

Binary Fun Week!

Boot loaders / bug hunting / attack surface auditing /
examples of vulnerabilities / and more

whoami

- Joseph Tartaro
 - 14 years at IOActive
 - Oracle



Agenda

- Security
 - Why?
- “Boot Loaders”
 - Why?
 - Common boot loaders
 - Supported features
- Attack Surface Analysis
 - Examples of bugs

Security

- Why?
 - Software is everywhere cars, hospitals, infrastructure...
- Security mindset helps developers
 - Avoid undefined behavior
 - Write safer interfaces and validation layers
 - Improve robustness
 - Crashes and logic flaws often share the same root cause

```
void copy_input(char *input) {  
    char buf[16];  
    strcpy(buf, input); // <-- no bounds checking  
    printf("Copied: %s\n", buf);  
}
```

```
void read_data(int len) {  
    char buf[64];  
    if (len < sizeof(buf)) {  
        fread(buf, len, 1, stdin);  
    }  
}
```

```
void allocate_and_copy(size_t count) {  
    size_t bytes = count * sizeof(int);  
    int *arr = malloc(bytes);  
    for (size_t i = 0; i < count; i++) arr[i] = i;  
}
```

Security Mindset

- Trust No One.
- ~~User~~ **Attacker** input.

Developer Mindset	Security Mindset
“How do I make this work?”	“How could this break?”
“User input will follow protocol”	“User input is adversarial.”
Focuses on expected paths	Focuses on edge cases
Concerned with it working	Concerned with it failing safely

“Boot Loaders”

- I'll be using a wide interpretation of “boot loader”... meaning “stuff in your boot chain”
- Code will call into BIOS/UEFI support routines
 - Similar to userland calling into kernel
 - SMM (System Management Mode) / TrustZone
- Similar to how an OS kernel should question everything and validate all attacker controlled data

Why?

- Boot loaders are a critical foundation of your devices security
 - They are a key component in the chain of trust
 - Everything that is loaded by them relies on the boot loaders integrity to uphold security guarantees
- Some secure device designers are poor at hardening and limited attack surface.
- Some designers underestimate Reverse Engineering (RE) and/or assume no bad actors in certain components.

Where?

- Everywhere!
 - Servers
 - Desktops / Workstations
 - Phones
 - Tablets
 - Entertainment Devices
 - Embedded Devices
 - ...
- Often some dependency on a secure boot loader

Standing on the shoulders of giants

Conversations, papers, ideas, books, presentations, code... with/from these people inspired and directory or indirectly contributed to the content here...

- Ilja van Sprundel
- Daniel Hodson
- Enrique Nissim
- Rodrigo Branco
- Vincent Zimmer
- Victor Tan
- Felix Domke
- Mathias Eissler
- Peter Baer
- Folks from legbacore
- Alex Matrosov
- ...

Common Boot Loaders

- These are publicly available open source and free software
- These are used in many real world scenarios
 - Often heavily customized though..
 - Hooks for secure boot
 - Drivers that aren't publicly available
 - HW specific changes
- What we'll cover to some extent:
 - Das U-Boot
 - Coreboot
 - Grub
 - Seabios
 - CFE
 - iPXE
 - TianoCore (UEFI)

Das U-Boot

- Boot loader that supports many boards and CPUs
 - Can configure your own board and/or CPU
- Offers a very customizable boot environment
 - Hundreds of defines you can set or not in your board config
 - Can set a bunch of default env vars you rely on
- Has a powerful shell
 - Many of the commands have dependencies on env vars
 - Offers API (run_command()) to run shell command, used by many configs
- Lots of drivers for various devices

Das U-Boot

- Feature Supports
 - Network (DHCP, ARP, DNS...)
 - File Systems (FAT, Reiser, JFFS2)
 - Loads various next stage images (Serial, Ethernet, Hard Drisk, CD-ROM, CompactFlash, USB, SCSI, NAND, Disk on Chip, PCI...)
- Used by *many* embedded devices

Coreboot

- Targeted at modern OS'
 - Does not support BIOS calls
 - Used to boot iPXE, gPXE, Etherboot, SeaBios... instead of implementing it's own features
- Used in chromebooks
- Interesting bits in Coreboot come from Google
- Contains System Management Mode (SMM)

Grub

- Common boot loader for linux
- Primary concern is Multiboot specification
- Filesystems
 - Amiga Fast File System (AFFS), AtheOS fs, BeFS, BtrFS, cpio, Linux ext2/ext3/ext4, DOS FAT12/FAT16/FAT32, exFAT, F2FS, HFS, HFS+, ISO9660, JFS, Minix fs, nilfs2, NTFS, ReiserFS, ROMFS, Amiga Smart FileSystem (SFS), Squash4, tar, UDF, BSD UFS/UFS2, XFS and ZFS
- There are UEFI signed version of Grub

Seabios

- Default BIOS for QEMU and KVM
- Supports BIOS calls, so it's commonly loaded by Coreboot or other boot loaders.
- Supports TPM
- Compatibility Support Module (CSM) for Unified Extensible Firmware Interface (UEFI) and Open Virtual Machine Firmware (OVMF)

