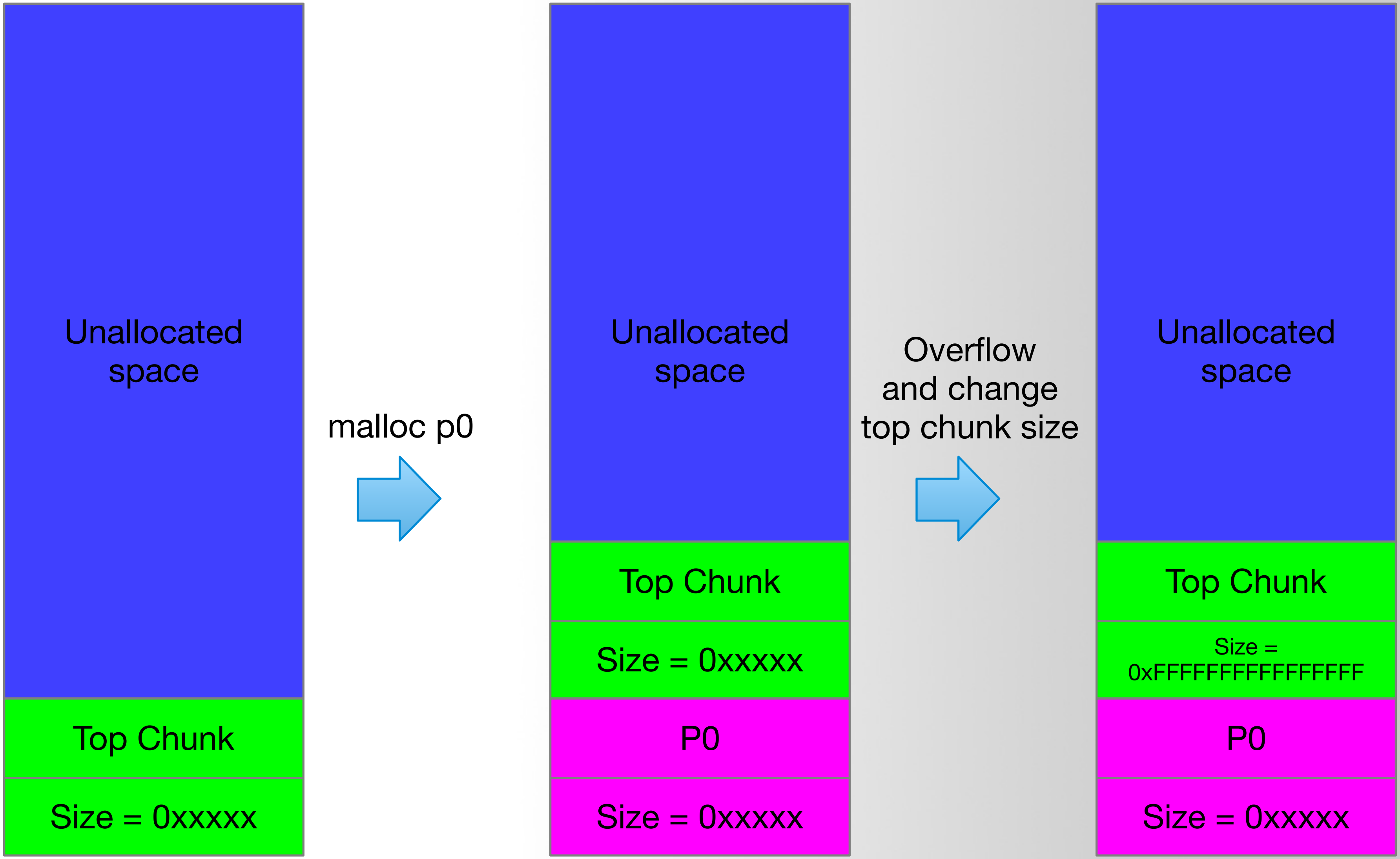
Class 23

CSC 472 Software Security
Use After Free (UAF), Double Free, Hacking Gaming Consoles

Dr. Si Chen (schen@wcupa.edu)

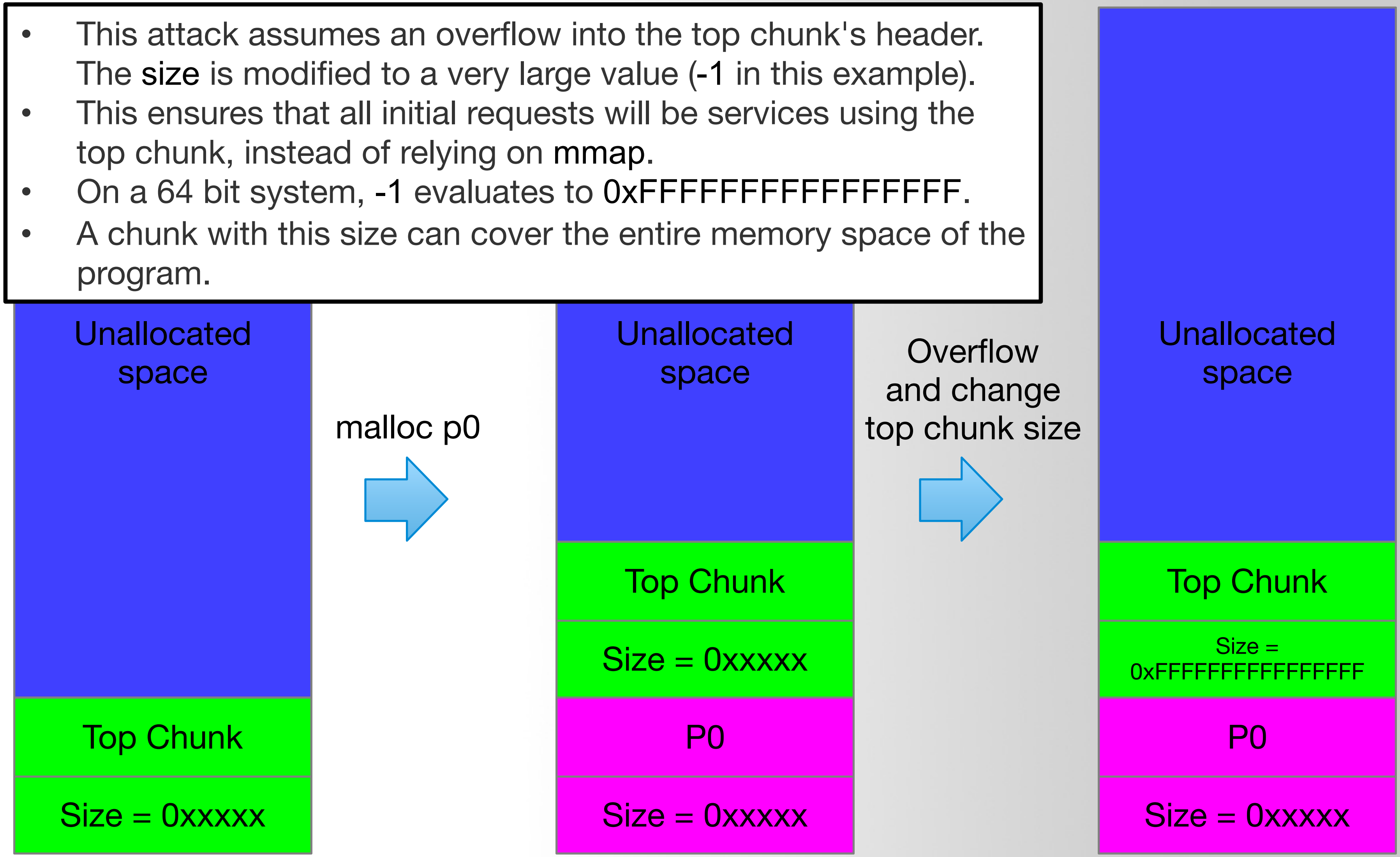
Review

House of Force



House of Force

- This attack assumes an overflow into the top chunk's header. The size is modified to a very large value (-1 in this example).
- This ensures that all initial requests will be services using the top chunk, instead of relying on mmap.
- On a 64 bit system, -1 evaluates to 0xFFFFFFFFFFFFFFFF.
- A chunk with this size can cover the entire memory space of the program.



House of Force

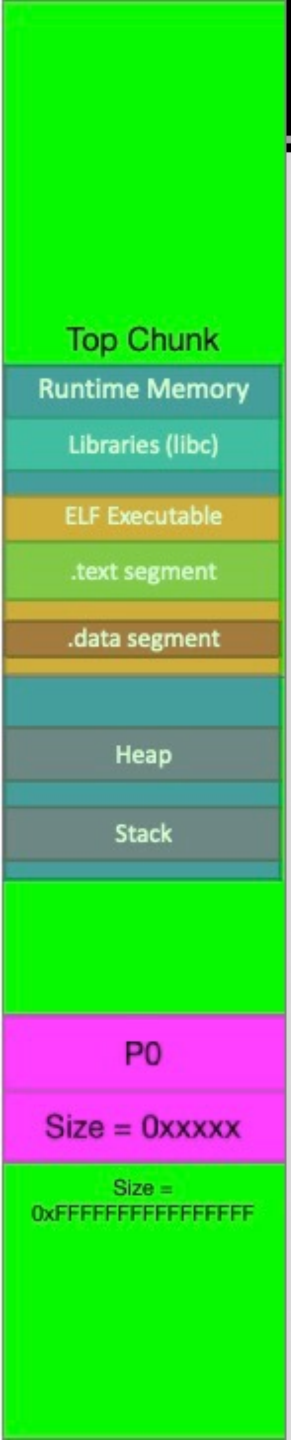
E.g. top_chunk=0x601200

malloc(0xffe00030)

0xffe00030 < top_chunk_size

0xffe00030+0x601200=0x100401230

top_chunk=0x401230

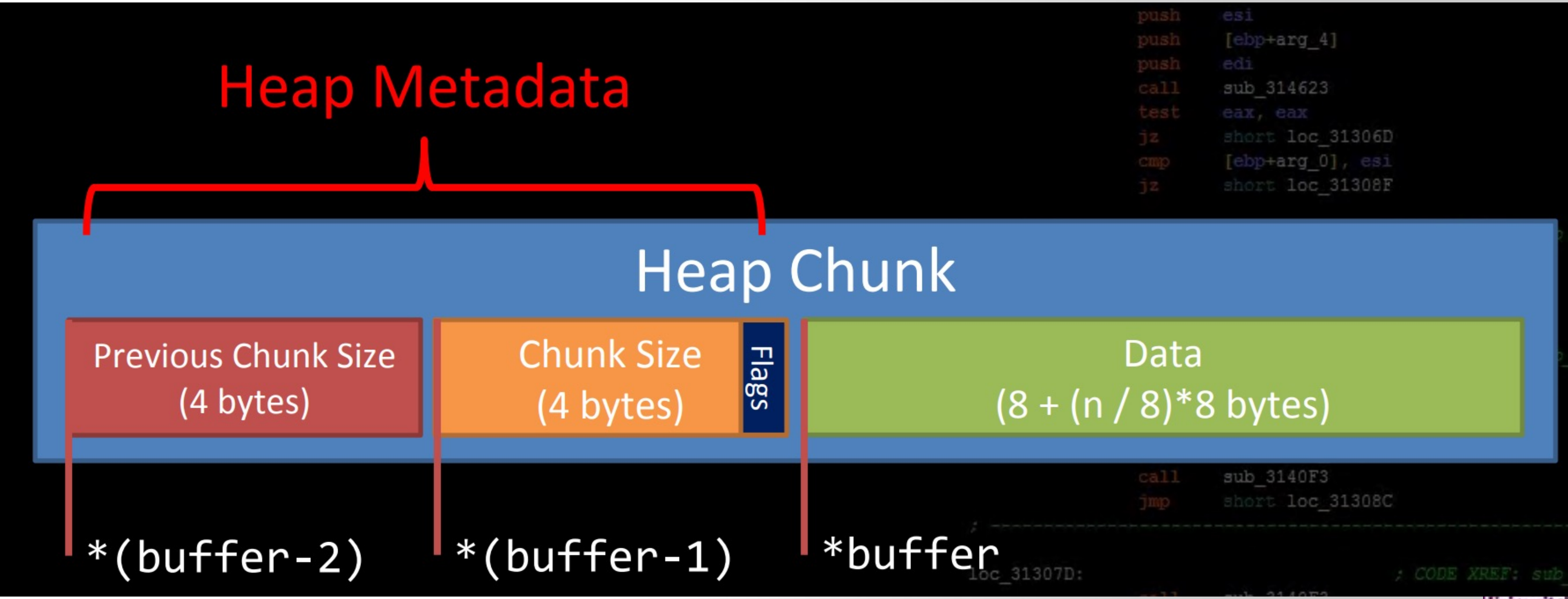


House of Force

- *Prerequisites*: Three malloc calls are required to successfully apply house of force as listed below:
  - Malloc 1: Attacker should be able to control the size of top chunk. Hence heap overflow should be possible on this allocated chunk which is physically located previous to top chunk.
  - Malloc 2: Attacker should be able to control the size of this malloc request.
  - Malloc 3: User input should be copied to this allocated chunk.

Metadata Corruption -- Unlink, House of Force

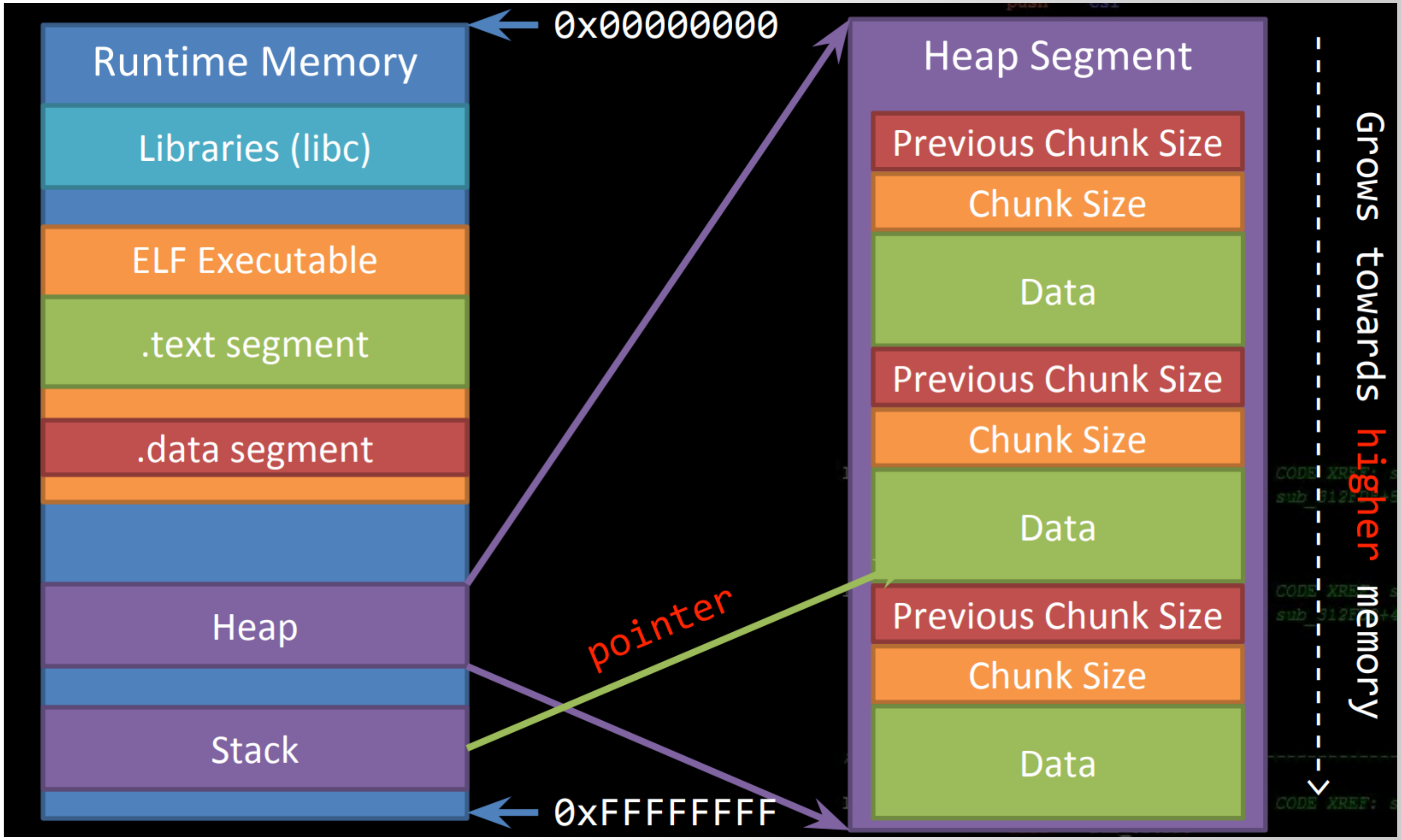
- Metadata corruption based exploits involve corrupting heap metadata in such a way that you can use the allocator’s internal functions to cause a controlled write of some sort
- Generally involves faking chunks, and abusing its different coalescing or unlinking processes



Use After Free

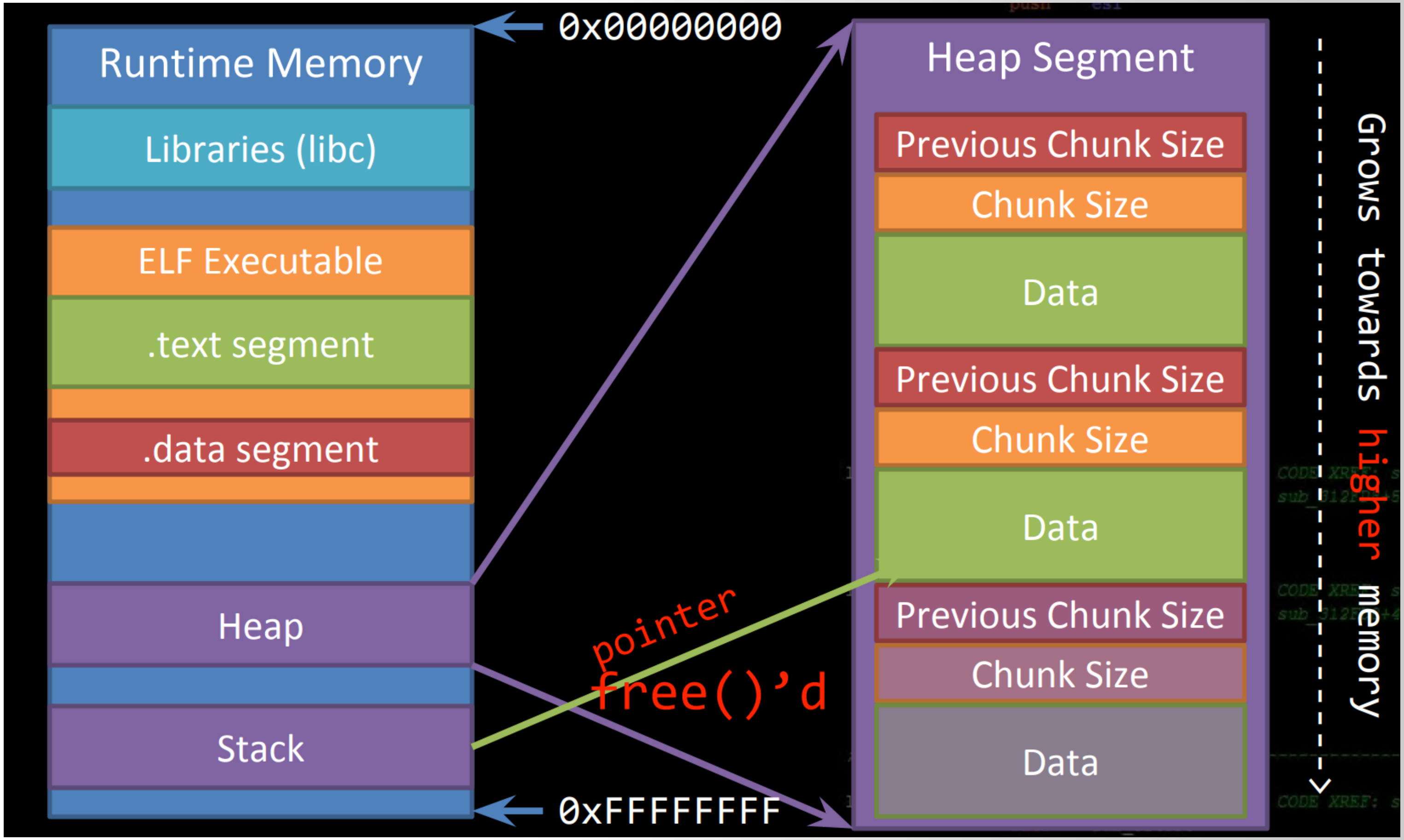
- **Use After Free**
  - A class of vulnerability where data on the heap is freed, but a leftover reference or ‘**dangling pointer**’ is used by the code as if the data were still valid
  - Most popular in Web Browsers, complex programs
  - Also known as UAF

