

Hiding @ Depth:

Exploring & Subverting NAND Flash memory

Josh 'm0nk' Thomas

(A DARPA CFT Project by MonkWorks, LLC)

RIP 4.1.13 - Long Live CFT

Thx Mudge

My Path, And You Can Too!

```
$ cat /proc/partitions
major minor #blocks name
31 0 409600 mtdblock0
31 1 6144 mtdblock1
31 2 103936 mtdblock2
31 3 430080 mtdblock3
179 0 7778304 mmcblk0
179 1 7777280 mmcblk0p1
```

```
$ cat /proc/mtd
dev: size erasesize name
mtd0: 190000000 000200000 "system"
mtd1: 006000000 000200000 "appslog"
mtd2: 065800000 000200000 "cache"
mtd3: 1a4000000 000200000 "userdata"
```



My Path, And You Can Too!

- Kernel Modules: Side Loading Fun!
- Sure, I'll be a "test" case

```
<base kernel source>/kernel/drivers/mtd/tests/
```

```
obj-$(CONFIG_MTD_TESTS) += nandx_find_simple.o
obj-$(CONFIG_MTD_TESTS) += nandx_find_complex.o
obj-$(CONFIG_MTD_TESTS) += nandx_hide.o
obj-$(CONFIG_MTD_TESTS) += mtd_oobtest.o
obj-$(CONFIG_MTD_TESTS) += mtd_pagetest.o
obj-$(CONFIG_MTD_TESTS) += mtd_readtest.o
obj-$(CONFIG_MTD_TESTS) += mtd_speedtest.o
obj-$(CONFIG_MTD_TESTS) += mtd_stresstest.o
obj-$(CONFIG_MTD_TESTS) += mtd_subpagetest.o
obj-$(CONFIG_MTD_TESTS) += mtd_torturetest.o
obj-$(CONFIG_MTD_TESTS) += mtd_erasepart.o
```

My Path, And You Can Too!

- Almost everything I do is simply calling the API in the wrong order
 - The I exception is the OOB write
- Path to Winning?
 - Pick a block and wipe it
 - Cover the entire block in 0xDEADBEEF
 - Mark the Block as “Bad”
 - 0x00 out the OOB in the case of Sony
 - Watch the reboot from collision!



nandx_hide.c

```
7530 ▶ /*  */
7539 static void nandx_file_injector(int blockLocation, void *bufferToWrite)
7540 ▼ {
7541 ▶ /*  */
7554
7555 //TODO: Grab and check return values here!!!!
7556
7557 ▶ /*  */
7564
7565 int err = 0;
7566
7567 //Moves all data out of the target block (no, it really doesn't)
7568 nandx_move_data_from_block( blockLocation );
7569
7570 //Erases the targeted block
7571 nandx_erase_block( blockLocation );
7572
7573 //Injects our buffer directly into the block
7574 nandx_buffer_write_to_block( blockLocation, bufferToWrite );
7575
7576 //Marks the target block as bad
7577 err = nandx_mark_bad_framework( blockLocation );
7578 ▼ if( !err ){
7579     printk(PRINT_PREF "First attempt at marking %d bad failed, going manual\n",
... blockLocation);
7580     err = nandx_mark_bad_manual( blockLocation );
7581     }
7582
7583     }
```

nandx_hide.c

```
7138 ▶ /*  */
7147 static int nandx_mark_bad_framework(int blockLocation)
7148 ▼ {
7149 ▶ /*  */
7168 int ret;
7169 loff_t addr = blockLocation * mtd->erasesize;
7170
7171 printk(PRINT_PREF "Marking the block %d as BAD\n", blockLocation);
7172
7173 ret = mtd->block_markbad(mtd, addr);
7174 if (ret)
7175     printk(PRINT_PREF "Success - block %d has been marked bad\n", blockLocation);
7176 else
7177     printk(PRINT_PREF "Failure - Why U no mark block %d as bad?\n", blockLocation);
7178
7179 return ret;
7180
7181 ◀ }
```

nandx_hide.c

```
7183 ▶ /*  */
7193 static int nandx_mark_bad_manual(int blockLocation)
7194 ▼ {
7195 ▶ /*  */
7219
7220 int ret;
7221 loff_t ofs = blockLocation * mtd->erasesize;
7222
7223 // THIS CALL IS THE ENTIRE MAGIC OF NANDX-HIDE
7224 ret = msm_nand_block_markbad(mtd, ofs);
7225
7226 if(ret)
7227     printk(PRINT_PREF "We call into the driver and make %d go away.\n", blockLocation);
7228 else
7229     printk(PRINT_PREF "Odd.. even a RAW write on the 00B doesn't kill block: %d\n",
... blockLocation);
7230     return ret;
7231 └ }
```