CFE

- Broadcom boot loader
- Used in a bunch of wireless routers and home entertainment platforms
 - Appl Airport, Asus routers, Buffalo AirStation, Linksys WRT54G series, Netgear, LG, Samsung TVs, etc
- Network (DHCP, TFTP, DNS, ARP, ...)

iPXE

- Open source PXE boot loader
 - Emphasis on network options
 - Can retrieve data over many protocols
 - Supports 802.11, FCoE, AOE,...
- Has UEFI signed versions

TianoCore

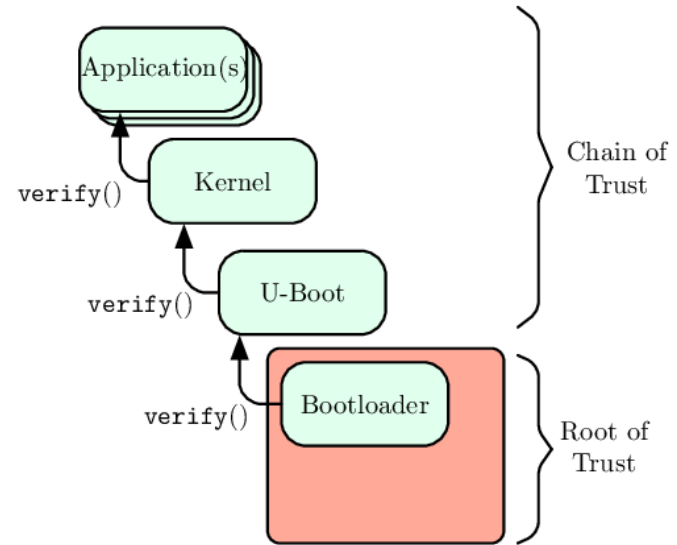
- Needs no introduction
 - Loads of documentation and specs
 - Dozens of security presentations regarding UEFI over the past 15 years or so...
- By far the most scrutinized and attacks
 - As a result, it's pretty mature compared to its alternatives
- Several implementations on top of it, such as Qualcomm's ABL, XBI...
- Platform specific stuff built on top

Related to boot loaders

- TrustZone OS
 - Optee_OS
 - NVIDIA trust little kernel
 - Trusty (Android)
 - Arm-trusted-firmware
- Reference implementation for secure world by ARM
 - Can be used by the Operating Systems mentioned above
- Host OS (e.g. Linux)
 - Could be anything. Often Linux and will focus on linux for this content

Secure Boot

- Root/Chain of Trust
 - Bootrom
 - 1st, 2nd and 3rd stage loader
 - Measured boot
 - Trusted boot
- TPM involvement
- TrustZone involvement
- Host OS booting and interaction



Root of Trust

- Start with bootrom (1st stage)
- Read only memory that can't be patched
 - Would require HW revision
- Memory size constrained
 - Does 'bare minimum'
 - Initializes some hardware
 - DRAM
 - IOMMU
 - ...
 - Might implement fastboot (e.g. has USB stack)
 - Hands of to the next stage
 - Read next stage from storage (e.g. flash, ssd, hdd)
 - Verify it's valid (signature checking) [trusted boot]
 - Might measure boot at various stages in cooperation with TPM [measured boot]
 - Run after verification succeeds

Chain of Trust

- Bootloader (2nd stage, optionally 3rd stage, ... Nth stage)
- Starts where bootrom left off
- Implement features needed by device:
 - Network / WiFi stack
 - SMM handlers
 - Environment variables
- Can hand off to next stage boot loaders
 - Will / should verify at each handoff stage before executing [trusted boot]
 - Might optionally measure at various points [measured boot]

Chain of Trust

- Some HW environments might offer more than 1 runtime environment
 - Secure World / Normal World
 - Arm: TrustZone
 - Windows: VTL0/VTL1, enforced by hypervisor
- Boot loader would hand off to TrustZone/HV boot code to set this up
 - Reads from storage
 - Verifies before running
 - Optionally measures
 - Hand off to next stage after verification

Chain of Trust

- Eventually loads OS
 - Read OS image/kernel from storage
 - Verify image/kernel before running
 - Optionally measure at various points
 - Hand off to OS Kernel
 - Might pass parameters along to OS Kernel
 - Some parameters might be influenced by NVRAM/ENV variables

Attack Surface

- NVRAM
- Files and File Systems
- Networking (TCP/IP, Bluetooth, 802.11, ...)
- Various Busses (USB, SPI, I2C, SDIO, ...)
- SMM
- Hardware

Attack Surface: NVRAM

- Environment Variables
- Can be modified from OS Runtime and effect configuration of the next boot sequence
- Standard parsing of user controlled data concerns

Attack Surface: NVRAM

Functions of interest when looking at attack surface:

- `env_get()` – U-boot
- `getenv()` – Coreboot
- `env_getenv()` – CFE
- `GetEnv()` – TianoCore