Use After Free



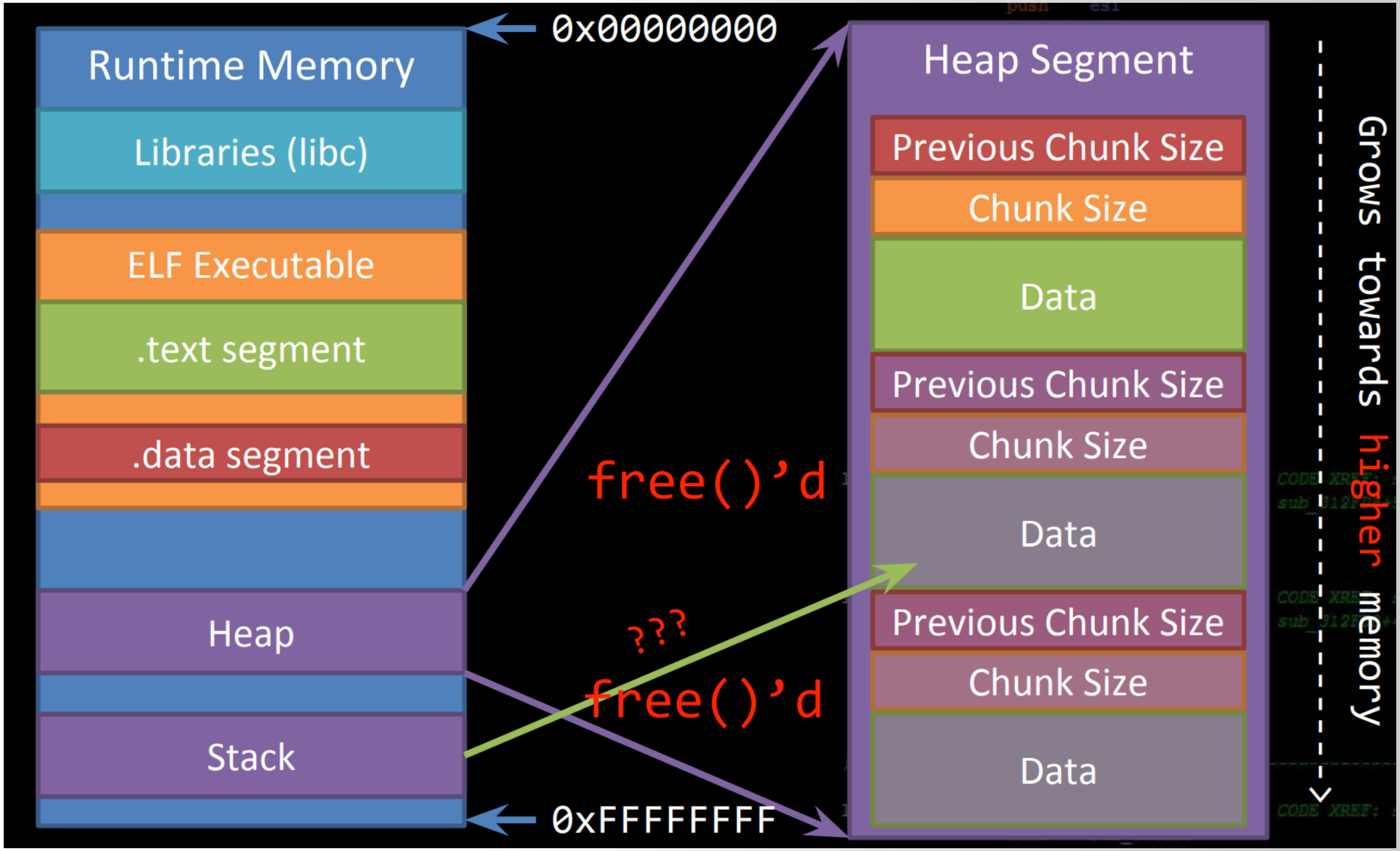
http://security.cs.rpi.edu/courses/binexp-spring2015/lectures/17/10_lecture.pdf

Use After Free



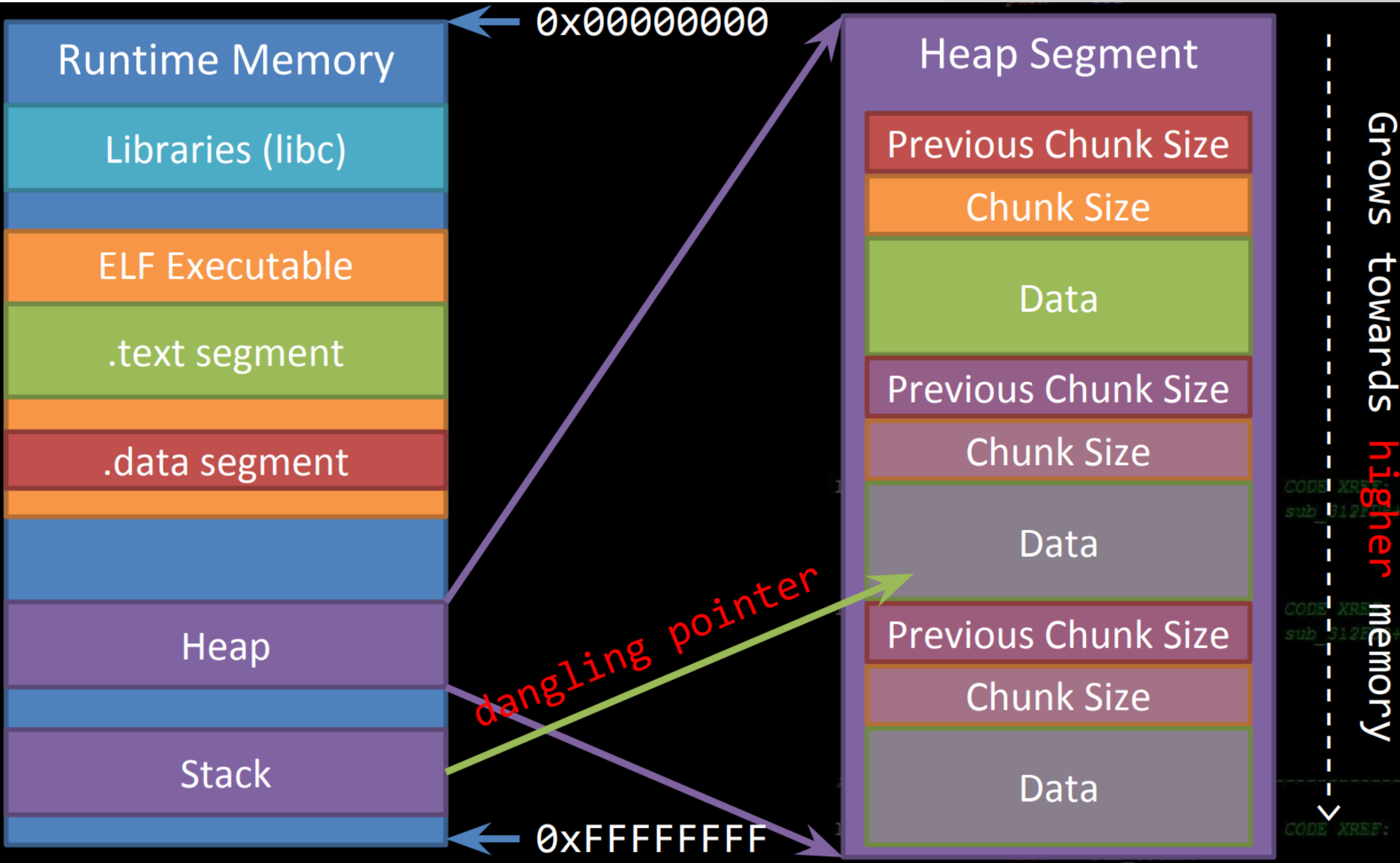
http://security.cs.rpi.edu/courses/binexp-spring2015/lectures/17/10_lecture.pdf

Use After Free



http://security.cs.rpi.edu/courses/binexp-spring2015/lectures/17/10_lecture.pdf

Use After Free



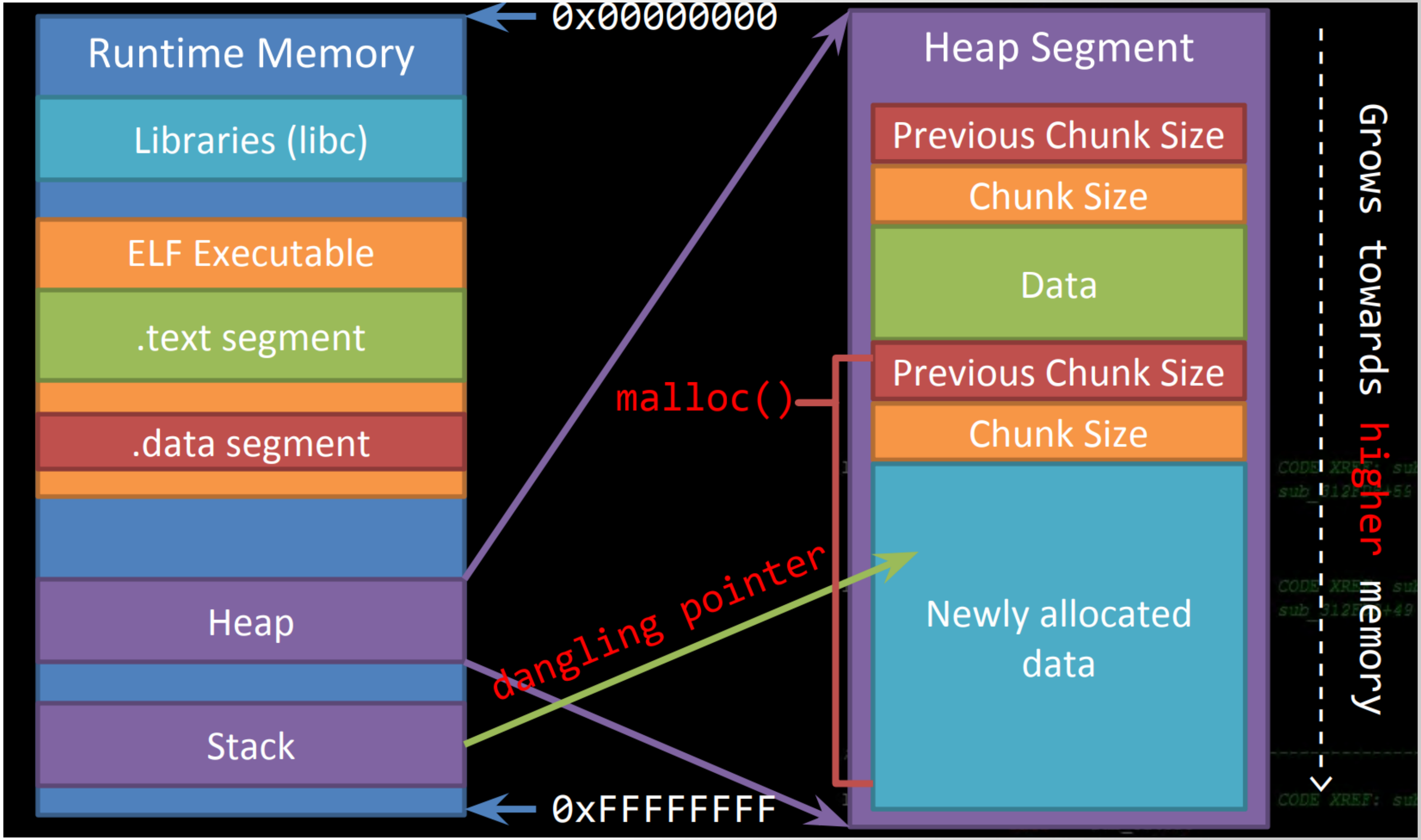
http://security.cs.rpi.edu/courses/binexp-spring2015/lectures/17/10_lecture.pdf

Use After Free

- **Dangling Pointer**
  - A left over pointer in your code that references free’d data and is prone to be re-used
  - As the memory it’s pointing at was freed, there’s no guarantees on what data is there now
  - Also known as **stale pointer, wild pointer**

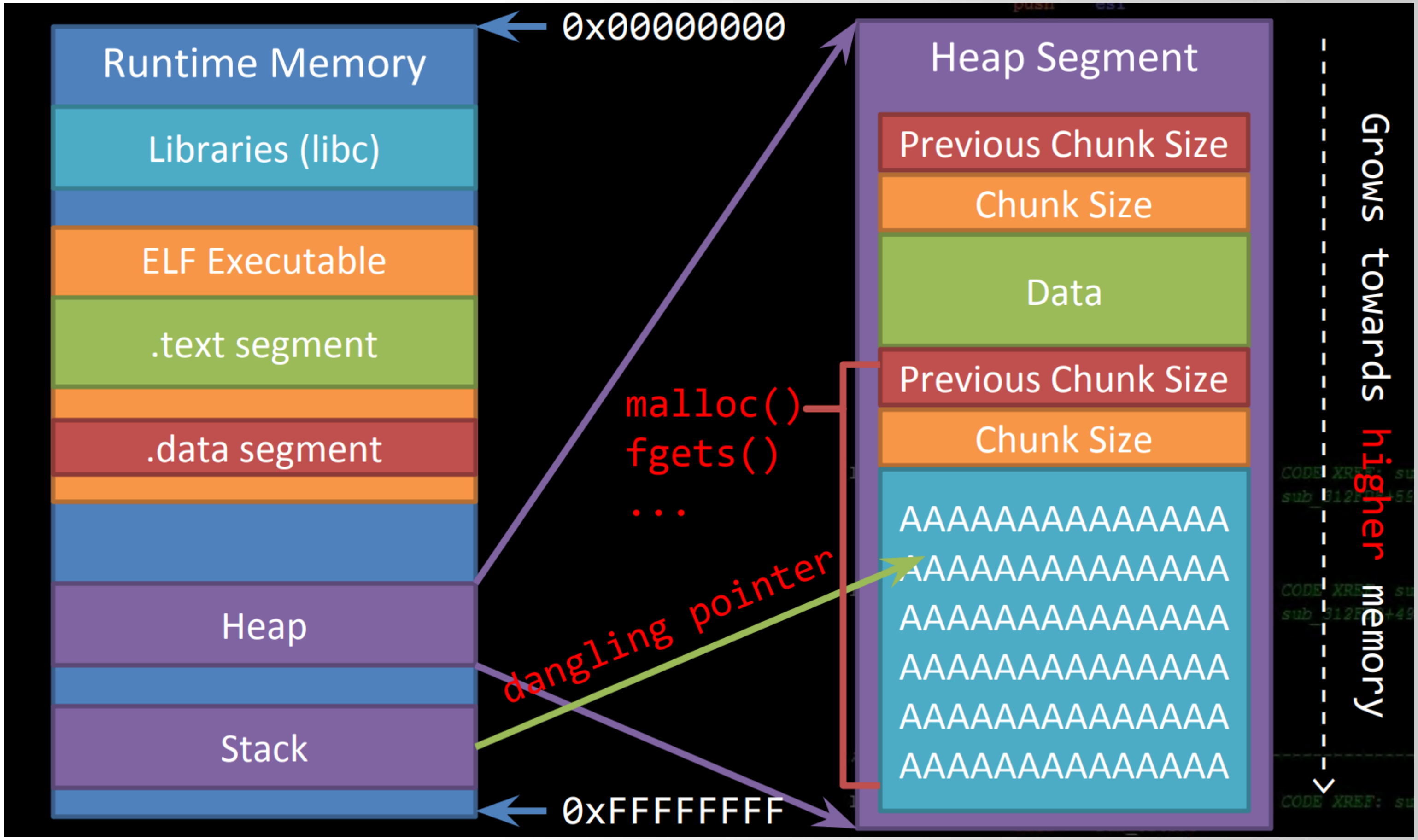
http://security.cs.rpi.edu/courses/binexp-spring2015/lectures/17/10_lecture.pdf

Use After Free



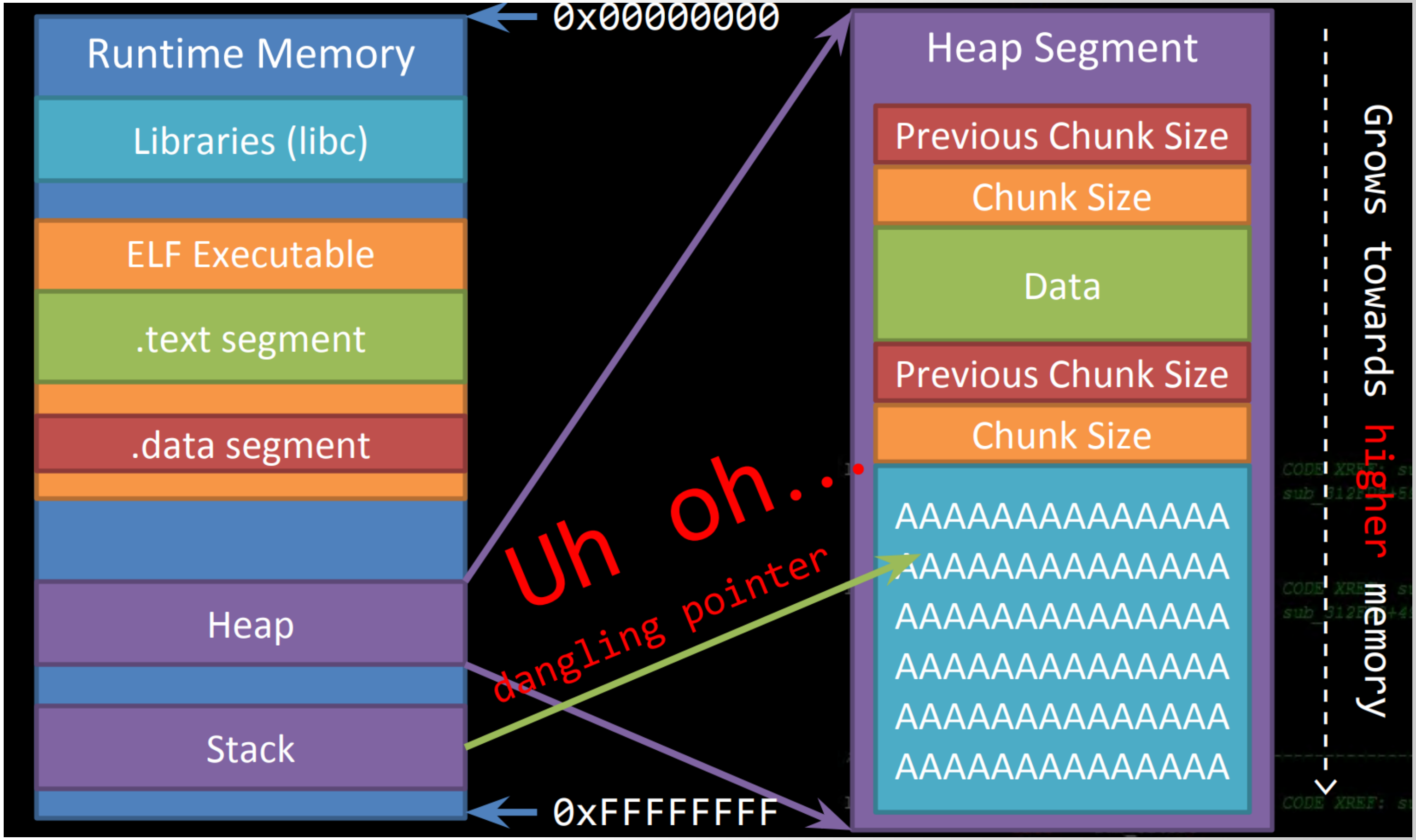
http://security.cs.rpi.edu/courses/binexp-spring2015/lectures/17/10_lecture.pdf