BadUSB — On accessories that turn evil

Karsten Nohl <nohl@srlabs.de>

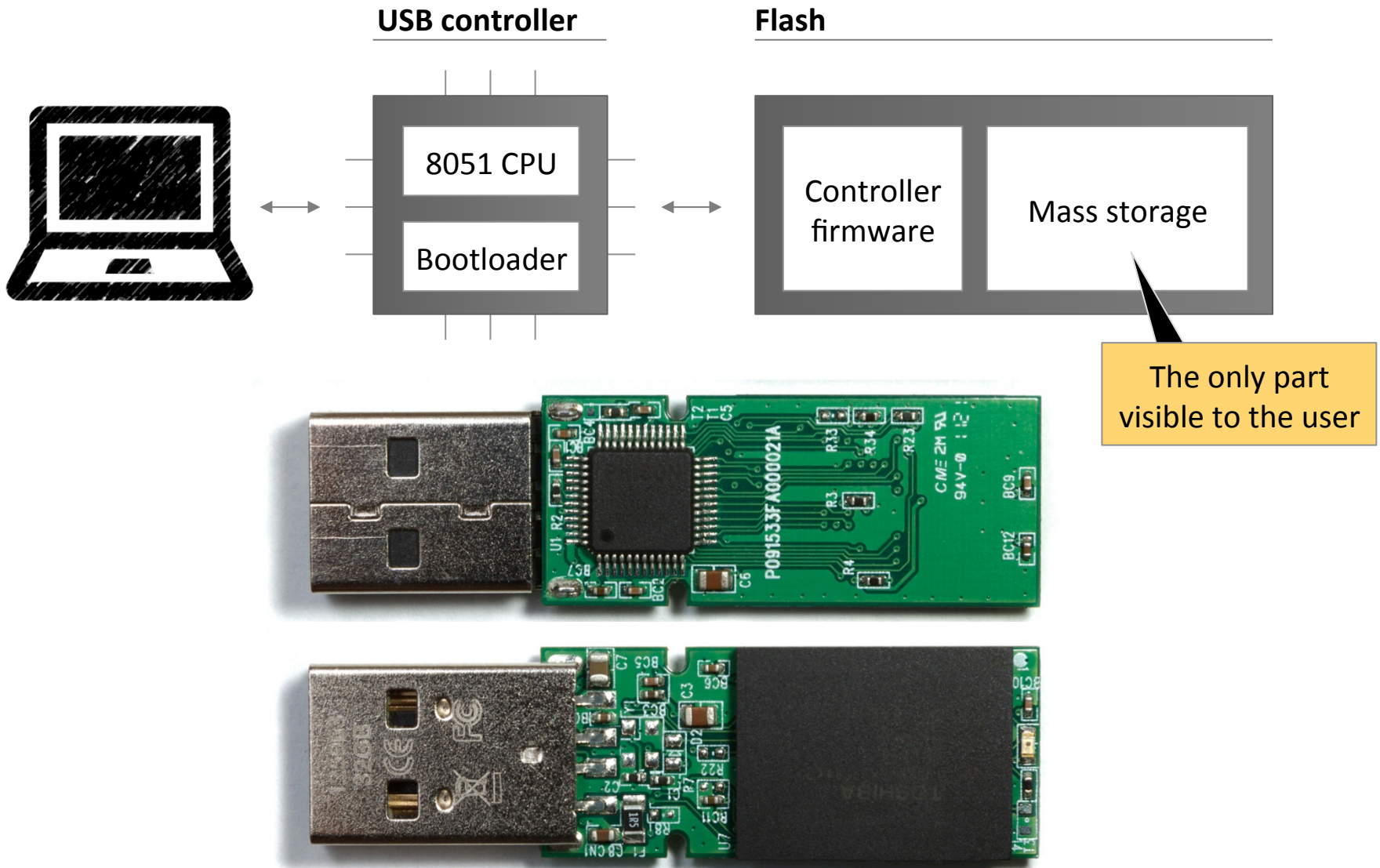
Sascha Krißler <sascha@srlabs.de>

Jakob Lell <jakob@srlabs.de>

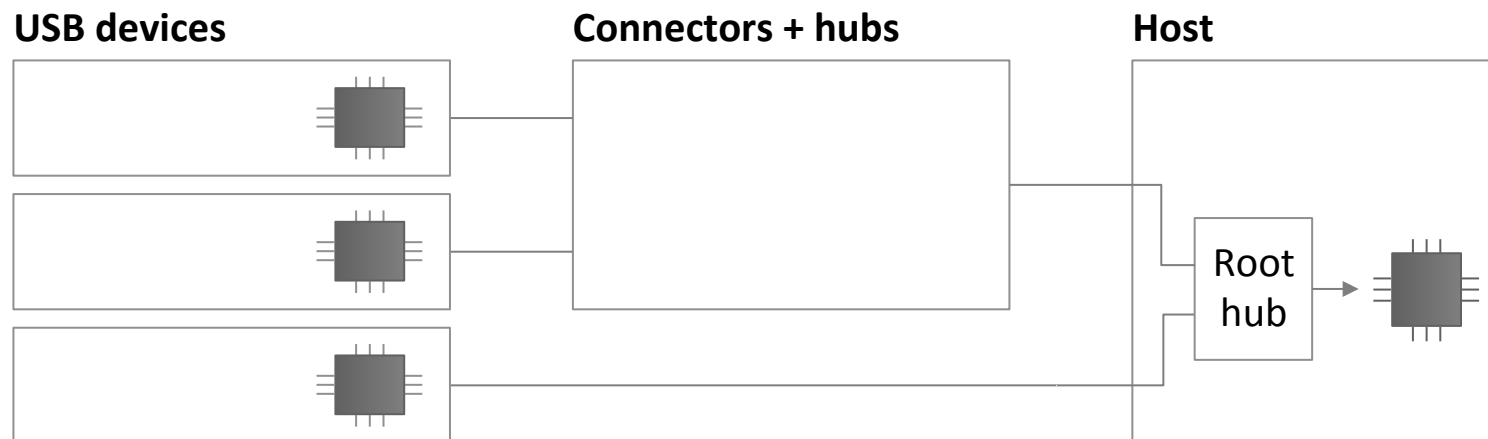


SECURITY
RESEARCH
LABS

USB devices include a micro-controller, hidden from the user



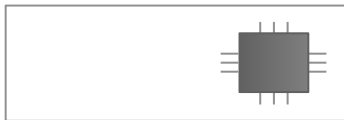
USB devices are identified



	Examples	
Identifier	USB thumb drive	Webcam
Interface class	8 – Mass Storage	a. 1 – Audio b. 14 – Video
End points	0 – Control 1 – Data transfers	0 – Control 1 – Video transfers 6 – Audio transfers 7 – Video interrupts
Serial number	AA627090820000000702	0258A350

USB devices are initialized in several steps

USB device



← USB plug-and-play →



Register →

← Set address

→ Send descriptor

← Set configuration

← Normal operation →

→ Optional: deregister

→ Register again ...

Load driver

Load another driver

Devices can have several identities

- A device indicates its capabilities through a descriptor
- A device can have several descriptors if it supports multiple device classes; like webcam + microphone
- Device can deregister and register again as a different device