U-Boot

```
#define put_vci(e, str) \
do { \
    size_t vci_strlen = strlen(str); \
    *e++ = 60; /* Vendor Class Identifier */ \
    *e++ = vci_strlen; \
    memcpy(e, str, vci_strlen); \
    e += vci_strlen; \
} while (0)

static u8 *add_vci(u8 *e)
{
    char *vci = NULL;
    char *env_vci = env_get("bootp_vci");

    #if defined(CONFIG_SPL_BUILD) && defined(CONFIG_SPL_NET_VCI_STRING)
        vci = CONFIG_SPL_NET_VCI_STRING;
    #elif defined(CONFIG_BOOTP_VCI_STRING)
        vci = CONFIG_BOOTP_VCI_STRING;
    #endif

    if (env_vci)
        vci = env_vci;

    if (vci)
        put_vci(e, vci);

    return e;
}
```



```
Client hardware address padding: 00000000000000000000
```

```

▶ Option: (53) DHCP Message Type (Discover)
▶ Option: (57) Maximum DHCP Message Size
▶ Option: (235) Private

```

```

Option: (66) TFTP Server Name

```

[illegible]

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	0.0.0.0	255.255.255.255	BOOTP	53	[Malformed Packet]
2	0.000083	41:41:41:41:41:41	03:01:00:00:3c:00	0x4141	257	Ethernet II
3	0.751583	41:41:41:41:41:41	41:41:41:41:41:41	0x4141	3164	Ethernet II

Capture Length: 3164 bytes (25312 bits)

[Frame is marked: False]

[Frame is ignored: False]

[Protocols in frame: eth:ethertype:data]

[Coloring Rule Name: Broadcast]

[Coloring Rule String: eth[0] & 1]

▼ Ethernet II, Src: 41:41:41:41:41:41 (41:41:41:41:41:41), Dst: 41:41:41:41:41:41 (41:41:41:41:41:41)

▶ Destination: 41:41:41:41:41:41 (41:41:41:41:41:41)

▶ Source: 41:41:41:41:41:41 (41:41:41:41:41:41)

Type: Unknown (0x4141)

▼ Data (3150 bytes)

Data: 41...

[Length: 3150]

0000	41 41 41 41 41 41	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAA	AAAAAAAA
0010	41 41 41 41 41 41	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAA	AAAAAAAA
0020	41 41 41 41 41 41	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAA	AAAAAAAA
0030	41 41 41 41 41 41	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAA	AAAAAAAA
0040	41 41 41 41 41 41	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAA	AAAAAAAA
0050	41 41 41 41 41 41	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAA	AAAAAAAA
0060	41 41 41 41 41 41	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAA	AAAAAAAA
0070	41 41 41 41 41 41	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAA	AAAAAAAA
0080	41 41 41 41 41 41	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAA	AAAAAAAA
0090	41 41 41 41 41 41	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAA	AAAAAAAA
00a0	41 41 41 41 41 41	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAA	AAAAAAAA
00b0	41 41 41 41 41 41	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAA	AAAAAAAA
00c0	41 41 41 41 41 41	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAA	AAAAAAAA

```
#if defined(CONFIG_BOOTP_SEND_HOSTNAME)
    hostname = env_get("hostname");
    if (hostname) {
        int hostnamelen = strlen(hostname);

        *e++ = 12;          /* Hostname */
        *e++ = hostnamelen;
        memcpy(e, hostname, hostnamelen);
        e += hostnamelen;
    }
#endif
```

```
char build_buf[128]; /* Buffer for building the bootline */  
tmp = env_get("bootdev");  
if (tmp) {  
    strcpy(build_buf, tmp);  
    ptr = strlen(tmp);  
tmp = env_get("bootfile");  
if (tmp)  
    ptr += sprintf(build_buf + ptr, "host:%s ", tmp);  
tmp = env_get("ipaddr");  
if (tmp) {  
    ptr += sprintf(build_buf + ptr, "e=%s", tmp);  
}
```

```
tmp = env_get("serverip");  
if (tmp)  
    ptr += sprintf(build_buf + ptr, "h=%s ", tmp);
```

```
tmp = env_get("gatewayip");  
if (tmp)  
    ptr += sprintf(build_buf + ptr, "g=%s ", tmp);
```

```
tmp = env_get("hostname");  
if (tmp)  
    ptr += sprintf(build_buf + ptr, "tn=%s ", tmp);
```

```
tmp = env_get("othbootargs");  
if (tmp) {  
    strcpy(build_buf + ptr, tmp);
```

```

void set_env_gpios(unsigned char state)
{
    char *ptr_env;
    char str_tmp[5]; /* must contain "ledX"*/
    unsigned char i, idx, pos1, pos2, ccount;
    unsigned char gpio_n, gpio_s0, gpio_s1;

    for (i = 0; i < MAX_NR_LEDS; i++) {
        sprintf(str_tmp, "led%d", i);

        /* If env var is not found we stop */
        ptr_env = env_get(str_tmp);
        if (NULL == ptr_env)
            break;

        /* Find operators position */
        pos1 = 0;
        pos2 = 0;
        ccount = 0;
        for (idx = 0; ptr_env[idx] != '\0'; idx++) {
            if (ptr_env[idx] == ',') {
                if (ccount++ < 2)
                    pos1 = idx;
                else
                    pos2 = idx;
            }
        }
        /* Bad led description skip this definition */
        if (pos2 <= pos1 || ccount > 2)
            continue;

        /* Get pin number and request gpio */
        memset(str_tmp, 0, sizeof(str_tmp));
        strncpy(str_tmp, ptr_env, pos1*sizeof(char));
    }
}