Use After Free



http://security.cs.rpi.edu/courses/binexp-spring2015/lectures/17/10_lecture.pdf

Use After Free



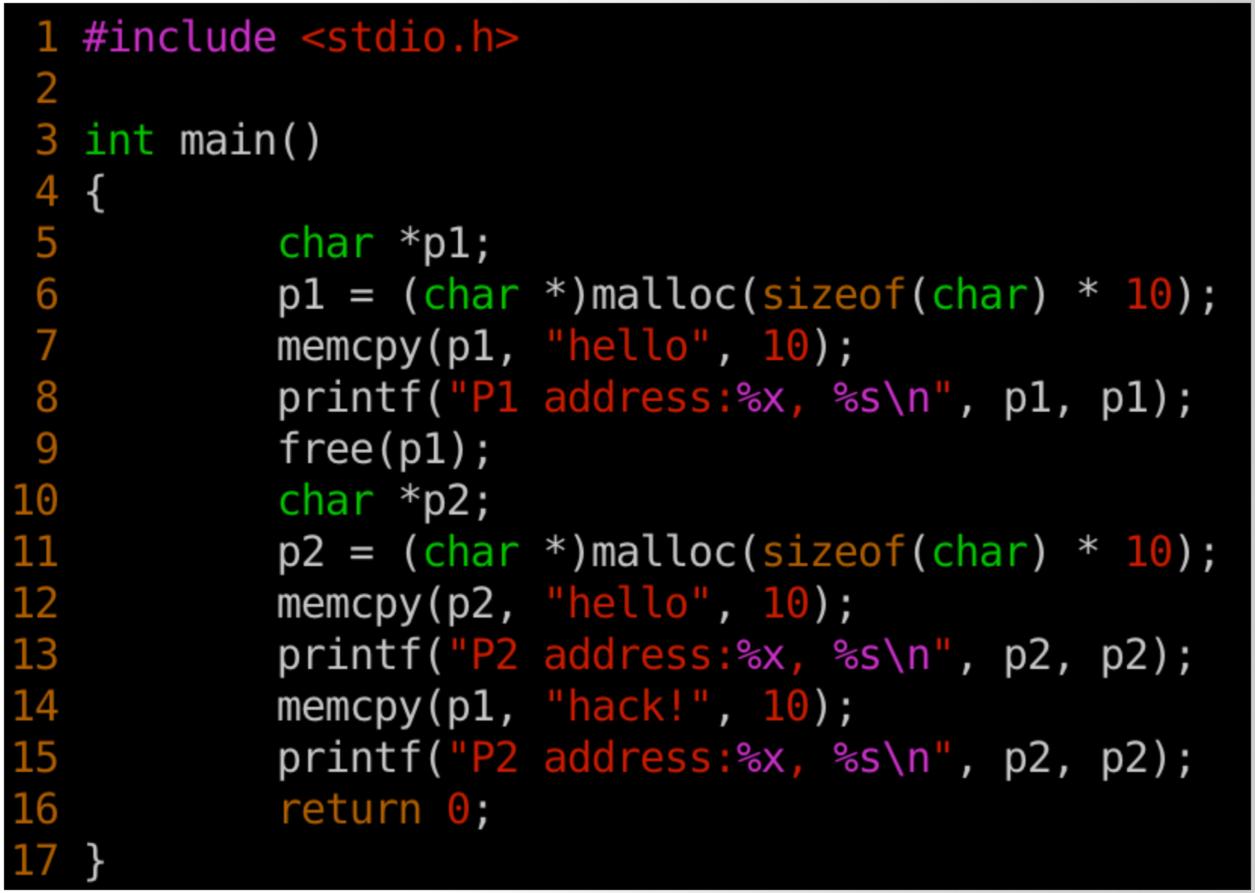
http://security.cs.rpi.edu/courses/binexp-spring2015/lectures/17/10_lecture.pdf

Use After Free

- You actually don’t need any form of memory corruption to leverage a use after free
- It’s simply an implementation issue
  - – pointer mismanagement

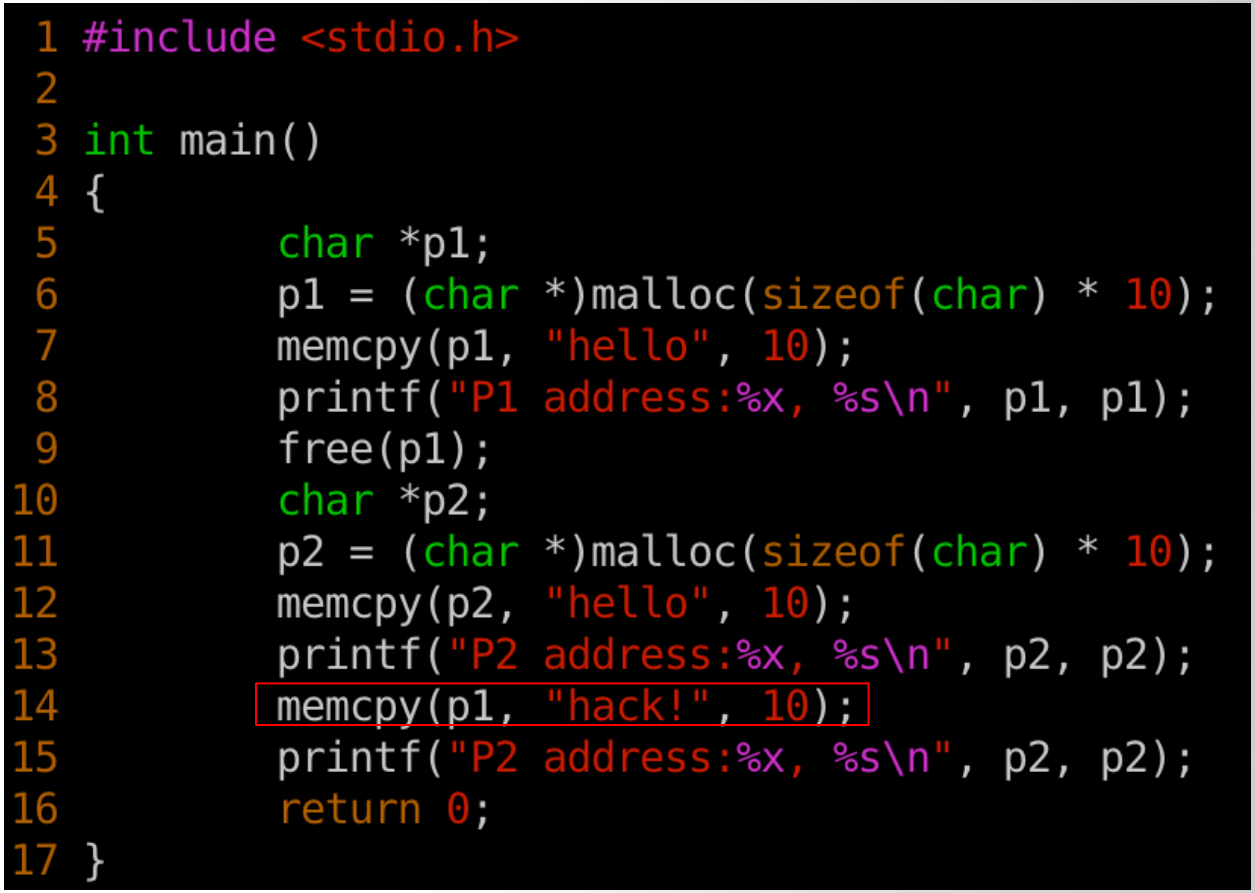
http://security.cs.rpi.edu/courses/binexp-spring2015/lectures/17/10_lecture.pdf

Use After Free: PoC Example

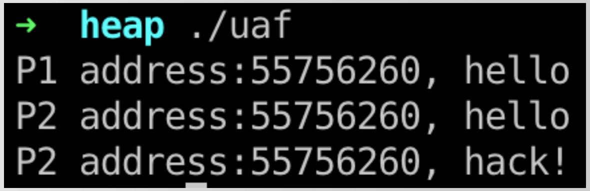


```c
#include <stdio.h>

int main()
{
        char *p1;
        p1 = (char *)malloc(sizeof(char) * 10);
        memcpy(p1, "hello", 10);
        printf("P1 address:%x, %s\n", p1, p1);
        free(p1);
        char *p2;
        p2 = (char *)malloc(sizeof(char) * 10);
        memcpy(p2, "hello", 10);
        printf("P2 address:%x, %s\n", p2, p2);
        memcpy(p1, "hack!", 10);
        printf("P2 address:%x, %s\n", p2, p2);
        return 0;
}
```

Use After Free: PoC Example

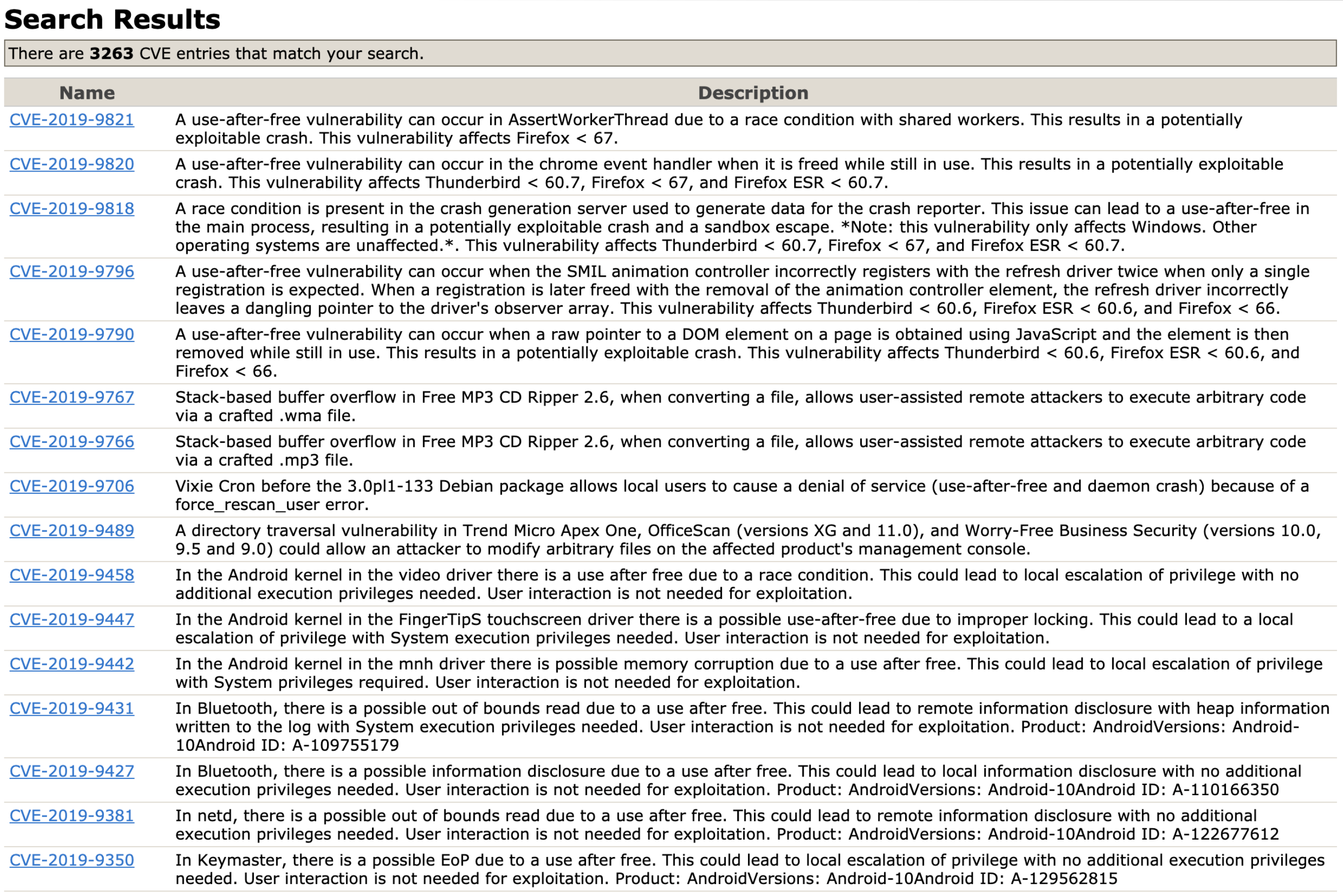


```c
#include <stdio.h>

int main()
{
	char *p1;
	p1 = (char *)malloc(sizeof(char) * 10);
	memcpy(p1, "hello", 10);
	printf("P1 address:%x, %s\n", p1, p1);
	free(p1);
	char *p2;
	p2 = (char *)malloc(sizeof(char) * 10);
	memcpy(p2, "hello", 10);
	printf("P2 address:%x, %s\n", p2, p2);
	memcpy(p1, "hack!", 10);
	printf("P2 address:%x, %s\n", p2, p2);
	return 0;
}
```



```
→  heap ./uaf
P1 address:55756260, hello
P2 address:55756260, hello
P2 address:55756260, hack!
```

Use After Free



Search Results

There are **3263** CVE entries that match your search.

Name	Description
CVE-2019-9821	A use-after-free vulnerability can occur in AssertWorkerThread due to a race condition with shared workers. This results in a potentially exploitable crash. This vulnerability affects Firefox < 67.
CVE-2019-9820	A use-after-free vulnerability can occur in the chrome event handler when it is freed while still in use. This results in a potentially exploitable crash. This vulnerability affects Thunderbird < 60.7, Firefox < 67, and Firefox ESR < 60.7.
CVE-2019-9818	A race condition is present in the crash generation server used to generate data for the crash reporter. This issue can lead to a use-after-free in the main process, resulting in a potentially exploitable crash and a sandbox escape. *Note: this vulnerability only affects Windows. Other operating systems are unaffected.*. This vulnerability affects Thunderbird < 60.7, Firefox < 67, and Firefox ESR < 60.7.
CVE-2019-9796	A use-after-free vulnerability can occur when the SMIL animation controller incorrectly registers with the refresh driver twice when only a single registration is expected. When a registration is later freed with the removal of the animation controller element, the refresh driver incorrectly leaves a dangling pointer to the driver's observer array. This vulnerability affects Thunderbird < 60.6, Firefox ESR < 60.6, and Firefox < 66.
CVE-2019-9790	A use-after-free vulnerability can occur when a raw pointer to a DOM element on a page is obtained using JavaScript and the element is then removed while still in use. This results in a potentially exploitable crash. This vulnerability affects Thunderbird < 60.6, Firefox ESR < 60.6, and Firefox < 66.
CVE-2019-9767	Stack-based buffer overflow in Free MP3 CD Ripper 2.6, when converting a file, allows user-assisted remote attackers to execute arbitrary code via a crafted .wma file.
CVE-2019-9766	Stack-based buffer overflow in Free MP3 CD Ripper 2.6, when converting a file, allows user-assisted remote attackers to execute arbitrary code via a crafted .mp3 file.
CVE-2019-9706	Vixie Cron before the 3.0pl1-133 Debian package allows local users to cause a denial of service (use-after-free and daemon crash) because of a force_rescan_user error.
CVE-2019-9489	A directory traversal vulnerability in Trend Micro Apex One, OfficeScan (versions XG and 11.0), and Worry-Free Business Security (versions 10.0, 9.5 and 9.0) could allow an attacker to modify arbitrary files on the affected product's management console.
CVE-2019-9458	In the Android kernel in the video driver there is a use after free due to a race condition. This could lead to local escalation of privilege with no additional execution privileges needed. User interaction is not needed for exploitation.
CVE-2019-9447	In the Android kernel in the FingerTipS touchscreen driver there is a possible use-after-free due to improper locking. This could lead to a local escalation of privilege with System execution privileges needed. User interaction is not needed for exploitation.
CVE-2019-9442	In the Android kernel in the mnh driver there is possible memory corruption due to a use after free. This could lead to local escalation of privilege with System privileges required. User interaction is not needed for exploitation.
CVE-2019-9431	In Bluetooth, there is a possible out of bounds read due to a use after free. This could lead to remote information disclosure with heap information written to the log with System execution privileges needed. User interaction is not needed for exploitation. Product: AndroidVersions: Android-10Android ID: A-109755179
CVE-2019-9427	In Bluetooth, there is a possible information disclosure due to a use after free. This could lead to local information disclosure with no additional execution privileges needed. User interaction is not needed for exploitation. Product: AndroidVersions: Android-10Android ID: A-110166350
CVE-2019-9381	In netd, there is a possible out of bounds read due to a use after free. This could lead to remote information disclosure with no additional execution privileges needed. User interaction is not needed for exploitation. Product: AndroidVersions: Android-10Android ID: A-122677612
CVE-2019-9350	In Keymaster, there is a possible EoP due to a use after free. This could lead to local escalation of privilege with no additional execution privileges needed. User interaction is not needed for exploitation. Product: AndroidVersions: Android-10Android ID: A-129562815

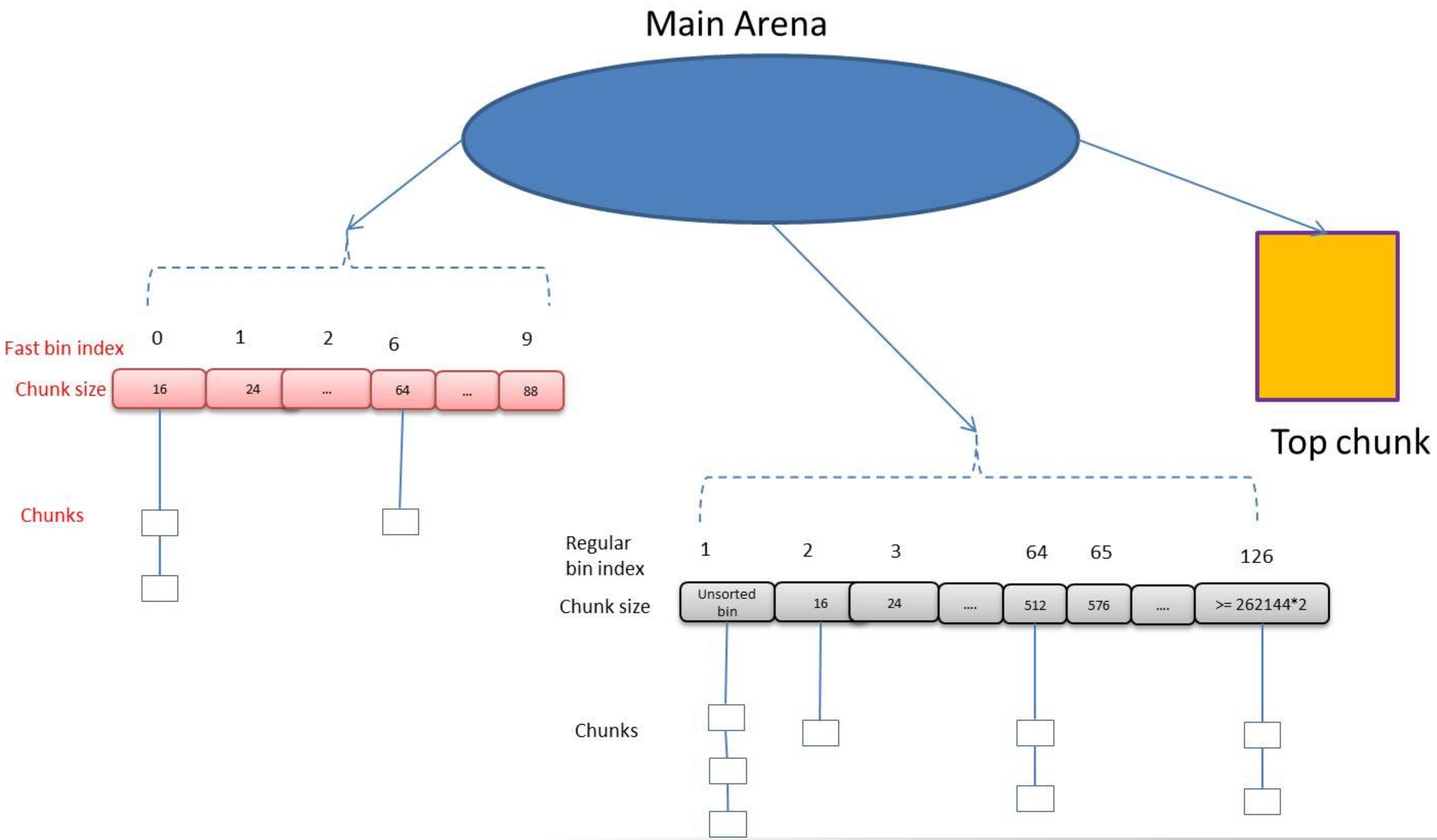
The ‘hot’ vulnerability nowadays, almost every modern browser exploit leverages a UAF

Use After Free

- From the defensive perspective, trying to detect use after free vulnerabilities in complex applications is **very difficult**, even in industry
- Why?
  - UAF's **only exist in certain states of execution**, so statically scanning source for them won't go far
  - They're **usually only found through crashes**, but symbolic execution and constraint solvers are helping find these bugs faster

Double Free

- **Double Free**
  - Freeing a resource more than once can lead to memory leaks.
  - The allocator's data structures get corrupted and can be exploited by an attacker.


