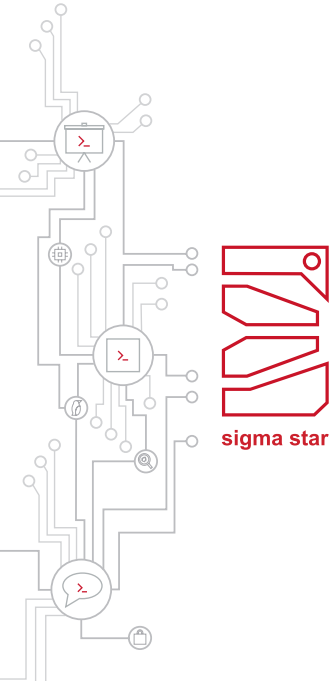
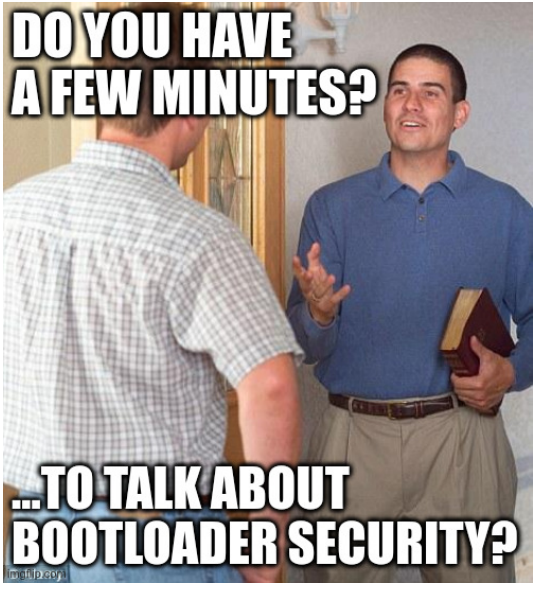


Why Bootloader Security Matters

Richard Weinberger - sigma star gmbh

2024-07-25



Richard Weinberger

- Co-founder of sigma star gmbh
- Linux kernel developer and maintainer
- Strong focus on Linux kernel, lowlevel components, virtualization, security, code audits

sigma star gmbh

- Software Development & Security Consulting
- Main areas: Embedded Systems, Linux Kernel & Security
- Contributions to Linux Kernel and other OSS projects

Bootloader Security and Verified/Secure Boot

- Started by boot ROM, starts OS
- Offline modifications of software are bad
- e.g. evil maid attack
- Run only authenticated software (IoT, SmartPhone)
- Bootloader is the first step of chain of trust
- The OS (usually!) protects you
- Hello CRA (Cyber Resilience Act), hello NIS-2 (Network and Information Security)

The Weakest Link: The Bootloader

- Break the bootloader and control the rest of the system:
  - Start our own code
  - Extract secrets (key material, IP)
  - Impersonate the device

U-Boot and Barebox

- Extremely common bootloaders for embedded Linux
- Load and authenticate files from a filesystem
- Started auditing their critical code paths

Vulnerability #1

- Integer overflow in ext4 symlink code
- Results in attacker driven out of bounds write
- Unauthenticated attacker can trigger it
- Both U-Boot and Barebox affected

```
static char *ext4fs_read_symlink(struct ext2fs_node *node)
{
...
   symlink = zalloc(le32_to_cpu(diro->inode.size) + 1);
   if (!symlink)
      return NULL;
...
}
```

Vulnerability #2

- Integer overflow in squashfs symlink code, like vulnerability #1.
- Results in attacker driven out of bounds write
- Unauthenticated attacker can trigger it
- Both U-Boot and Barebox affected
- Although they have different squashfs implementations

Vulnerability #3

- Stack overflow in squashfs symlink code
- Code follows symlinks recursively
- Results in attacker driven stack smashing
- Unauthenticated attacker can trigger it
- Only U-Boot affected

```
int sqfs_size(const char *filename, loff_t *size)
{
...
   switch (get_unaligned_le16(&base->inode_type)) {
...
   case SQFS_LSYMLINK_TYPE:
   symlink = (struct squashfs_symlink_inode *)ipos;
   resolved = sqfs_resolve_symlink(symlink, filename);
   ret = sqfs_size(resolved, size);
   free(resolved);

   break;
...
}
```