```

```

int do_jffs2_fsload(cmd_tbl_t *cmdtp, int flag, int argc, char * const argv[])
{
    char *fsname;
    char *filename;
    ...
    /* pre-set Boot file name */
    filename = env_get("bootfile");
    ...

    if ((part = jffs2_part_info(current_mtd_dev, current_mtd_partnum))) {

        /* check partition type for cramfs */
        fsname = (cramfs_check(part) ? "CRAMFS" : "JFFS2");
        printf("### %s loading '%s' to 0x%lx\n", fsname, filename, offset);

        if (cramfs_check(part)) {
            size = cramfs_load ((char *) offset, part, filename);
        } else {
            /* if this is not cramfs assume jffs2 */
            size = jffs2_lpass_load((char *)offset, part, filename);
        }
    }
    ...
}

```

```
u32
jffs2_1pass_load(char *dest, struct part_info * part, const char *fname)
{
    ...
    if (! (inode = jffs2_1pass_search_inode(pl, fname, 1))) { ... }
    ...
}

static u32
jffs2_1pass_search_inode(struct b_lists * pL, const char *fname, u32 pino)
{
    int i;
    char tmp[256];
    char working_tmp[256];
    char *c;

    /* discard any leading slash */
    i = 0;
    while (fname[i] == '/')
        i++;
    strcpy(tmp, &fname[i]);
    ...
}
```

Attack Surface: Filesystems

- FS mounting
- Often FS itself is not signed / integrity protected
- E.g. FAT FS on a USB device
- Prime attack surface

Attack Surface: Files

- Files inside the file system
- Depending on software some/most/all are not integrity protected
- File parsers should be considered attack surface
 - E.g. use AFL to fuzz file parsers

Example: Filesystem/Files

- Ext2
 - Grub
 - Reading symlinks
- Drivers
- Binaries
- Partition Tables
- Malicious Capsule Updates
- BMP Splash Images

Grub

```
static char *
grub_ext2_read_symlink (grub_fshelp_node_t node)
{
  char *symlink;
  struct grub_fshelp_node *diro = node;

  if (! diro->inode_read)
    {
      grub_ext2_read_inode (diro->data, diro->ino, &diro->inode);
      if (grub_errno)
        return 0;

      if (diro->inode.flags & grub_cpu_to_le32_compile_time (EXT4_ENCRYPT_FLAG))
        {
          grub_error (GRUB_ERR_NOT_IMPLEMENTED_YET, "symlink is encrypted");
          return 0;
        }
    }

  symlink = grub_malloc (grub_le_to_cpu32 (diro->inode.size) + 1);
  if (! symlink)
    return 0;

  /* If the filesize of the symlink is bigger than
     60 the symlink is stored in a separate block,
     otherwise it is stored in the inode. */
  if (grub_le_to_cpu32 (diro->inode.size) <= sizeof (diro->inode.symlink))
    grub_memcpy (symlink,
                 diro->inode.symlink,
                 grub_le_to_cpu32 (diro->inode.size));
  else
    {
      grub_ext2_read_file (diro, 0, 0, 0,
                           grub_le_to_cpu32 (diro->inode.size),
                           symlink);
      if (grub_errno)
        {
          grub_free (symlink);
          return 0;
        }
    }

  symlink[grub_le_to_cpu32 (diro->inode.size)] = '\0';
  return symlink;
} « end grub_ext2_read_symlink »
```

TianoCore

MdeModulePkg/HiImage: Fix stack overflow when corrupted BMP is parse...

...d (CVE-2018-12181)

REF: https://bugzilla.tianocore.org/show_bug.cgi?id=1135

For 4bit BMP, there are only $2^4 = 16$ colors in the palette.
But when a corrupted BMP contains more than 16 colors in the palette,
today's implementation wrongly copies all colors to the local
PaletteValue[16] array which causes stack overflow.

The similar issue also exists in the logic to handle 8bit BMP.