Main Arena
Fast bin index
0
1
2
6
9
Chunk size
16
24
...
64
...
88
Chunks
Top chunk
Regular bin index
1
2
3
64
65
126
Chunk size
Unsorted bin
16
24
....
512
576
....
>= 262144*2
Chunks

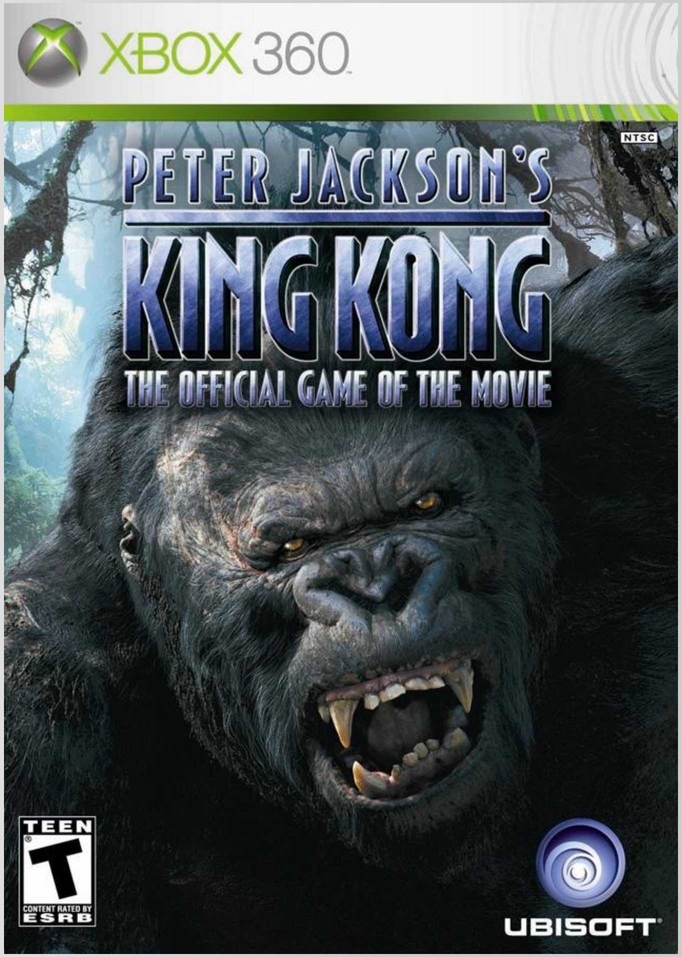
Game Consoles

- Evolving entertainment platforms
  - Play games, stream media, browse the web
- 100% consistent machine for developers
  - Don’t have to account for different specs (eg. PC’ s)
- Enforces DRM much better than PC’s can
  - It’s a controlled platform that only runs code as blessed by Sony, Microsoft, Nintendo

Xbox 360 – Nov. 2005

Xbox 360 – Nov. 2005

- Security Perspective
  - Only runs signed code or executables
  - Rigorous chain of trust, secure bootstrapping
  - Encrypted runtime memory
  - eFuses to enforce updates (these are awesome)
  - NX/DEP
  - No ASLR



KING KONG EXPLOIT

updates don’t always patch bugs, sometimes they introduce them

King Kong Exploit – Dec. 2006

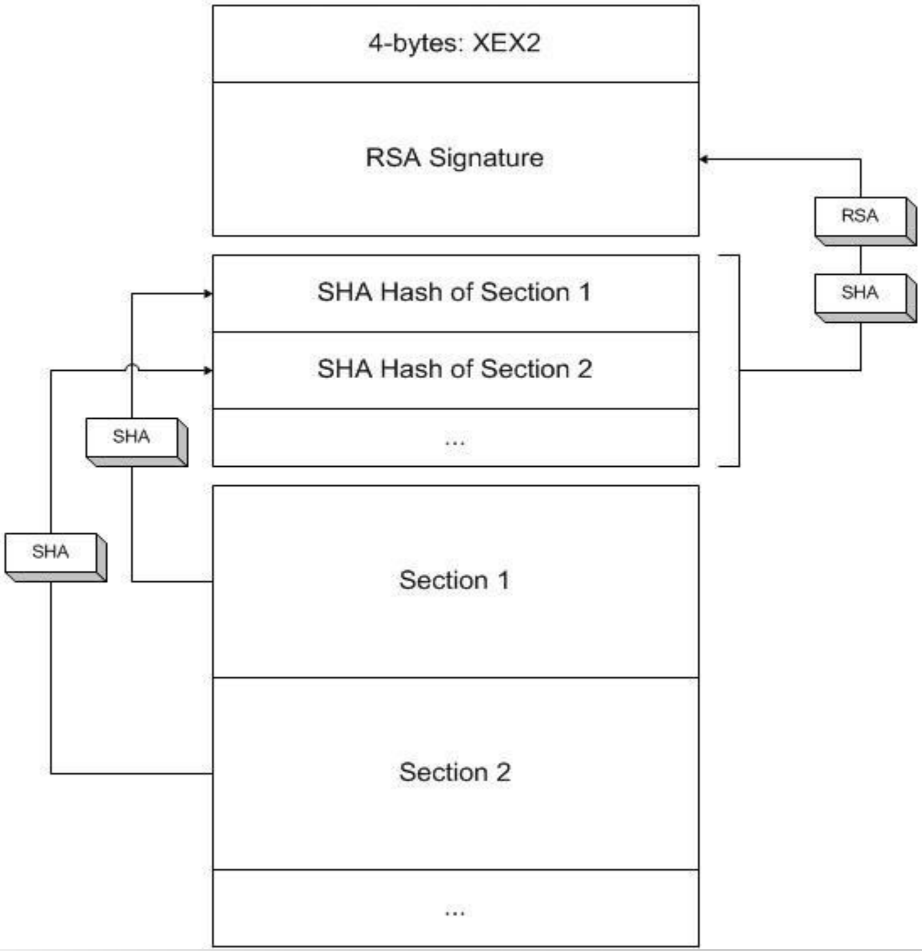
- Integer based bug, resulting in code execution at the Hypervisor context
  - Complete system control
- The bug leveraged by the King Kong Exploit was INTRODUCED in kernel version 4532, and patched two updates later in v4552
  - For reference, the Xbox 360 shipped on v1888

About the Xbox 360 & Games

- All executables (.XEX's) are signed by Microsoft which the system verifies to prevent tampering with code

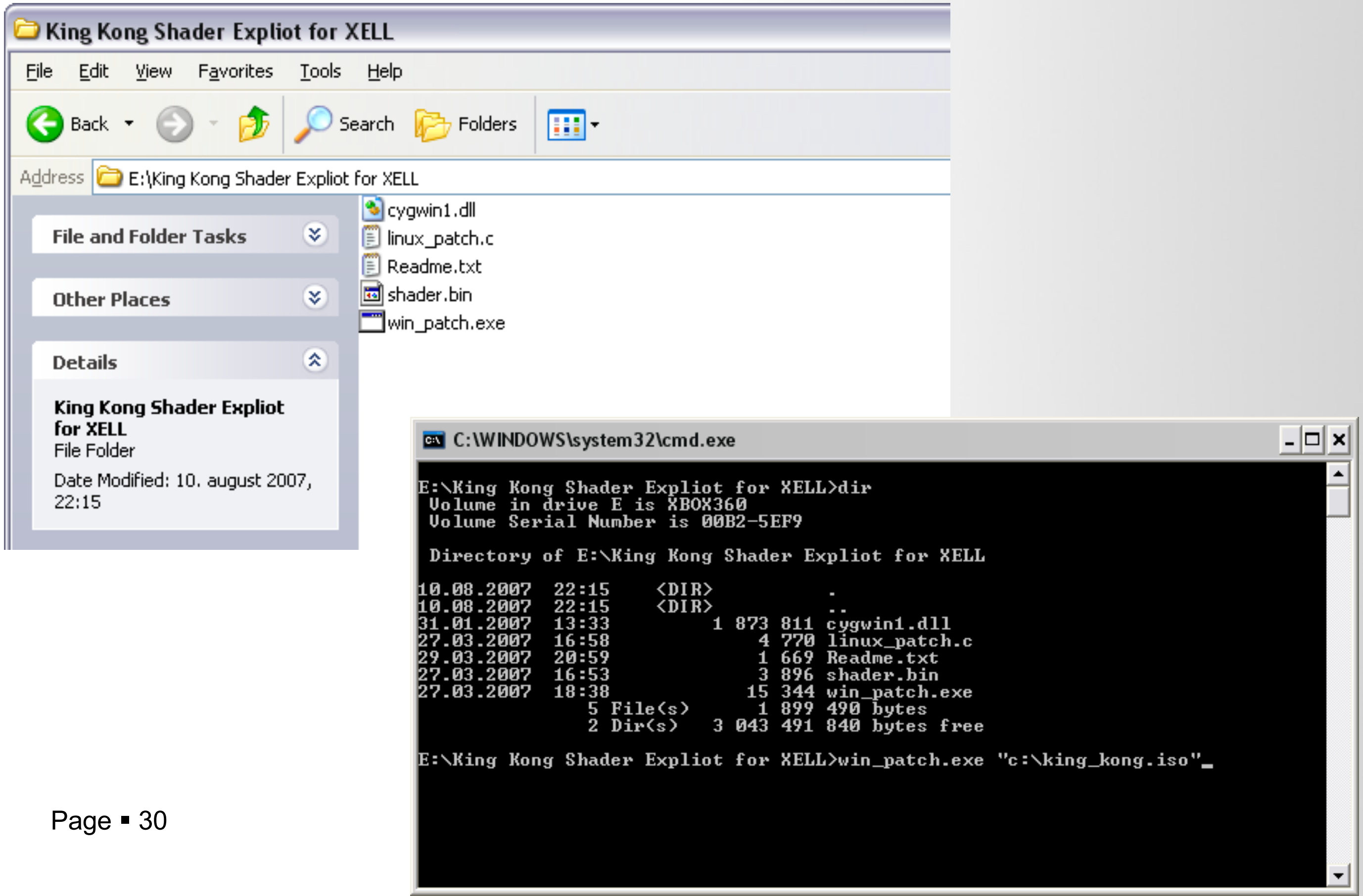
- Data assets such as textures, models, shaders, and audio as used by games are **NOT** signed!
  - Find bugs in game asset parsers

Figure 1. The Cryptographic Anatomy of a XEX File



Stage One: King Kong's Role

- A **maliciously crafted unsigned shader file parsed by the signed King Kong game XEX**, can lead to *unprivileged code execution* on the system
- King Kong was one of many possible memory corruption vectors that could have been used to get basic code exec



King Kong Shader Expliot for XELL

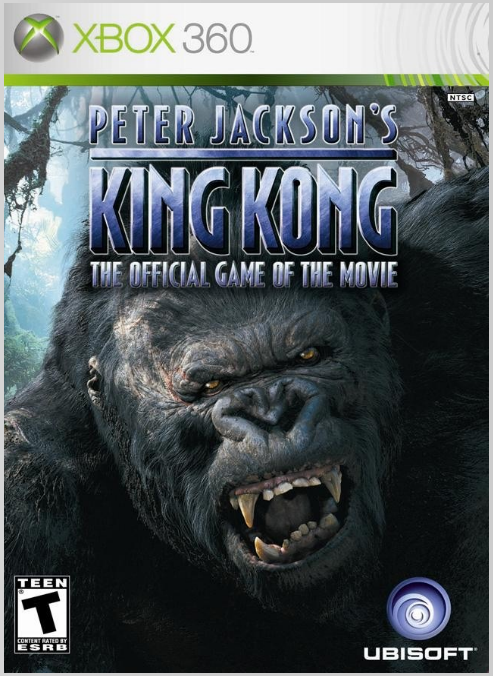
E:\King Kong Shader Expliot for XELL

cygwin1.dll
linux_patch.c
Readme.txt
shader.bin
win_patch.exe

King Kong Shader Expliot for XELL
File Folder
Date Modified: 10. august 2007, 22:15

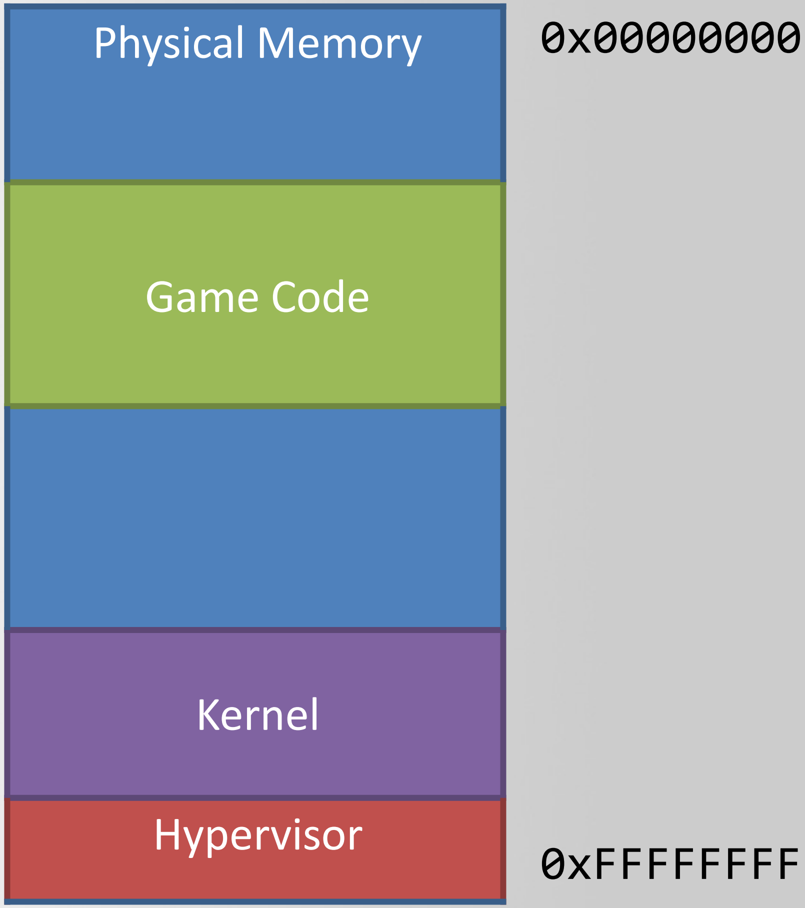
C:\WINDOWS\system32\cmd.exe

```
E:\King Kong Shader Expliot for XELL>dir
 Volume in drive E is XBOX360
 Volume Serial Number is 00B2-5EF9

 Directory of E:\King Kong Shader Expliot for XELL

10.08.2007  22:15    <DIR>          .
10.08.2007  22:15    <DIR>          ..
31.01.2007  13:33         1 873 811 cygwin1.dll
27.03.2007  16:58             4 770 linux_patch.c
29.03.2007  20:59             1 669 Readme.txt
27.03.2007  16:53             3 896 shader.bin
27.03.2007  18:38            15 344 win_patch.exe
               5 File(s)      1 899 490 bytes
               2 Dir(s)   3 043 491 840 bytes free

E:\King Kong Shader Expliot for XELL>win_patch.exe "c:\king_kong.iso"_
```



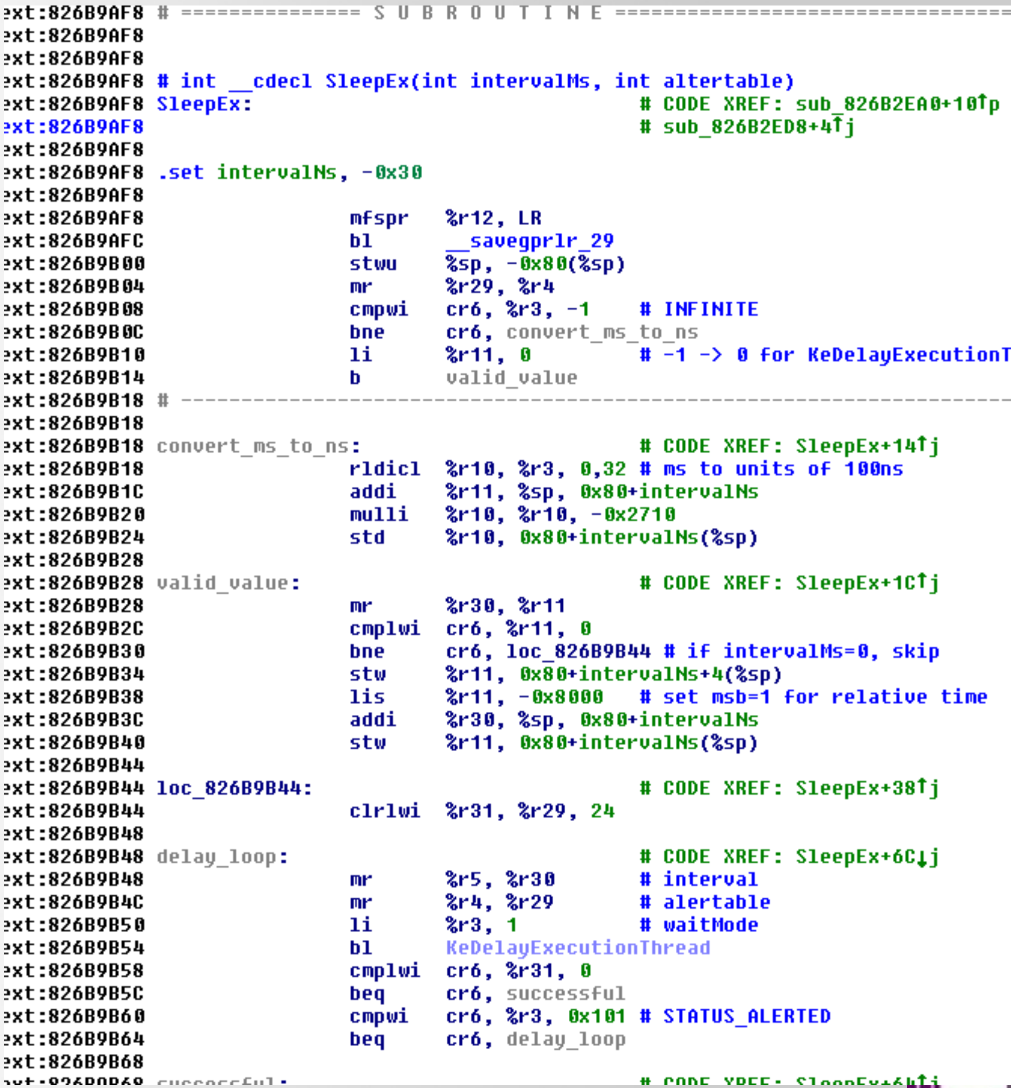
About the Xbox 360 Hypervisor

- A small Hypervisor (Hv) sits next to the kernel, near the top of memory
- The Hv handles some crypto keys, low level IO, memory encryption/decryption operations and more
- If you can take over the Hv, you have access to physmem and the highest privilege of execution



