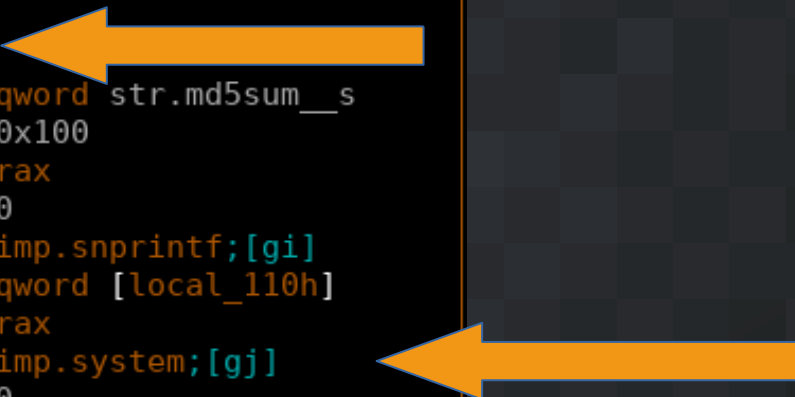
← Input filtering

← System is called

`system("md5sum $(whoami)")`

Finding a Command Injection with Radare2

```
[0x8df] ;[gd]
; CODE XREF from 0x000008ca (main)
00:0000 lea rax, qword [local_210h]
00:0000 mov rsi, rax
; 0xall
; "file: %s\n"
00:0000 lea rdi, qword str.file:__s
00:0000 mov eax, 0
00:0000 call sym.imp.printf;[gh]
00:0000 lea rdx, qword [local_210h]
00:0000 lea rax, qword [local_110h]
00:0000 mov rcx, rdx
; 0xalb
; "md5sum %s"
00:0000 lea rdx, qword str.md5sum__s
00:0000 mov esi, 0x100
00:0000 mov rdi, rax
00:0000 mov eax, 0
00:0000 call sym.imp.snprintf;[gi]
00:0000 lea rax, qword [local_110h]
00:0000 mov rdi, rax
00:0000 call sym.imp.system;[gj]
00:0000 mov eax, 0
```



Impact of Command Injection Vulns

- Malicious commands can be executed
- They can be prevented by properly filtering input to system
 - Make sure no shell metacharacters make it to system
 - Never trust user input

Recap



Acquire Firmware



Extract Firmware



Reverse Binaries



Find Vulns