The patch fixes the issue by only copies the first 16 or 256 colors
in the palette depending on the BMP type.

```
372 372 ZeroMem (PaletteValue, sizeof (PaletteValue));
373 - CopyRgbToGopPixel (PaletteValue, Palette->PaletteValue, PaletteNum);
373 + CopyRgbToGopPixel (PaletteValue, Palette->PaletteValue, MIN (PaletteNum, ARRAY_SIZE (PaletteValue)));
374 374 FreePool (Palette);
375 375
376 376 //
@@ -447,7 +447,7 @@ Output8bitPixel (
@@
447 447 CopyMem (Palette, PaletteInfo, PaletteSize);
448 448 PaletteNum = (UINT16)(Palette->PaletteSize / sizeof (EFI_HII_RGB_PIXEL));
449 449 ZeroMem (PaletteValue, sizeof (PaletteValue));
450 - CopyRgbToGopPixel (PaletteValue, Palette->PaletteValue, PaletteNum);
450 + CopyRgbToGopPixel (PaletteValue, Palette->PaletteValue, MIN (PaletteNum, ARRAY_SIZE (PaletteValue)));
451 451 FreePool (Palette);
452 452
```

Attack Surface: TCP/IP

- Whole TCP/IP network stack
- Talk to Services
 - BOOTP/DHCP
 - DNS
 - Network Filesystem (ISCSI, NFS)
 - IPSec
 - HTTP(S)
 - TFTP

Attack Surface: TCP/IP

- TCP/IP TLV Parsing
 - DoS' are plenty
- DNS & DHCP
 - Cache poisoning and lease stealing
 - Static ID, no ID validation, predictable ID
- Memory corruption bugs
- Information leaks
 - Leak uninitialized data over network... think heartbleed
- Use network stack fuzzers
 - E.g. isic

U-Boot

```
static void dns_send(void)
{
    struct header *header;
    int n, name_len;
    uchar *p, *pkt;
    const char *s;
    const char *name;
    enum dns_query_type qtype = DNS_A_RECORD;

    name = net_dns_resolve;
    pkt = (uchar *)(net_tx_packet + net_eth_hdr_size() + IP_UDP_HDR_SIZE);
    p = pkt;

    /* Prepare DNS packet header */
    header = (struct header *)pkt;
    header->tid = 1;
    header->flags = htons(0x100); /* standard query */
    header->nqueries = htons(1); /* Just one query */
    header->nanswers = 0;
    header->nauth = 0;
    header->nother = 0;
```

CFE

```
static int dhcp_wait_reply(int s,uint32_t id,dhcpreply_t *reply,int expfcode)
{
    uint32_t tmpd;
    uint8_t tmpb;
    int64_t timer;
    int nres = 0;
    uint8_t ciaddr[IP_ADDR_LEN];
    uint8_t yiaddr[IP_ADDR_LEN];
    uint8_t siaddr[IP_ADDR_LEN];
    uint8_t giaddr[IP_ADDR_LEN];
    uint8_t junk[128];
    ...
    if (tmpd == DHCP_MAGIC_NUMBER) {
        uint8_t tag;
        uint8_t len;

        while (buf->eb_length > 0) {
            ebuf_get_u8(buf,tag);
            if (tag == DHCP_TAG_END) break;
            ebuf_get_u8(buf,len);
            ebuf_get_bytes(buf,junk,len);
        }
    }
    ...
}
```


CFE

```
#define TFTP_BLOCKSIZE      512

typedef struct tftp_info_s {
    int tftp_socket;
    uint8_t tftp_data[TFTP_BLOCKSIZE];
    ...
    char *tftp_filename;
} tftp_info_t;

static int _tftp_readmore(tftp_info_t *info)
{
    ebuf_t *buf;
    ...
    buf = udp_alloc();
    ...
    buf = udp_recv_with_timeout(info->tftp_socket, tftp_rcv_timeout);
    ...
    ebuf_get_bytes(buf, info->tftp_data, ebuf_length(buf));
    ...
}
```

CFE

```
int _icmp_ping(icmp_info_t *icmp,uint8_t *dest,int seq,int len)
{
    ebuf_t *buf;
    int64_t timer;
    ...
    while (!TIMER_EXPIRED(timer)) {

        POLL();
        buf = (ebuf_t *) q_deqnext(&(icmp->icmp_echoresponses));

        /* If we get a packet, make sure it matches. */

        if (buf) {
            uint16_t rxid,rxseq;

            cksum = ip_chksum(0,buf->eb_ptr,ebuf_length(buf));
            if (cksum == 0xFFFF) {
                ebuf_skip(buf,2);
                ebuf_skip(buf,2);          /* skip checksum */
                ebuf_get_u16_be(buf,rxid);
                ebuf_get_u16_be(buf,rxseq);

                if ((id == rxid) && ((uint16_t) seq == rxseq)) {
                    result = 1;
                    break;
                }
            }
            _ip_free(icmp->icmp_ipinfo,buf);
        }

        /*
         * Don't accept any more replies.
         */

        icmp->icmp_maxreplies = 0;          /* allow ICMP replies */

        if (buf) _ip_free(icmp->icmp_ipinfo,buf);

        return result;
    }
}
```

CFE

```
static int ip_rx_callback(ebuf_t *buf, void *ref)
{
...
    uint16_t length;
...
    ebuf_get_u16_be(buf, length);
...
    ebuf_setlength(buf, length-hdrlen); /* set length to just data portion */
...
...
    /*
     * Find matching protocol dispatch
     */

    pdisp = ipi->ip_protocols;
    res = ETH_DROP;
    for (idx = 0; idx < IP_MAX_PROTOCOLS; idx++) {
        if (pdisp->cb && (pdisp->protocol == proto)) {
            res = (*(pdisp->cb))(pdisp->ref, buf, dstip, srcip);
            break;
        }
        pdisp++;
    }

    return res;

drop:
    return ETH_DROP;
}
```