- The PPC instruction ‘sc’ is used to make system calls on the Xbox 360, the Hv handles these calls as they are made

- Unfortunately, along came a bug in the syscall handler):



```
ext:826B9AF8 # =============== S U B R O U T I N E ================================
ext:826B9AF8
ext:826B9AF8
ext:826B9AF8 # int __cdecl SleepEx(int intervalMs, int altertable)
ext:826B9AF8 SleepEx:                                # CODE XREF: sub_826B2EA0+10↑p
ext:826B9AF8                                         # sub_826B2ED8+4↑j
ext:826B9AF8
ext:826B9AF8 .set intervalNs, -0x30
ext:826B9AF8
ext:826B9AF8                 mfspr   %r12, LR
ext:826B9AFC                 bl      __savegprlr_29
ext:826B9B00                 stwu    %sp, -0x80(%sp)
ext:826B9B04                 mr      %r29, %r4
ext:826B9B08                 cmpwi   cr6, %r3, -1    # INFINITE
ext:826B9B0C                 bne     cr6, convert_ms_to_ns
ext:826B9B10                 li      %r11, 0         # -1 -> 0 for KeDelayExecutionT
ext:826B9B14                 b       valid_value
ext:826B9B18 # ---------------------------------------------------------------------
ext:826B9B18
ext:826B9B18 convert_ms_to_ns:                       # CODE XREF: SleepEx+14↑j
ext:826B9B18                 rldicl  %r10, %r3, 0,32 # ms to units of 100ns
ext:826B9B1C                 addi    %r11, %sp, 0x80+intervalNs
ext:826B9B20                 mulli   %r10, %r10, -0x2710
ext:826B9B24                 std     %r10, 0x80+intervalNs(%sp)
ext:826B9B28
ext:826B9B28 valid_value:                            # CODE XREF: SleepEx+1C↑j
ext:826B9B28                 mr      %r30, %r11
ext:826B9B2C                 cmplwi  cr6, %r11, 0
ext:826B9B30                 bne     cr6, loc_826B9B44 # if intervalMs=0, skip
ext:826B9B34                 stw     %r11, 0x80+intervalNs+4(%sp)
ext:826B9B38                 lis     %r11, -0x8000   # set msb=1 for relative time
ext:826B9B3C                 addi    %r30, %sp, 0x80+intervalNs
ext:826B9B40                 stw     %r11, 0x80+intervalNs(%sp)
ext:826B9B44
ext:826B9B44 loc_826B9B44:                           # CODE XREF: SleepEx+38↑j
ext:826B9B44                 clrlwi  %r31, %r29, 24
ext:826B9B48
ext:826B9B48 delay_loop:                             # CODE XREF: SleepEx+6C↓j
ext:826B9B48                 mr      %r5, %r30       # interval
ext:826B9B4C                 mr      %r4, %r29       # alertable
ext:826B9B50                 li      %r3, 1          # waitMode
ext:826B9B54                 bl      KeDelayExecutionThread
ext:826B9B58                 cmplwi  cr6, %r31, 0
ext:826B9B5C                 beq     cr6, successful
ext:826B9B60                 cmpwi   cr6, %r3, 0x101 # STATUS_ALERTED
ext:826B9B64                 beq     cr6, delay_loop
ext:826B9B68
ext:826B9B68 successful:                             # CODE XREF: SleepEx+64↑j
```

Pseudocode of the Hv Bug

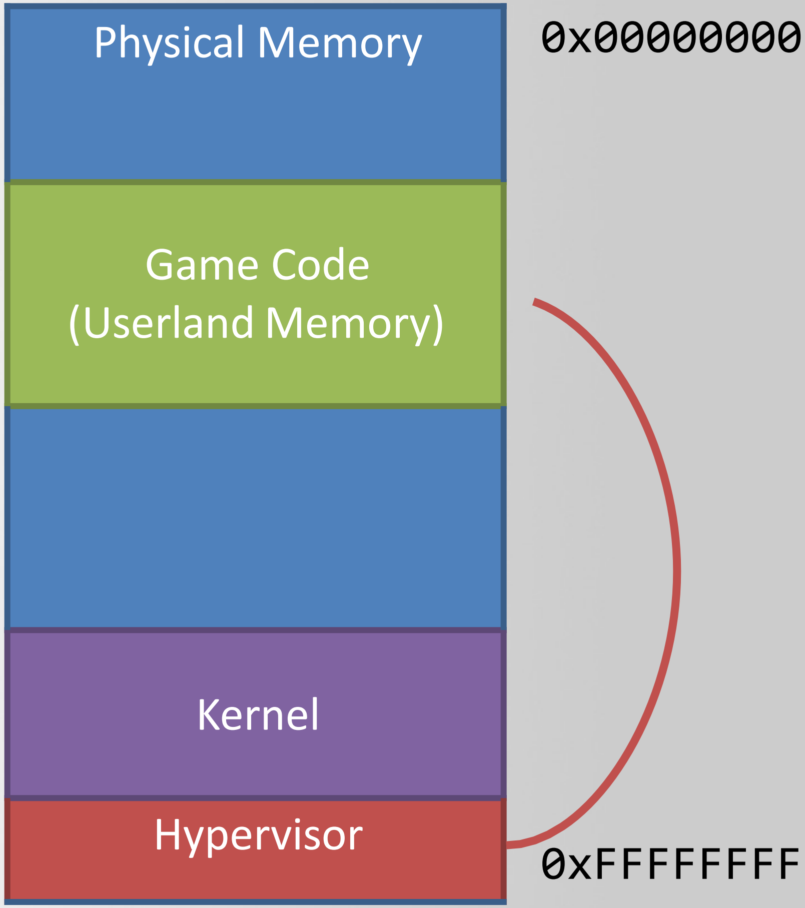
```
int syscall_handler(uint64_t syscall_num, ...)
{

   /* check for invalid syscall */
   if((uint32_t)syscall_num > 0x61)
      return 0;

   /* call the respective syscall func */
   syscall_table[syscall_num](...);
   ...
```

The Oops

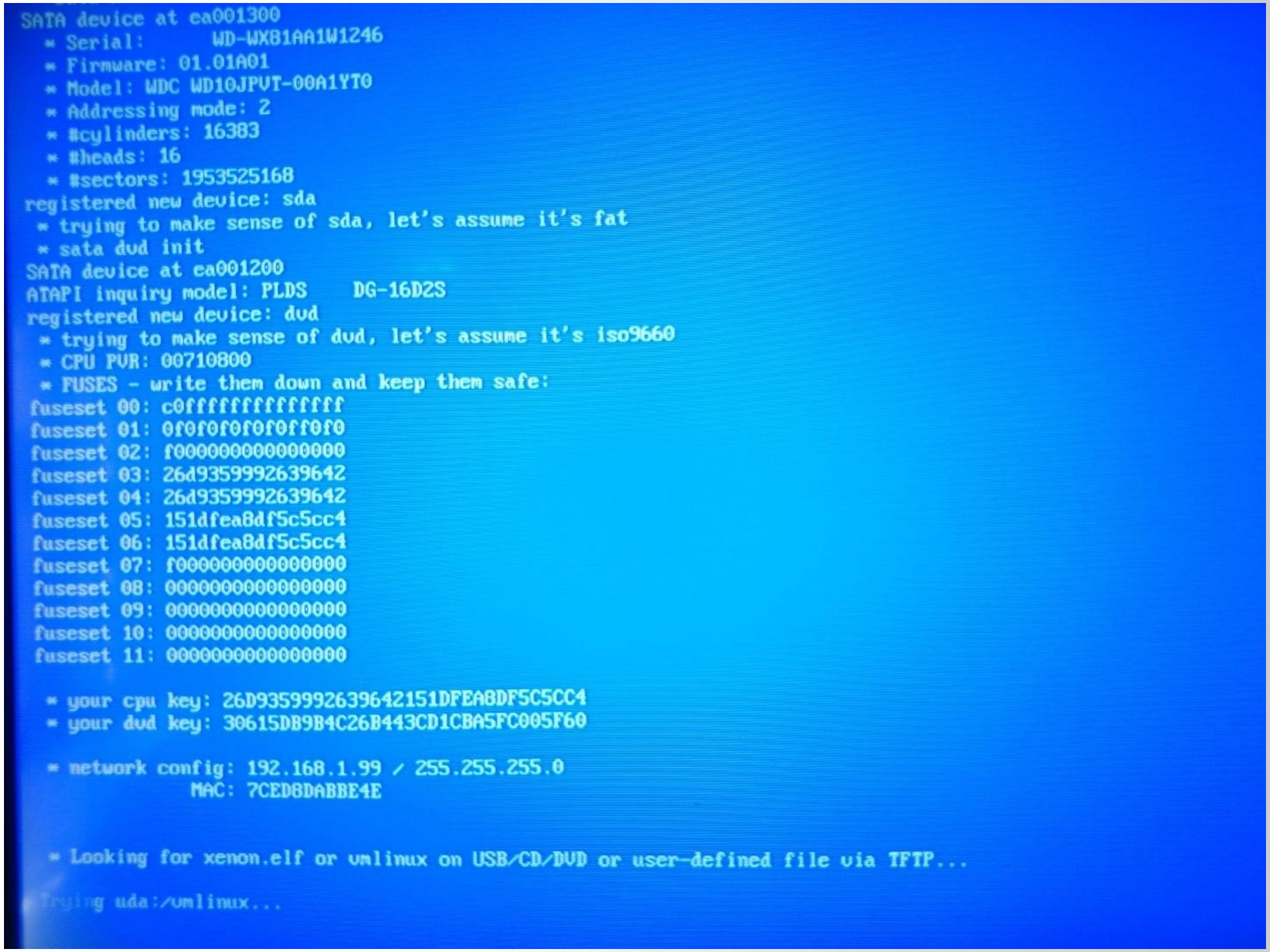
- Only the lower 32 bits of the syscall number are sanity checked
- The whole 64 bit number is used in address calculation

```
syscall_table[syscall_num](...);
```

```
Arbitrary jump into userland
memory/code at the HV Context
```



Game Over



```
SATA device at ea001300
  * Serial:      WD-WXB1AA1W1246
  * Firmware: 01.01A01
  * Model: WDC WD10JPVT-00A1YT0
  * Addressing mode: 2
  * #cylinders: 16383
  * #heads: 16
  * #sectors: 1953525168
registered new device: sda
 * trying to make sense of sda, let's assume it's fat
 * sata dvd init
SATA device at ea001200
ATAPI inquiry model: PLDS    DG-16D2S
registered new device: dvd
 * trying to make sense of dvd, let's assume it's iso9660
 * CPU PVR: 00710800
 * FUSES - write them down and keep them safe:
fuseset 00: c0ffffffffffffff
fuseset 01: 0f0f0f0f0f0ff0f0
fuseset 02: f000000000000000
fuseset 03: 26d9359992639642
fuseset 04: 26d9359992639642
fuseset 05: 151dfea8df5c5cc4
fuseset 06: 151dfea8df5c5cc4
fuseset 07: f000000000000000
fuseset 08: 0000000000000000
fuseset 09: 0000000000000000
fuseset 10: 0000000000000000
fuseset 11: 0000000000000000

 * your cpu key: 26D9359992639642151DFEA8DF5C5CC4
 * your dvd key: 30615DB9B4C26B443CD1CBA5FC005F60

 * network config: 192.168.1.99 / 255.255.255.0
             MAC: 7CED8DABBE4E


 * Looking for xenon.elf or vmlinux on USB/CD/DVD or user-defined file via TFTP...

Trying uda:/vmlinux...
```

XBOX 360 HARDWARE ATTACKS

Straying from binary exploitation, but still interesting

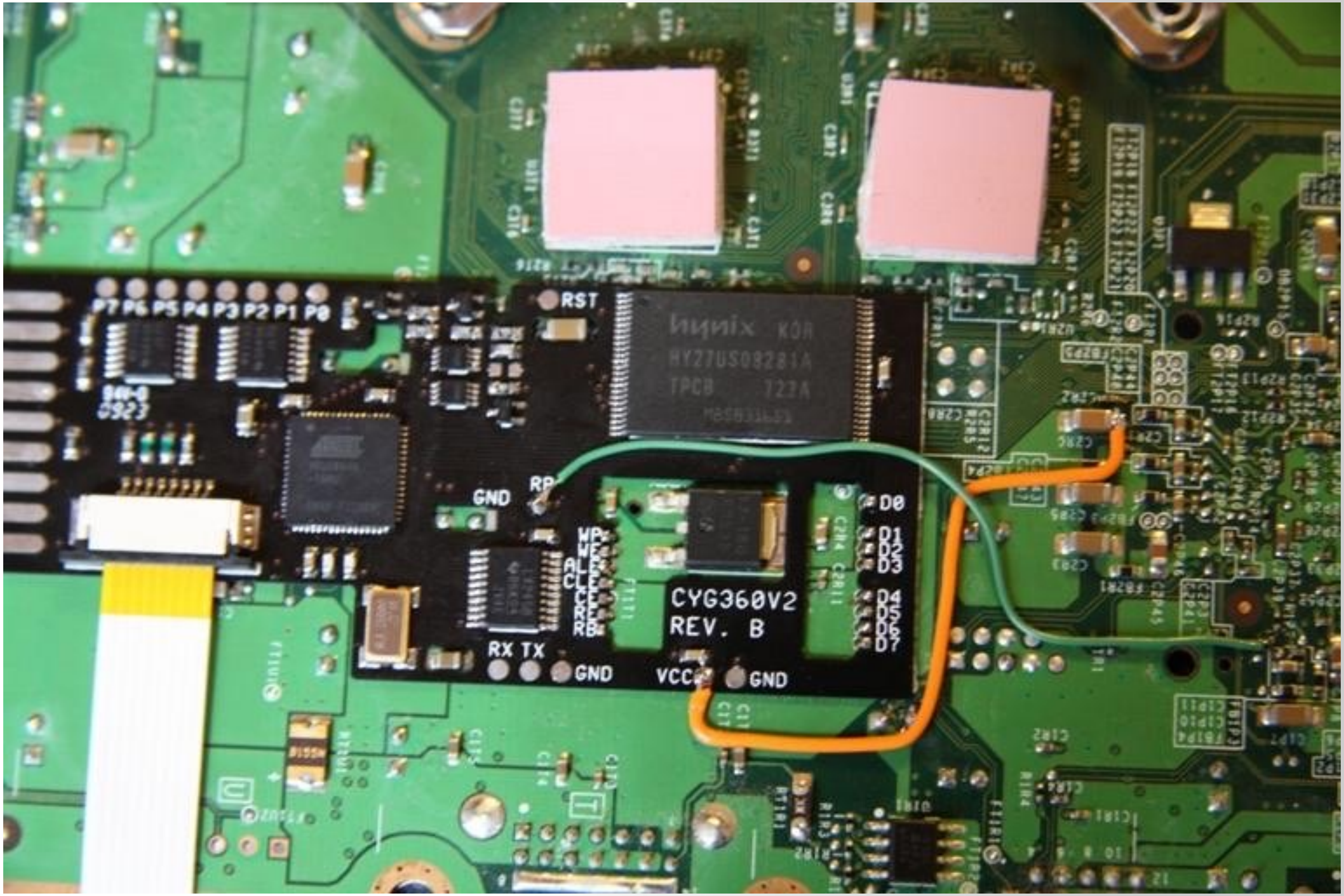
SMC / JTAG Hack – 2007-2009

- Uses the SMC and JTAG to trigger a DMA overwrite instantly at bootup rather than having to load a game such a King Kong

- Cat and mouse for a few years, allowing hackers to boot into downgraded, exploitable kernels (eg v4532)

- Eventually Patched by MS when they decided to rework the boot process

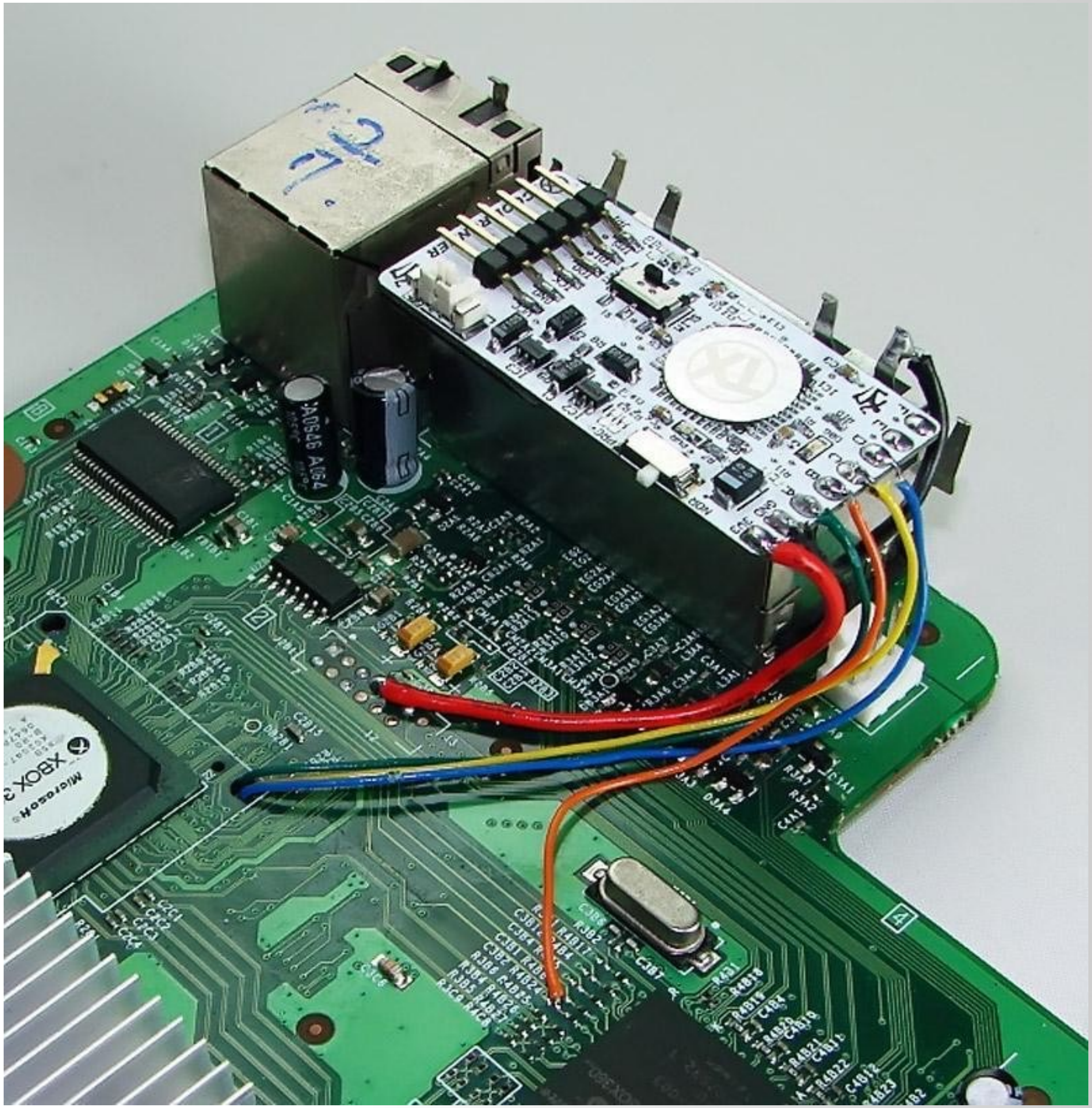
SMC / JTAG Hack



Reset Glitch Hack (RGH) – Aug. 2011

- There’s some hash checks that expect a 0 to be returned for a good hash, or 1 for a hash mismatch (fail)
- Sending a specific reset signal down a pin on the CPU clears the CPU registers
- Reset the registers as the hash check returns