Vulnerability #4

- Multiple integer overflows in memory allocator
- You ask for `N` bytes but get much less
- Can get triggered by most filesystem drivers
- Unauthenticated attacker can trigger it
- Both U-Boot and Barebox affected
- They use Doug Lea's Malloc, but broke it 25 years ago
- Bonus: Another integer overflow in their `sbrk()`
- Bonus #2: `ptrdiff_t` too small on x86_64, more overflows

```
/* pad request bytes into a usable size */

#define request2size(req) \
 (((long)((req) + (SIZE_SZ + MALLOC_ALIGN_MASK)) < \
  (long)(MINSIZE + MALLOC_ALIGN_MASK)) ? MINSIZE : \
   (((req) + (SIZE_SZ + MALLOC_ALIGN_MASK)) & ~(MALLOC_ALIGN_MASK)) ↲
         ))

Void_t* mALLOc_impl(size_t bytes)
{
...
  if ((long)bytes < 0) return NULL;

  nb = request2size(bytes);  /* padded request size; */
...
}
```

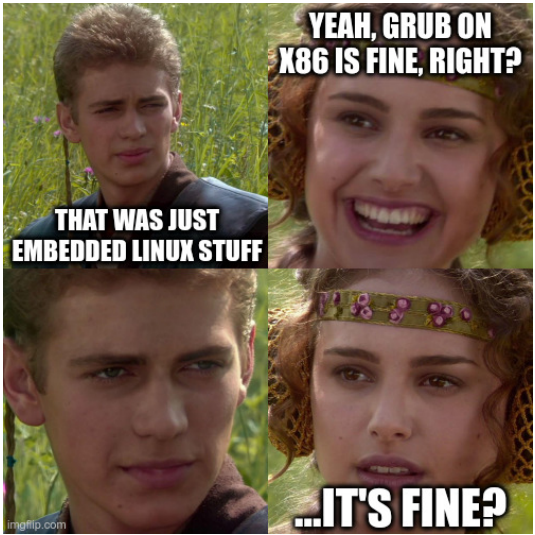
Outcome

- Found 10 vulnerabilities
- At least four beefy vulnerabilities that allow full compromise
- Sent bug reports and patches for all vulnerabilities
- Improved U-Boot's ASAN integration
- Barebox revived thier fuzzing project

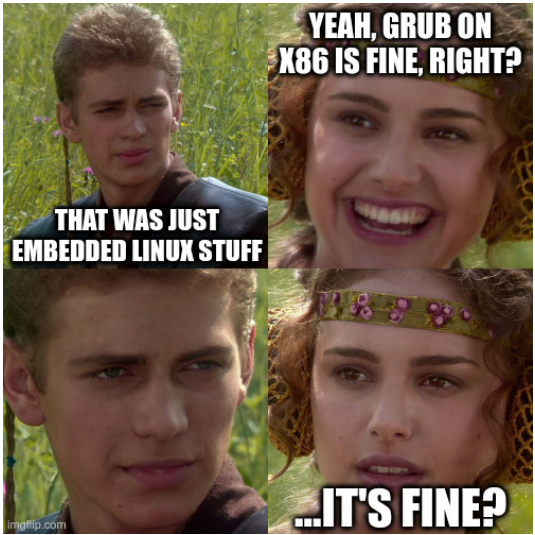
Just Update the Damn Bootloader?!

- Think of downgrade attacks
- Attacker can always install the old vulnerable bootloader
- Mitigations:
  - Have a revoke list (hard!)
  - Have an authenticated version counter in hardware

What About Non-Embedded?



What About Non-Embedded?



- CVE-2024-2312
- CVE-2024-1048
- CVE-2023-4693
- CVE-2023-4692
- CVE-2023-4001
- CVE-2022-28736
- CVE-2022-28735
- CVE-2022-28734
- CVE-2022-28733
- CVE-2022-3775
- CVE-2022-2601
- CVE-2021-46705
- CVE-2021-20233
- CVE-2021-20225
- CVE-2021-3981
- ...

FIN

Thank you!

Questions, Comments?

Richard Weinberger
richard@sigma-star.at