Attack Surface: 802.11

- Some parse mostly on radio and then tell host
- Some are mostly pass-through and send directly to host
- Only a handful of bootloaders support 802.11
 - At the present time?
- Accidental vs. Purposeful attack surface reduction

iPXE

```
int net80211_probe_step ( struct net80211_probe_ctx *ctx )
{
...
    struct io_buffer *iob;
...
    char ssid[IEEE80211_MAX_SSID_LEN + 1];
...
    while ( ( iob = net80211_mgmt_dequeue ( dev, &signal ) ) != NULL ) {
        struct ieee80211_frame *hdr;
        struct ieee80211_beacon *beacon;
        union ieee80211_ie *ie;
...
        hdr = iob->data;
        type = hdr->fc & IEEE80211_FC_SUBTYPE;
        beacon = ( struct ieee80211_beacon * ) hdr->data;

        if ( type != IEEE80211_STYPE_BEACON &&
            type != IEEE80211_STYPE_PROBE_RESP ) {
            DBG2 ( dev, "802.11 %p probe: non-beacon\n", dev );
            goto drop;
        }
...
        ie = beacon->info_element;
...
        memcpy ( ssid, ie->ssid, ie->len );
        ssid[ie->len] = 0;
...
    }

    return 0;
}
```

Attack Surface: Bluetooth

- Often the same radio as 802.11, from the same manufacturer
- Surprisingly few boot environments seem to support Bluetooth
 - HID: keyboard and mouse
- Network protocol parsing
 - Large frames (~65k) might cause issues
 - Very short frames might cause issues
 - Fragmentation games

Attack Surface: USB

- USB devices
 - Storage (e.g. filesystems)
 - Ethernet dongles
- Descriptor parsing is often wrong
 - Straight up overflows
 - Descriptor double fetches
- Protocols like Fastboot, etc.

Grub

```
grub_usb_err_t
grub_usb_device_initialize (grub_usb_device_t dev)
{
    struct grub_usb_desc_device *descdev;
    struct grub_usb_desc_config config;
    ...
    err = grub_usb_get_descriptor (dev, GRUB_USB_DESCRIPTOR_DEVICE,
                                   0, 8, (char *) &dev->descdev);
    ...
    err = grub_usb_get_descriptor (dev, GRUB_USB_DESCRIPTOR_DEVICE,
                                   0, sizeof (struct grub_usb_desc_device),
                                   (char *) &dev->descdev);
    ...
    descdev = &dev->descdev;
    ...
    for (i = 0; i < descdev->configcnt; i++)
    {
        ...
        err = grub_usb_get_descriptor (dev, GRUB_USB_DESCRIPTOR_CONFIG, i, 4,
                                       (char *) &config);

        data = grub_malloc (config.totallen);
        ...
        dev->config[i].descconf = (struct grub_usb_desc_config *) data;
        err = grub_usb_get_descriptor (dev, GRUB_USB_DESCRIPTOR_CONFIG, i,
                                       config.totallen, data);
        ...
        for (currif = 0; currif < dev->config[i].descconf->numif; currif++)
        {
            ...
            dev->config[i].interf[currif].descif
                = (struct grub_usb_desc_if *) &data[pos];
            ...
        }
    }
    ...
}
```


TianoCore

```
EFI_STATUS
UsbHubReadDesc (
    IN  USB_DEVICE      *HubDev,
    OUT EFI_USB_HUB_DESCRIPTOR *HubDesc
)
{
    EFI_STATUS      Status;

    //
    // First get the hub descriptor length
    //
    Status = UsbHubCtrlGetHubDesc (HubDev, HubDesc, 2);

    if (EFI_ERROR (Status)) {
        return Status;
    }

    //
    // Get the whole hub descriptor
    //
    return UsbHubCtrlGetHubDesc (HubDev, HubDesc, HubDesc->Length);
} « end UsbHubReadDesc »
```

TianoCore

693	-	EFI_USB_HUB_DESCRIPTOR	HubDesc;
653	+	UINT8	HubDescBuffer[256];
654	+	EFI_USB_HUB_DESCRIPTOR	*HubDesc;
694	655	USB_ENDPOINT_DESC	*EpDesc;
695	656	USB_INTERFACE_SETTING	*Setting;
696	657	EFI_USB_IO_PROTOCOL	*UsbIo;
@@	@@	-725,14 +686,19 @@	UsbHubInit (
725	686	return	EFI_DEVICE_ERROR;
726	687	}	
727	688		
728	-	Status = UsbHubReadDesc	(HubDev, &HubDesc);
689	+	//	
690	+	// The length field of descriptor is UINT8 type, so the buffer	
691	+	// with 256 bytes is enough to hold the descriptor data.	
692	+	//	
693	+	HubDesc = (EFI_USB_HUB_DESCRIPTOR *)	HubDescBuffer;
694	+	Status = UsbHubReadDesc	(HubDev, HubDesc);
729	695		