Xbox 360 Reset Glitch Hack (RGH)

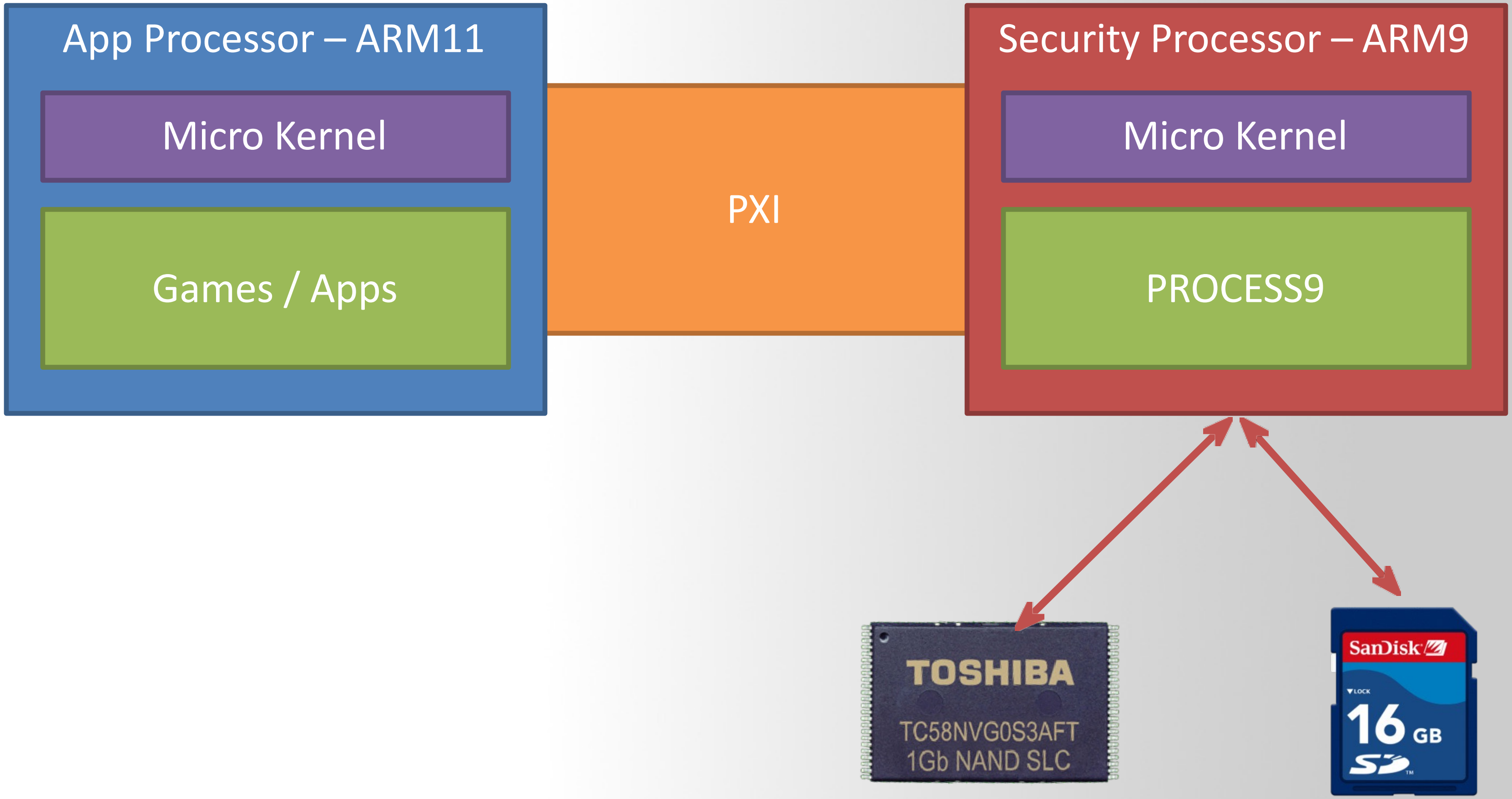


Nintendo 3DS – Feb. 2011

Nintendo 3DS – Feb. 2011

- Security Perspective
  - Very tightly sealed bootrom, hardware disabled
  - Only runs signed code or executables
  - Hardware based keyscrambler for crypto keys
  - NX/DEP (Only used on the ARM11 Core)
  - Runtime memory is not encrypted
  - Has eFuses, not really used
  - No ASLR

Nintendo 3DS Architecture



PWNING OVER THE PXI

Owning the SysCore through the PXI

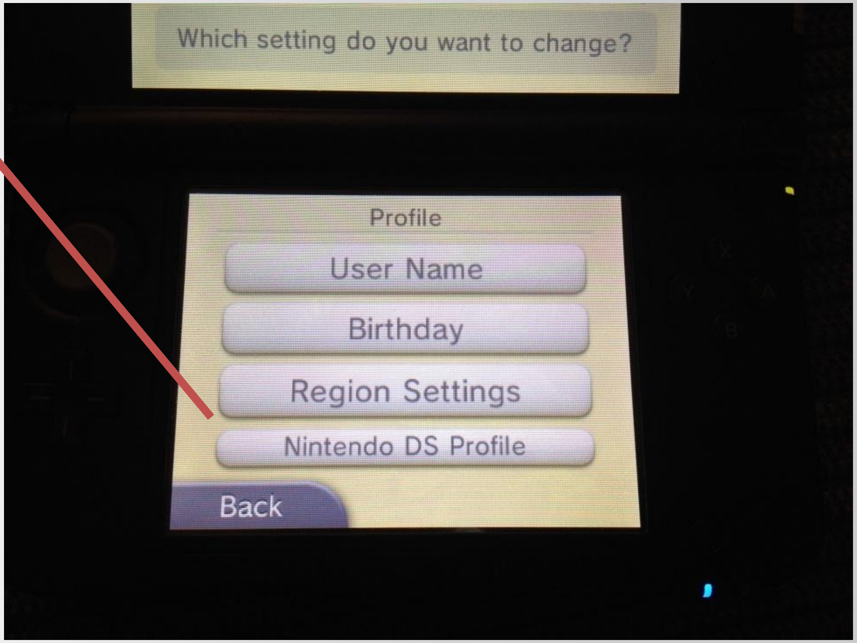
VerifyRsaSha256() – Jun. 2013

- Straight stack smash bug, results in code execution on the Security Processor (ARM9)
  - Complete system control
- Present from firmware version 1.0.0 – 4.5.0
- Bug discovered in 2012

Stage One: ARM11 Code Exec

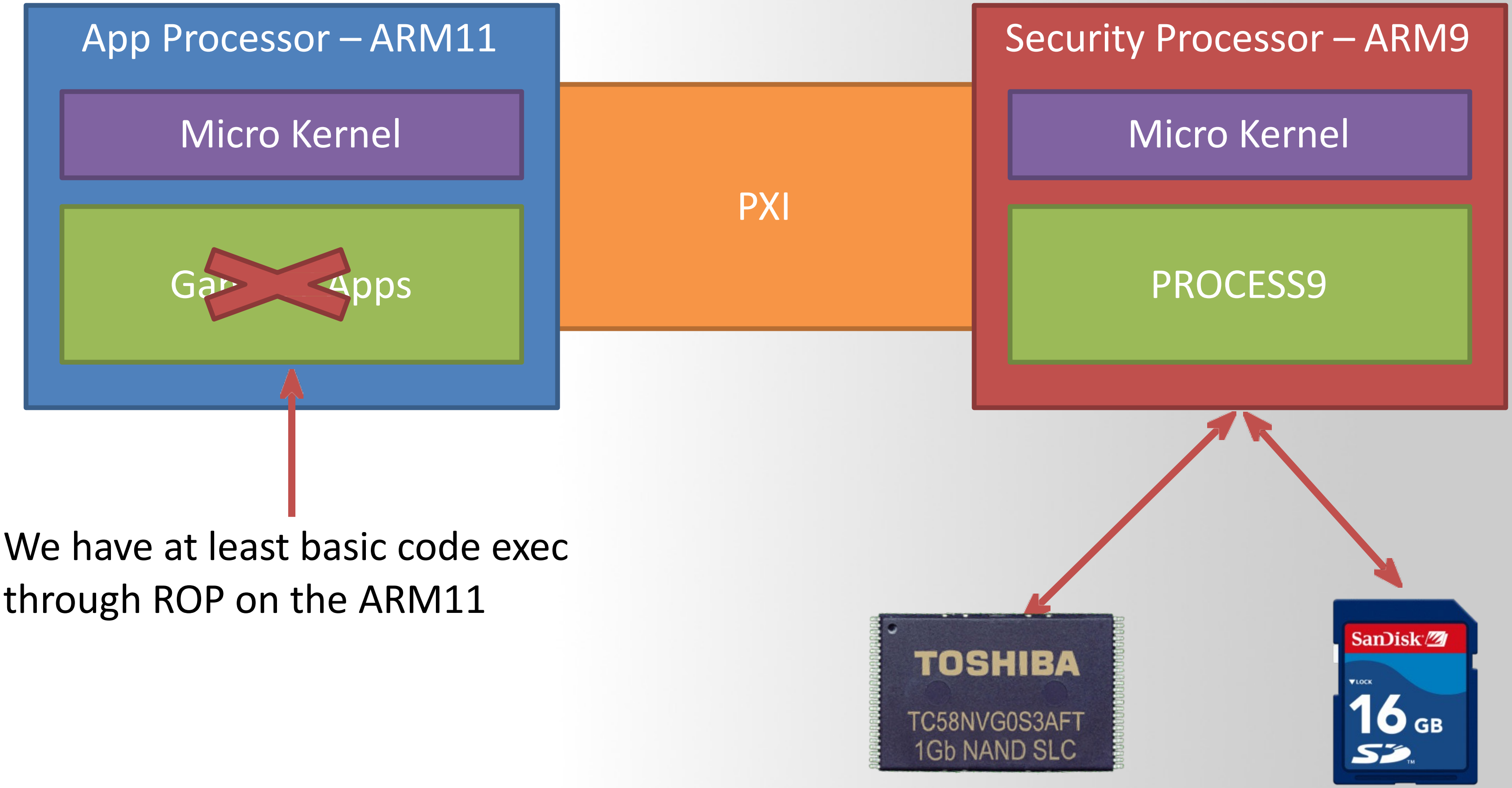
- A stack smash exists in the DS Profile fields in the native settings application on all 3DS's at the time. No need for any games!

- This is a straight stack smash that will get us control, but there is DEP on the ARM11 so you must ROP


Which setting do you want to change?