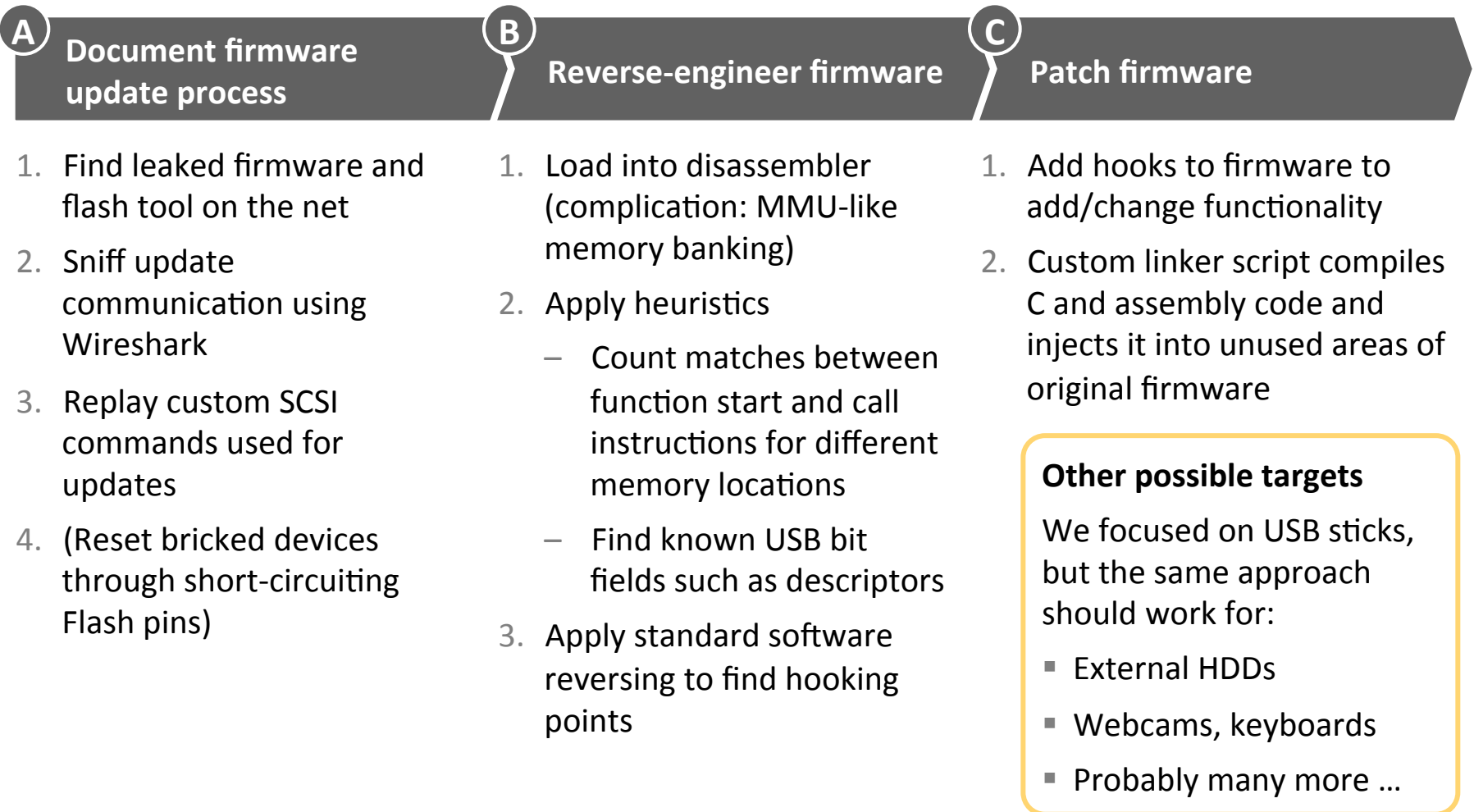
Agenda

-
- USB background

- ▶ **Reprogramming peripherals**

- USB attack scenarios
 - Defenses and next steps
-

Reversing and patching USB firmware took less than 2 months



Agenda

-
- USB background
 - Reprogramming peripherals

 **USB attack scenarios**

- Defenses and next steps
-

Keyboard emulation is enough for infection and privilege escalation (w/o need for software vulnerability)

Challenge – Linux malware runs with limited user privileges, but needs *root* privileges to infect further sticks

Approach – Steal *sudo* password in screensaver

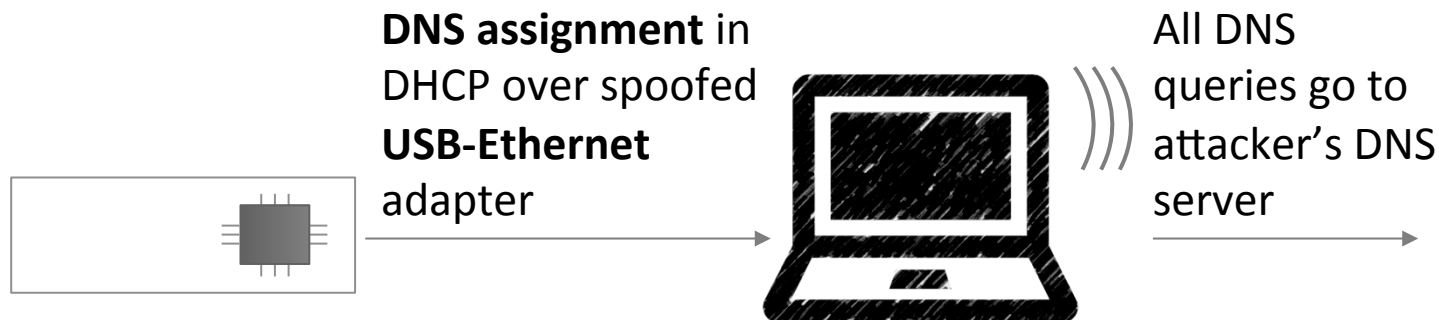
Restart screensaver (or *policykit*) with password stealer added via an LD_PRELOAD library



- User enters password to unlock screen
- Malware intercepts password and gains root privileges using *sudo*

Privilege escalation module will be submitted to Metasploit

Network traffic can be diverted by “DHCP on USB”



Attack steps

1. USB stick spoofs Ethernet adapter
2. Replies to DHCP query with DNS server on the Internet, but without default gateway



Result

3. Internet traffic is still routed through the normal Wi-Fi connection
4. However, DNS queries are sent to the USB-supplied server, enabling redirection attacks