Seabios

```
static struct usb_config_descriptor *  
get_device_config(struct usb_pipe *pipe)  
{  
    struct usb_config_descriptor cfg;  
  
    struct usb_ctrlrequest req;  
    req.bRequestType = USB_DIR_IN | USB_TYPE_STANDARD | USB_RECIP_DEVICE;  
    req.bRequest = USB_REQ_GET_DESCRIPTOR;  
    req.wValue = USB_DT_CONFIG<<8;  
    req.wIndex = 0;  
    req.wLength = sizeof(cfg);  
    int ret = usb_send_default_control(pipe, &req, &cfg);  
    if (ret)  
        return NULL;  
  
    void *config = malloc_tmphigh(cfg.wTotalLength);  
    if (!config) {  
        warn_noalloc();  
        return NULL;  
    }  
    req.wLength = cfg.wTotalLength;  
    ret = usb_send_default_control(pipe, &req, config);  
    if (ret) {  
        free(config);  
        return NULL;  
    }  
    //hexdump(config, cfg.wTotalLength);  
    return config;  
} « end get_device_config »
```

Nvidia Tegra Vulnerability (Nintendo Switch)

<https://fail0verflow.com/blog/2018/shofel2/>

https://github.com/Qyriad/fusee-launcher/blob/master/report/fusee_gelee.md

```
// Handle GET_STATUS requests.
if (setup_packet.request == REQUEST_GET_STATUS)
{
    // If this is asking for the DEVICE's status, respond accordingly.
    if(setup_packet.recipient == RECIPIENT_DEVICE) {
        status      = get_usb_device_status();
        size_to_tx = sizeof(status);
    }
    // Otherwise, respond with the ENDPOINT status.
    else if (setup_packet.recipient == RECIPIENT_ENDPOINT){
        status      = get_usb_endpoint_status(setup_packet.index);
        size_to_tx = length_read; // <-- This is a critical error!
    }
    else {
        /* ... */
    }

    // Send the status value, which we'll copy from the stack variable 'status'.
    data_to_tx = &status;
}

// Copy the data we have into our DMA buffer for transmission.
// For a GET_STATUS request, this copies data from the stack into our DMA buffer.
memcpy(dma_buffer, data_to_tx, size_to_tx);
```

iPhone Checkm8 vulnerability

<https://habr.com/en/company/dsec/blog/472762/>

When the device exits the DFU mode, the previously allocated IO buffer is freed. If the image was successfully acquired in the DFU mode, it will be verified and booted. If there was any error or it was impossible to boot the image, the DFU will be initialized again, and the whole process will repeat from the beginning.

The described algorithm has a use-after-free vulnerability. If we send a SETUP packet at the time of image uploading and complete the transaction skipping Data Stage, the global state will remain initialized during the next DFU cycle, and we will be able to write to the address of the IO buffer allocated during the previous iteration of DFU.

Attack Surface: SPI/SDIO/I2C

- Flash is often connected over SPI
- Not intended to be used by end user, so developers mistake it as trusted. IT IS NOT.
- This includes the TPM!

Seabios

```
static int
tpm20_getcapability(u32 capability, u32 property, u32 count,
                   struct tpm_rsp_header *rsp, u32 rsize)
{
    struct tpm2_req_getcapability trg = {
        .hdr.tag = cpu_to_be16(TPM2_ST_NO_SESSIONS),
        .hdr.totlen = cpu_to_be32(sizeof(trg)),
        .hdr.ordinal = cpu_to_be32(TPM2_CC_GetCapability),
        .capability = cpu_to_be32(capability),
        .property = cpu_to_be32(property),
        .propertycount = cpu_to_be32(count),
    };

    u32 resp_size = rsize;
    int ret = tpmhw_transmit(0, &trg.hdr, rsp, &resp_size,
                           TPM_DURATION_TYPE_SHORT);

    ret = (ret ||
          rsize < be32_to_cpu(rsp->totlen)) ? -1 : be32_to_cpu(rsp->errcode);

    dprintf(DEBUG_tcg, "TCGBIOS: Return value from sending TPM2_CC_GetCapability = 0x%08x\n",
            ret);

    return ret;
} « end tpm20_getcapability »

static int
tpm20_get_pcrbanks(void)
{
    u8 buffer[128];
    struct tpm2_res_getcapability *trg =
        (struct tpm2_res_getcapability *) &buffer;

    int ret = tpm20_getcapability(TPM2_CAP_PCERS, 0, 8, &trg->hdr,
                                sizeof(buffer));

    if (ret)
        return ret;

    u32 size = be32_to_cpu(trg->hdr.totlen) -
               offsetof(struct tpm2_res_getcapability, data);
    tpm20_pcr_selection = malloc_high(size);
    if (tpm20_pcr_selection) {
        memcpy(tpm20_pcr_selection, &trg->data, size);
        tpm20_pcr_selection_size = size;
    } else {
        warn_noalloc();
        ret = -1;
    }

    return ret;
} « end tpm20_get_pcrbanks »
```