Profile

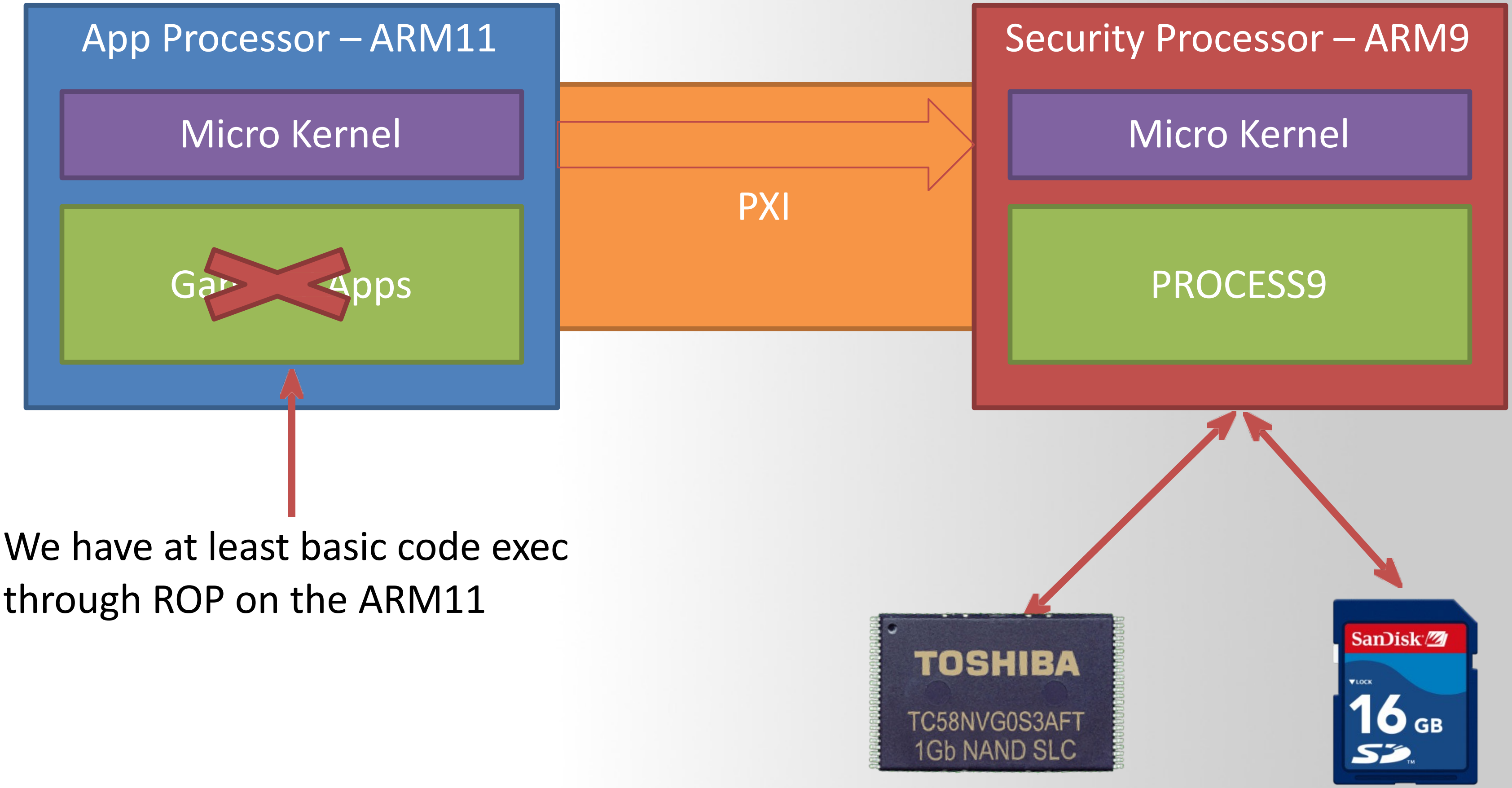
State of Control



We have at least basic code exec through ROP on the ARM11

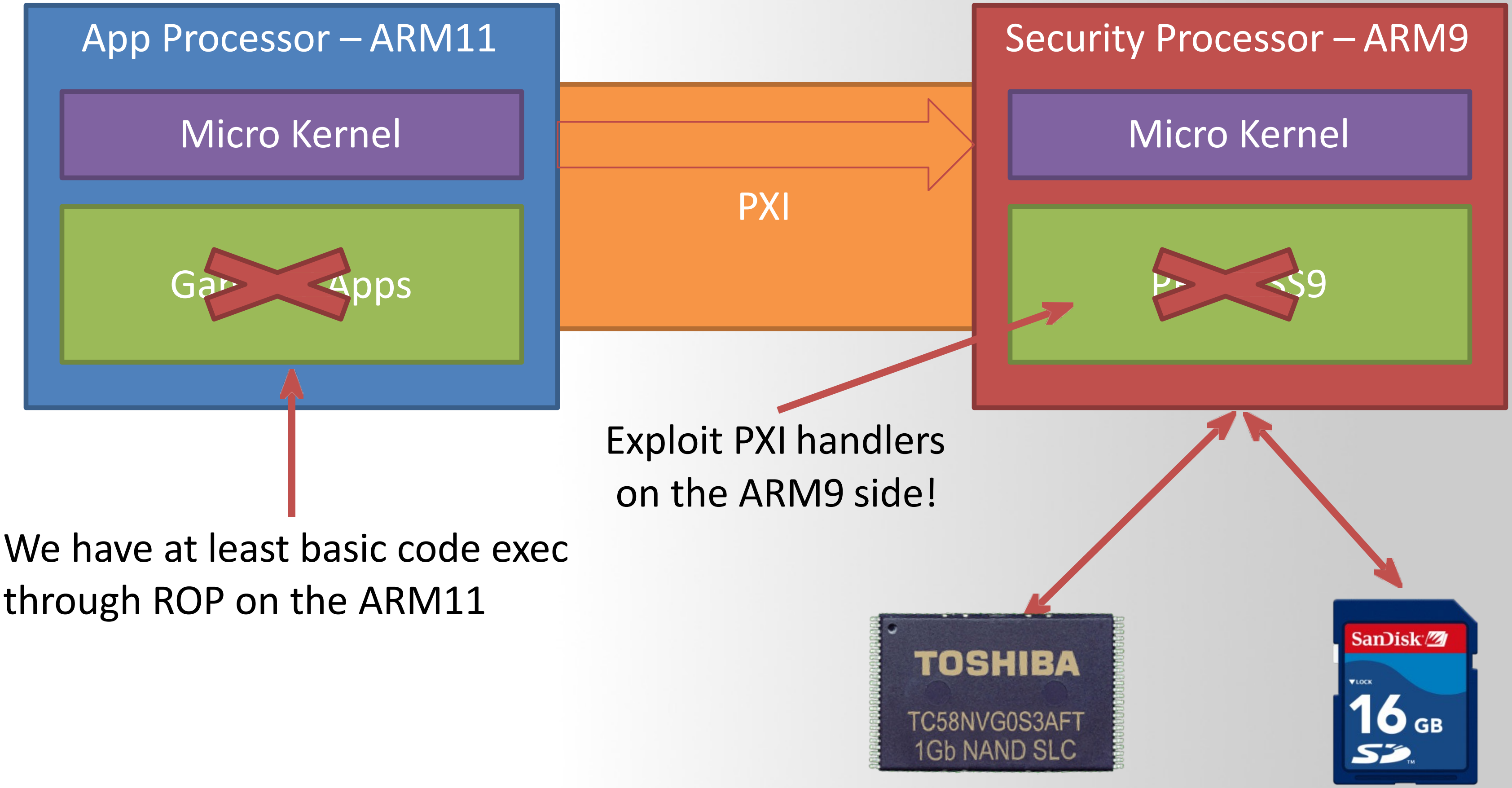
TAKING OVER THE ARM9

Malicious PXI Requests



We have at least basic code exec through ROP on the ARM11

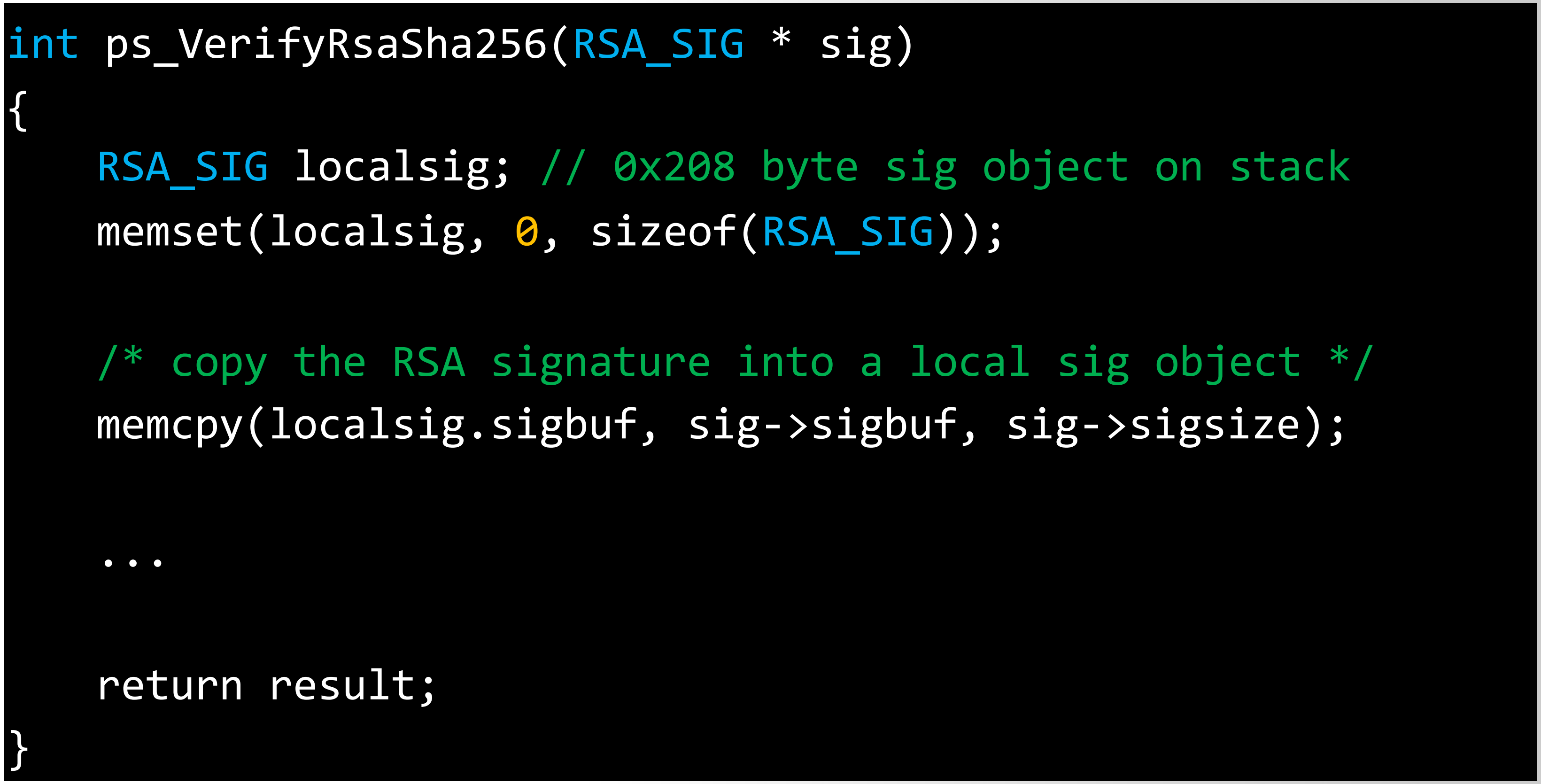
Malicious PXI Requests



Exploit PXI handlers on the ARM9 side!

We have at least basic code exec through ROP on the ARM11

Pseudocode of the ARM9 Bug



```
int ps_VerifyRsaSha256(RSA_SIG * sig)
{
   RSA_SIG localsig; // 0x208 byte sig object on stack
   memset(localsig, 0, sizeof(RSA_SIG));

   /* copy the RSA signature into a local sig object */
   memcpy(localsig.sigbuf, sig->sigbuf, sig->sigsize);


   ...


   return result;
}
```

Pseudocode of the ARM9 Bug

```
int ps_VerifyRsaSha256(RSA_SIG * sig)
{
    RSA_SIG localsig; // 0x208 byte sig object on stack
    memset(localsig, 0, sizeof(RSA_SIG));

    /* copy the RSA signature into a local sig object */
    memcpy(localsig.sigbuf, sig->sigbuf, sig->sigsize);

    ...

    return result;
}
```

Attacker Controlled Data

VerifyRsaSha256() – Jun. 2013

- Bug is basically a memcpy with user controlled data, and a user specified size

- No DEP or ASLR on the ARM9, simply overwrite return address and jump onto your buffer! (:

- With control of the ARM9 you can do anything
  - Load a custom firmware & soft reboot the system

GAME OVER

Owning the 3DS

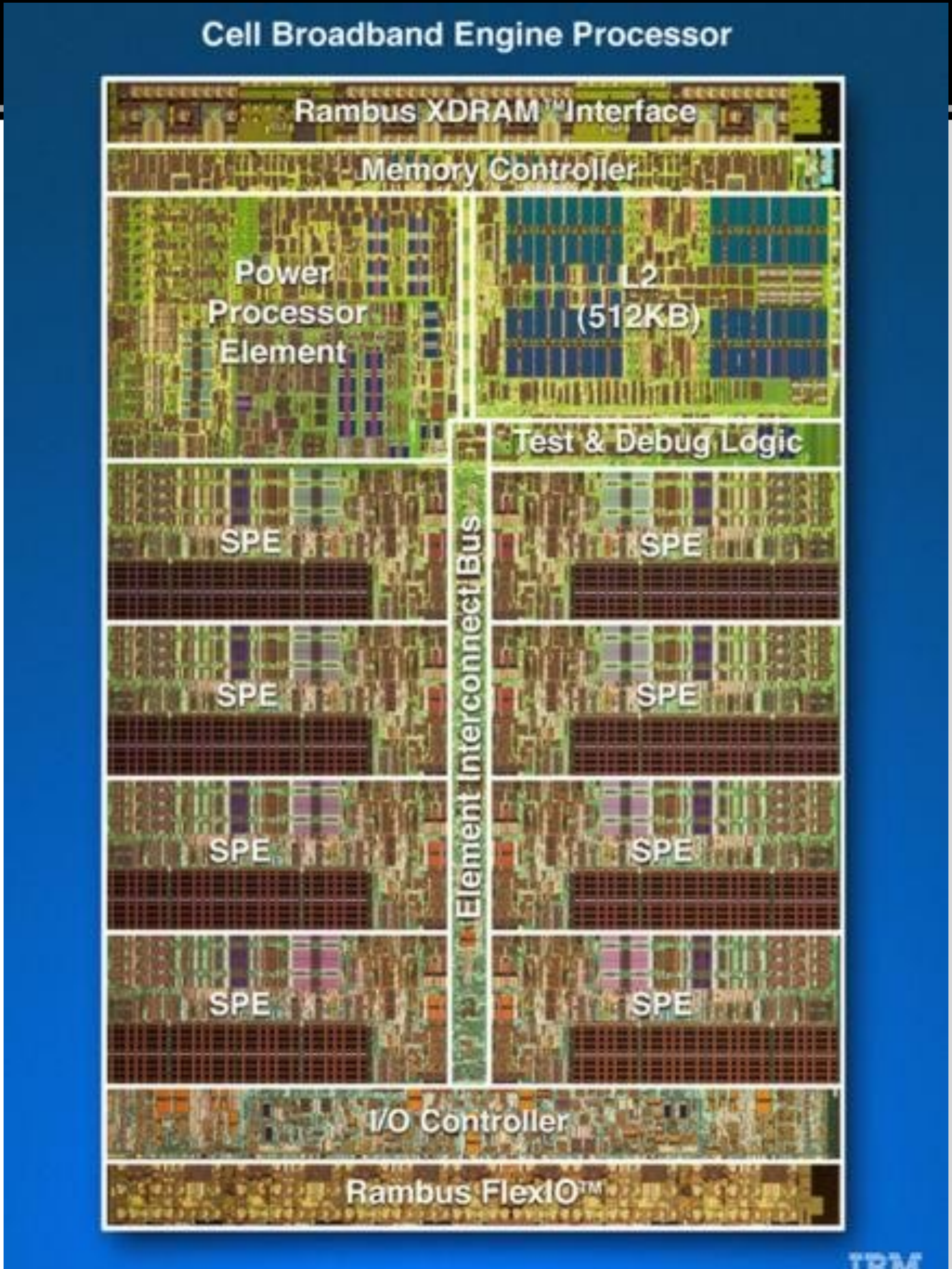
- Code exec on the ARM11 is easy
  - Tons of crappy vulnerable games everywhere, less exciting exploits exist to do this

- Owning the ARM9 is much harder
  - Limited attack surface with little user input

PlayStation 3 – Nov. 2006

PlayStation 3 – Nov. 2006

- Security Perspective
  - FreeBSD Based OS
  - Only runs signed code or executables
  - Rigorous chain of trust, secure bootstrapping
  - Cell Architecture
    - Isolates cores from each other, HV
    - Dedicated System / Security Cell
  - Encrypted runtime memory
  - Encrypted HDD
  - eFuses
  - NX/DEP
  - No ASLR


Cell Broadband Engine Processor
Rambus XDRAM™ Interface
Memory Controller
Power Processor Element
L2 (512KB)
Test & Debug Logic
SPE
SPE
Element Interconnect Bus
SPE
SPE
SPE
SPE
SPE
SPE
I/O Controller
Rambus FlexIO™
IBM

Chain of Trust

Name	Processor / Mode	updateable	revocable*	usage
bootldr	SPE	✗	✗	boot lv0
lv0	PPE HV	✓	✗	boot lv1
metldr	SPE	✗	✗	run *ldr
lv1ldr	SPE	✓	✗	decrypt lv1
lv1	PPE HV	✓	✗	hypervisor
isoldr	SPE	✓	✗	decrypt modules
sc_iso	SPE	✓	✓	
...				
lv2ldr	SPE	✓	✗	decrypt lv2
lv2	PPE SV	✓	✓	kernel
appldr	SPE	✓	✓	decrypt games
some game	PPE PS	✓	✓	:-)

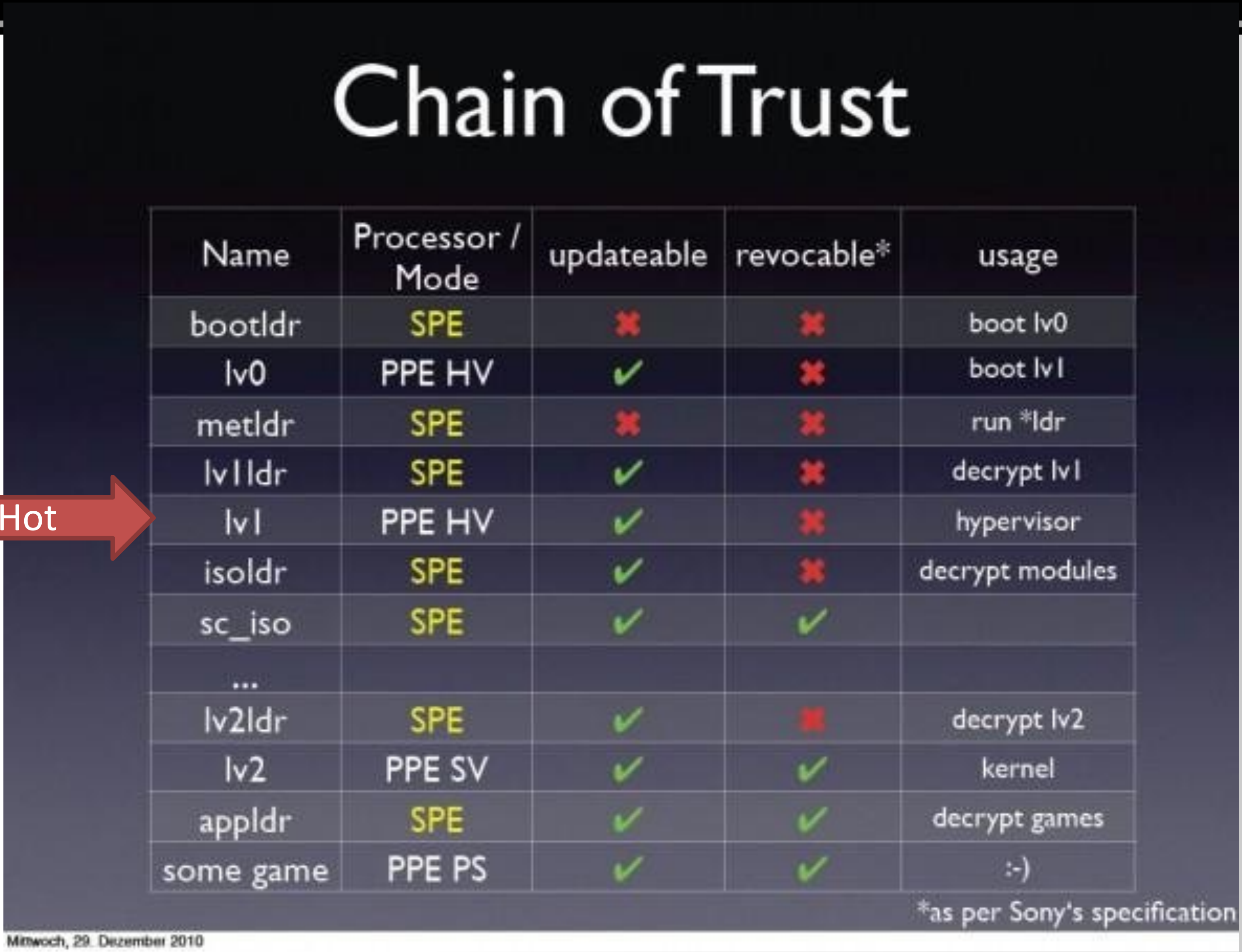
*as per Sony's specification

GeoHot hacks PS3 Hv – Jan. 2010

- Through OtherOS (Linux on PS3) and chip glitching, GeoHot owns the PS3 Hypervisor

- Glitching ‘creates’ a **use after free (UAF)** scenario in the Hypervisor that is then exploited to get code exec

- Dumps of PS3 HV & kernel make their way public



Chain of Trust

Name	Processor / Mode	updateable	revocable*	usage
bootldr	SPE	✗	✗	boot lv0
lv0	PPE HV	✓	✗	boot lv1
metldr	SPE	✗	✗	run *ldr
lv1ldr	SPE	✓	✗	decrypt lv1
lv1	PPE HV	✓	✗	hypervisor
isoldr	SPE	✓	✗	decrypt modules
sc_iso	SPE	✓	✓	
...				
lv2ldr	SPE	✓	✗	decrypt lv2
lv2	PPE SV	✓	✓	kernel
appldr	SPE	✓	✓	decrypt games
some game	PPE PS	✓	✓	:-)

*as per Sony's specification

Mittwoch, 29. Dezember 2010

GeoHot

More Privileged

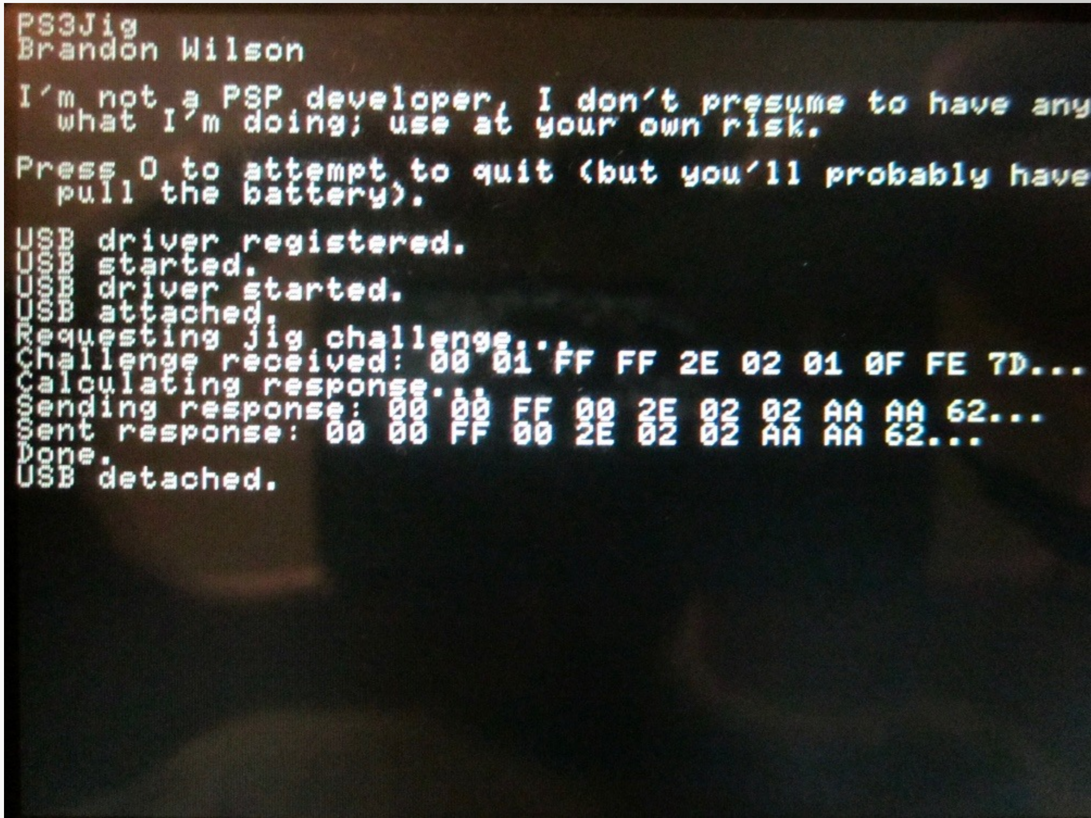
Sony Disables OtherOS – Mar. 2010

PS3 Jailbreak – Aug. 2010

- With the PS3 Kernel (LV2) dumped, heap overflow found in USB handling during startup while the system searches for a service jig

- The main bug is an overflow in long device descriptors that leads to memory corruption on the heap