Attack Surface: SMM

- System Management Mode (ring - 2)
 - Entire presentations on SMM and security
 - Doing SMM right is HARD.
- UEFI does pretty well these days
 - edkll
 - 3rd party stuff has occasional issues
- Re-implementing from scratch is HARD
 - You'll probably get it wrong the first couple of tries

Coreboot

```
/*
 * Check that the given range is legal.
 *
 * Legal means:
 * - not pointing into SMRAM
 * - ...?
 *
 * returns 0 on success, -1 on failure
 */
static int range_check(void *start, size_t size)
{
    TODO: fill in
    return 0;
}

/* Param is usually EBX, ret in EAX */
uint32_t smmstore_exec(uint8_t command, void *param)
{
    uint32_t ret = SMMSTORE_RET_FAILURE;

    switch (command) {
    case SMMSTORE_CMD_READ: {
        printk(BIOS_DEBUG, "Reading from SMM store\n");
        struct smmstore_params_read *params = param;

        if (range_check(params->buf, params->bufsize) != 0)
            break;

        if (smmstore_read_region(params->buf, &params->bufsize) == 0)
            ret = SMMSTORE_RET_SUCCESS;
        break;
    }

    case SMMSTORE_CMD_APPEND: {
        printk(BIOS_DEBUG, "Appending into SMM store\n");
        struct smmstore_params_append *params = param;

        if (range_check(params->key, params->keysize) != 0)
            break;
        if (range_check(params->val, params->valsize) != 0)
            break;

        if (smmstore_append_data(params->key, params->keysize,
                                params->val, params->valsize) == 0)
            ret = SMMSTORE_RET_SUCCESS;
        break;
    }
}
```

Hardware

- This can mean a lot of things
 - Glitching attacks
 - Side Channel
 - Chipsec (silicon)

Hardware: Glitching

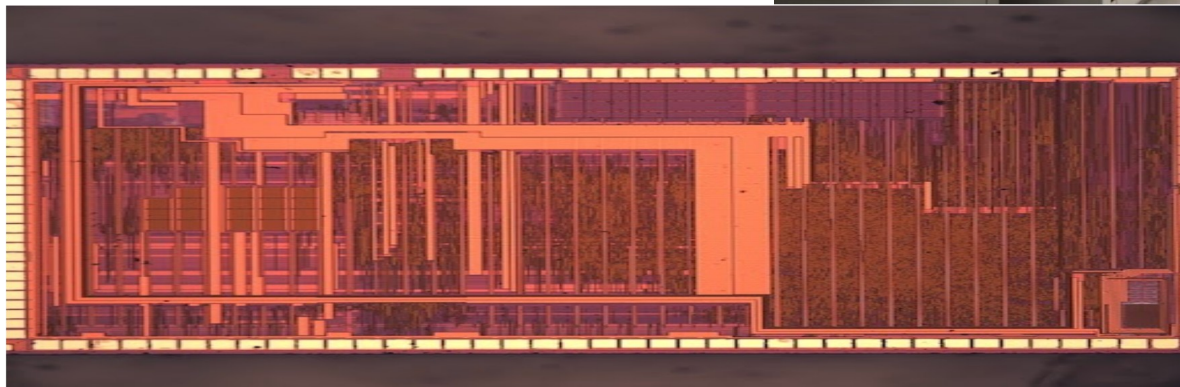
- Fault Injection
 - Clock
 - Voltage
 - Laser
 - Coil
- Common scenario: glitching code integrity checking
- Modern examples: breaking nRF52

Hardware: Side Channel

- Timing discrepancies
- Power consumption discrepancies
- CPU bugs
 - Spectre & meltdown
- Leak secrets

Hardware: Chipsec

- Decapping, FIB, SEM, ...
- Very sophisticated attacker
- Optical ROM Extraction
 - Beginning of the Chain of Trust!



A note on code integrity

- Doing code integrity right is hard
- Some things we've seen go wrong
 - Weak crypto/algo
 - Blacklist problems
 - List exhausted
 - Fail open/closed in case of failure (e.g. OOM)
 - Known bad but not in blacklist (blacklist update fail)
 - E.g. signed grub with known bug for UEFI (Kaspersky)
- Some sections in exec file not signed
- Checking if signatures exist but not verifying them

Conclusions

- We've just covered the tip of the iceberg
- Surprising amount of low quality code
 - Strcpy / sprintf type of bugs
- You get to NDA hell pretty quickly once you start digging around
 - All of the proprietary stuff...

Advice / Mitigations

- Minimize image, both boot environment and host OS.
- Turn off features you don't need
 - Net
 - USB
 - File Systems
 - ENV
 - Anything you don't explicitly need. It could have attack surface.
- Need more people reviewing bootloaders
 - Fuzz interfaces! (net, bus, file, fs, ...)
 - Static analysis
 - Periodic reviews

Contact Info

joseph.tartaro@gmail.com

@droogie1xp