- Results in control of the LV2



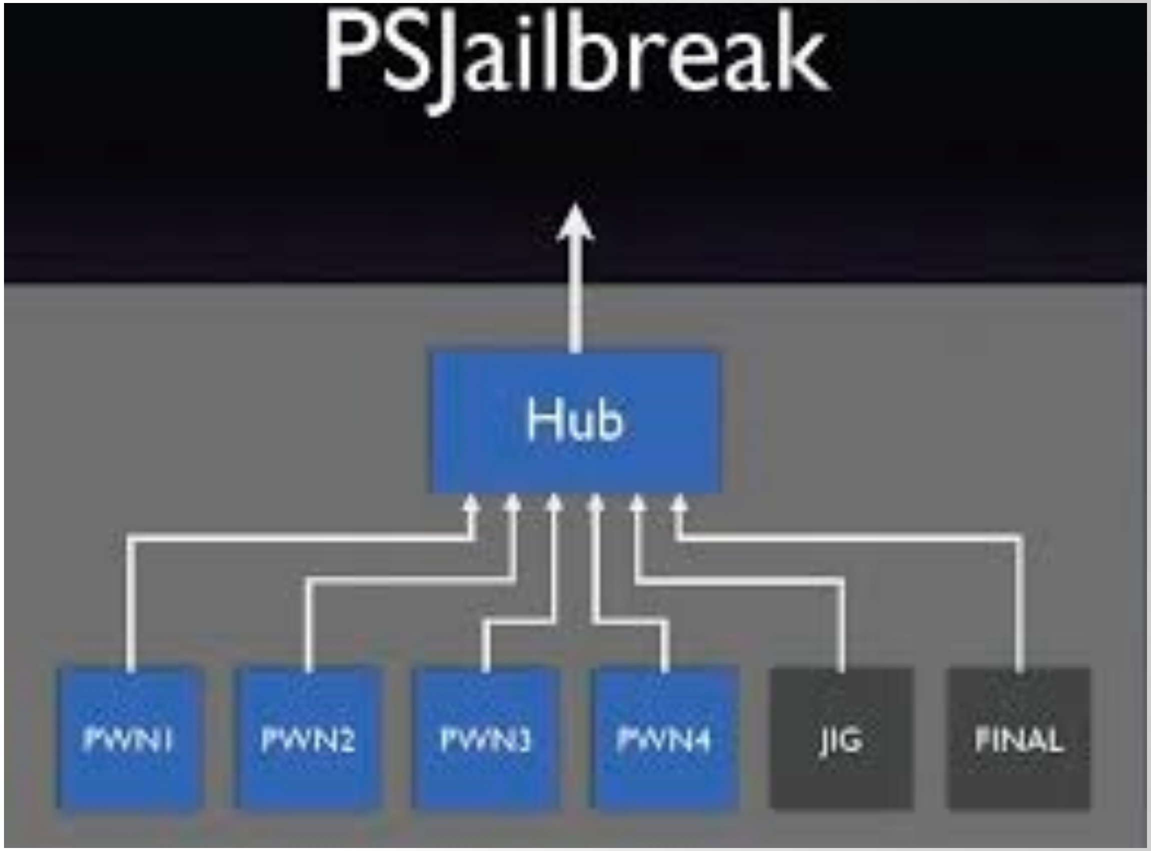
```
PS3Jig
Brandon Wilson

I'm not a PSP developer, I don't presume to have any
  what I'm doing; use at your own risk.

Press O to attempt to quit (but you'll probably have
  pull the battery).

USB driver registered.
USB started.
USB driver started.
USB attached.
Requesting jig challenge...
Challenge received: 00 01 FF FF 2E 02 01 0F FE 7D...
Calculating response...
Sending response: 00 00 FF 00 2E 02 02 AA AA 62...
Sent response: 00 00 FF 00 2E 02 02 AA AA 62...
Done.
USB detached.
```

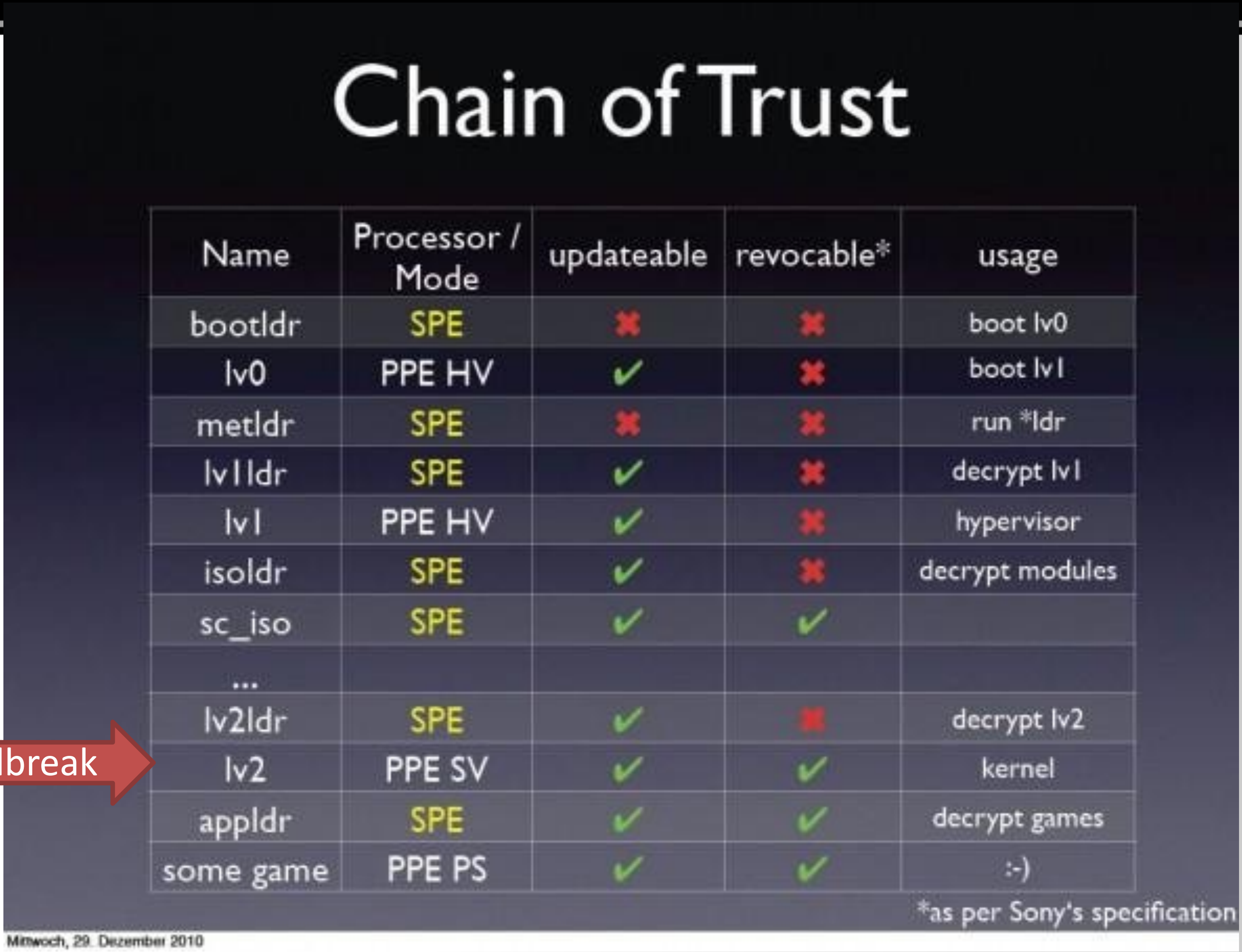
PS3 Jailbreak – Aug. 2010

PS3 Jailbreak – Aug. 2010

- Heap overflow setup and triggered through a USB hub (oops) and six USB's

- It's a bit like musical chairs, plugging and unplugging a number of USB's to malloc/free stuff – everyone just emulates this process with a single USB





Chain of Trust

Name	Processor / Mode	updateable	revocable*	usage
bootldr	SPE	✗	✗	boot lv0
lv0	PPE HV	✓	✗	boot lv1
metldr	SPE	✗	✗	run *ldr
lv1ldr	SPE	✓	✗	decrypt lv1
lv1	PPE HV	✓	✗	hypervisor
isoldr	SPE	✓	✗	decrypt modules
sc_iso	SPE	✓	✓	
...				
lv2ldr	SPE	✓	✗	decrypt lv2
lv2	PPE SV	✓	✓	kernel
appldr	SPE	✓	✓	decrypt games
some game	PPE PS	✓	✓	:-)

*as per Sony's specification

Mittwoch, 29. Dezember 2010

PS3 Jailbreak

More Privileged

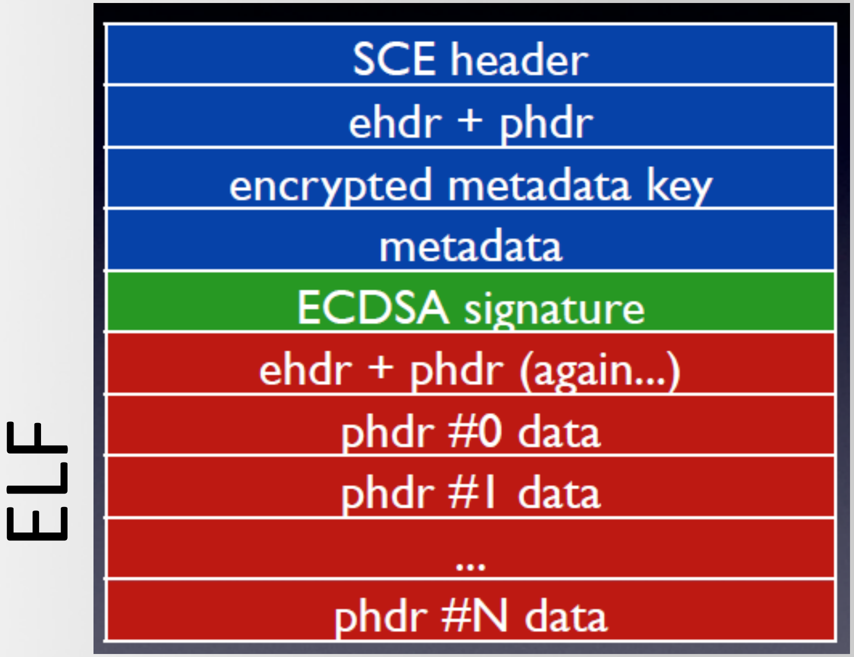
PS3 ECDSA KEY EXTRACTION

Largest console break of this generation stems from crypto flaw

PS3 ECDSA Key Extraction – Jan. 2011

- Executables running on the PS3 are modified ELF's known as SELF's

- Signed by Sony's ECDSA Key, encrypted by the associated Lv(0,1,2) keys
    - Elliptic Curve Digital Signature Algorithm



SCE header

ehdr + phdr

encrypted metadata key

metadata

ECDSA signature

ehdr + phdr (again...)

phdr #0 data

phdr #1 data

...

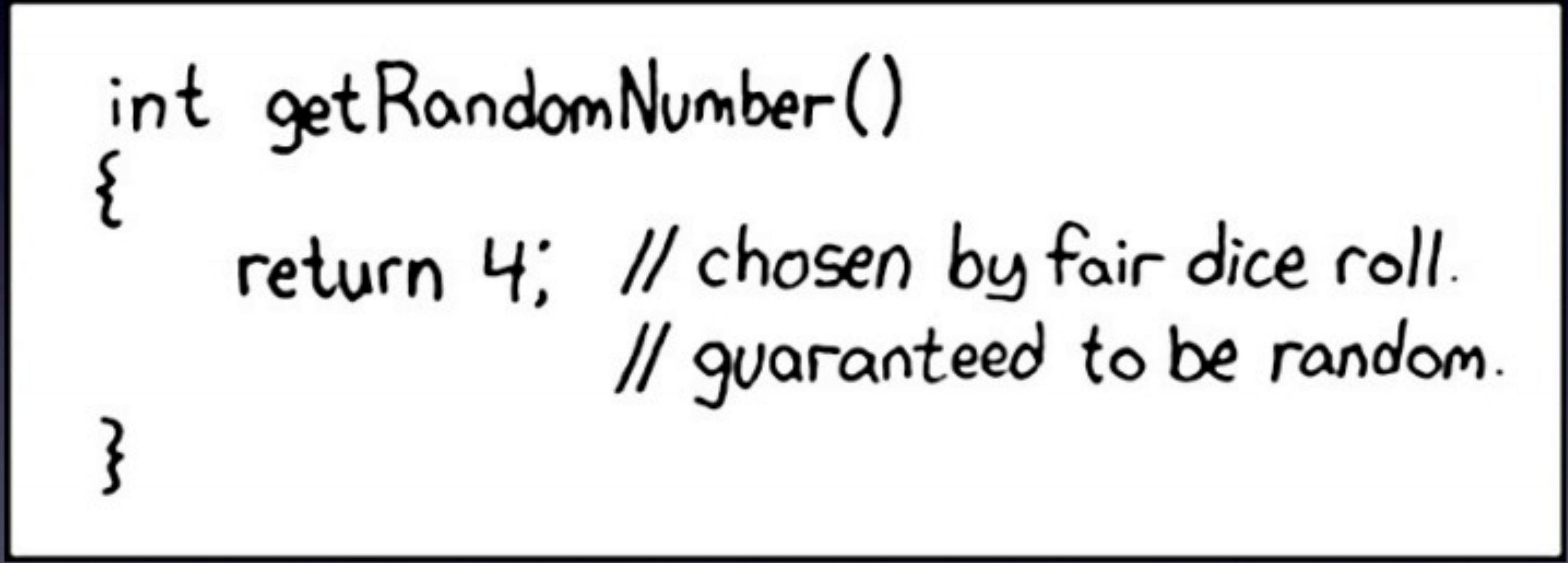
phdr #N data

PS3 ECDSA Key Extraction – Jan. 2011

- With control of the LV2, you can make crypto requests to the security SPE and use it as a black box

- A crypto implementation flaw is uncovered by fail0verflow regarding Sony’s ECDSA signatures

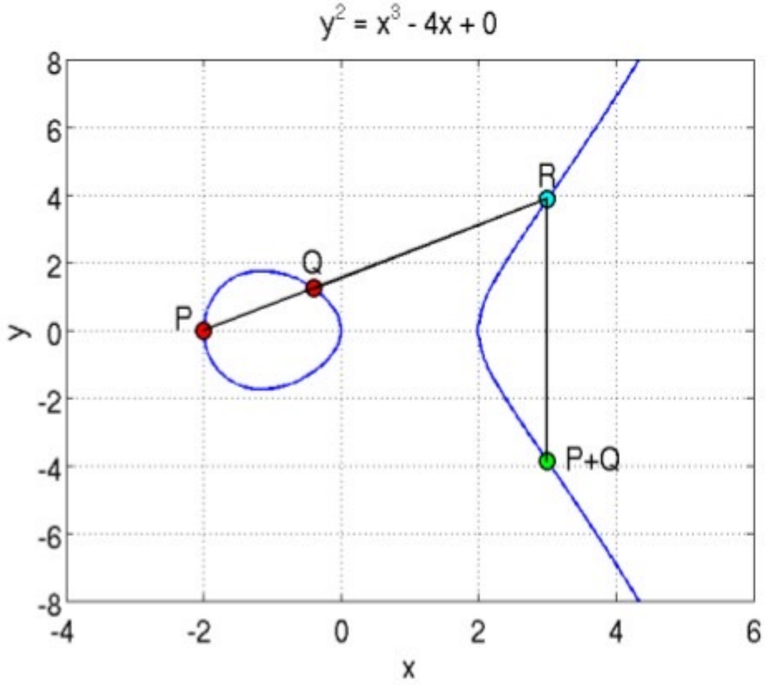
Sony's ECDSA code

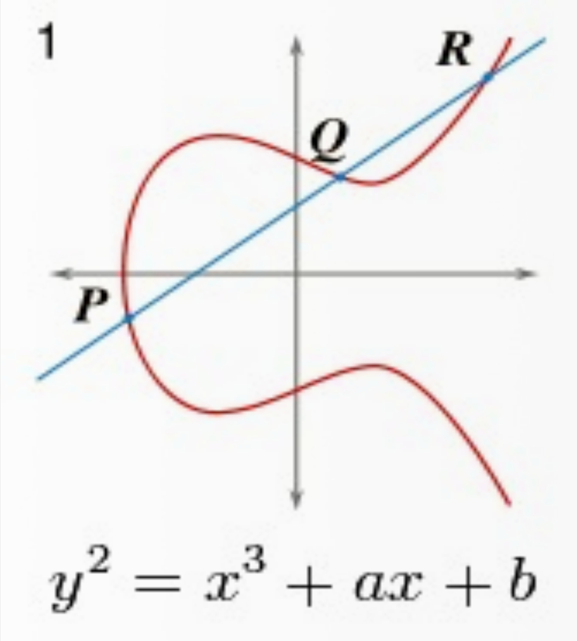


```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

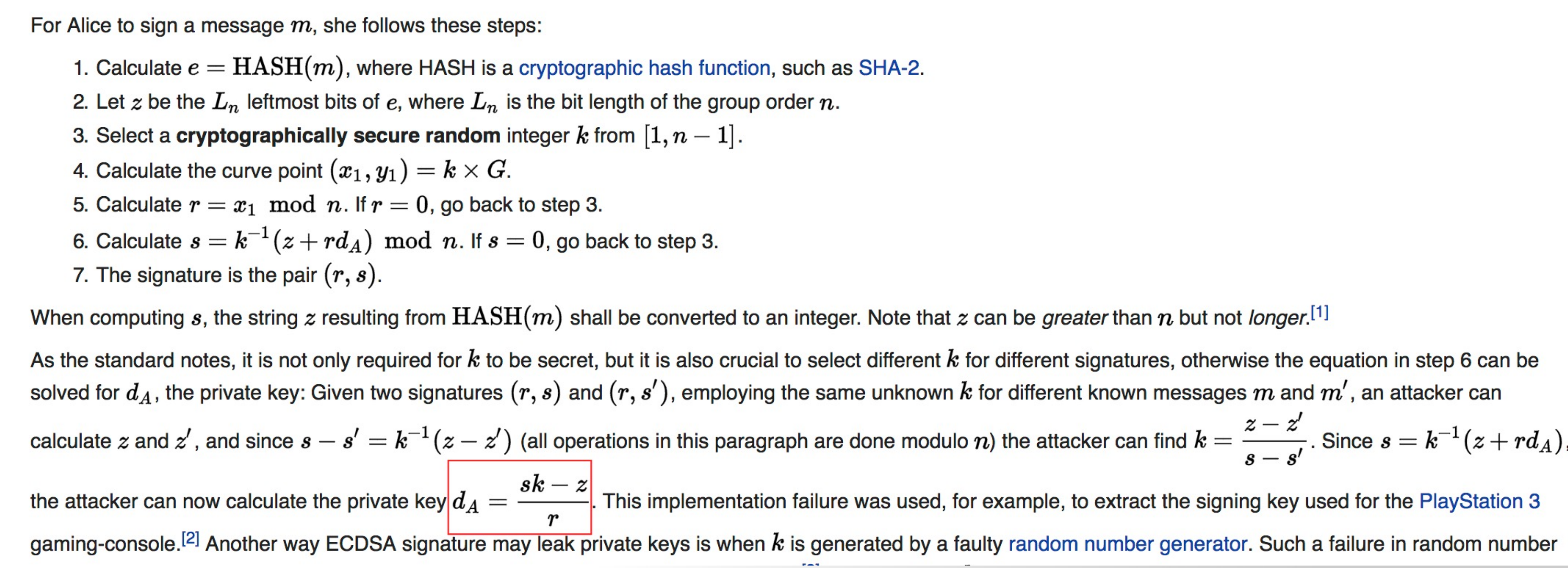
fail0verflow

Elliptic Curve Cryptography





Const Instead of Nonce



For Alice to sign a message m, she follows these steps:

1. Calculate $e = \mathrm{HASH}(m)$, where HASH is a cryptographic hash function, such as SHA-2.
2. Let z be the L_n leftmost bits of e, where L_n is the bit length of the group order n.
3. Select a **cryptographically secure random** integer k from $[1, n-1]$.
4. Calculate the curve point $(x_1, y_1) = k \times G$.
5. Calculate $r = x_1 \mod n$. If $r = 0$, go back to step 3.
6. Calculate $s = k^{-1}(z + rd_A) \mod n$. If $s = 0$, go back to step 3.
7. The signature is the pair (r, s).

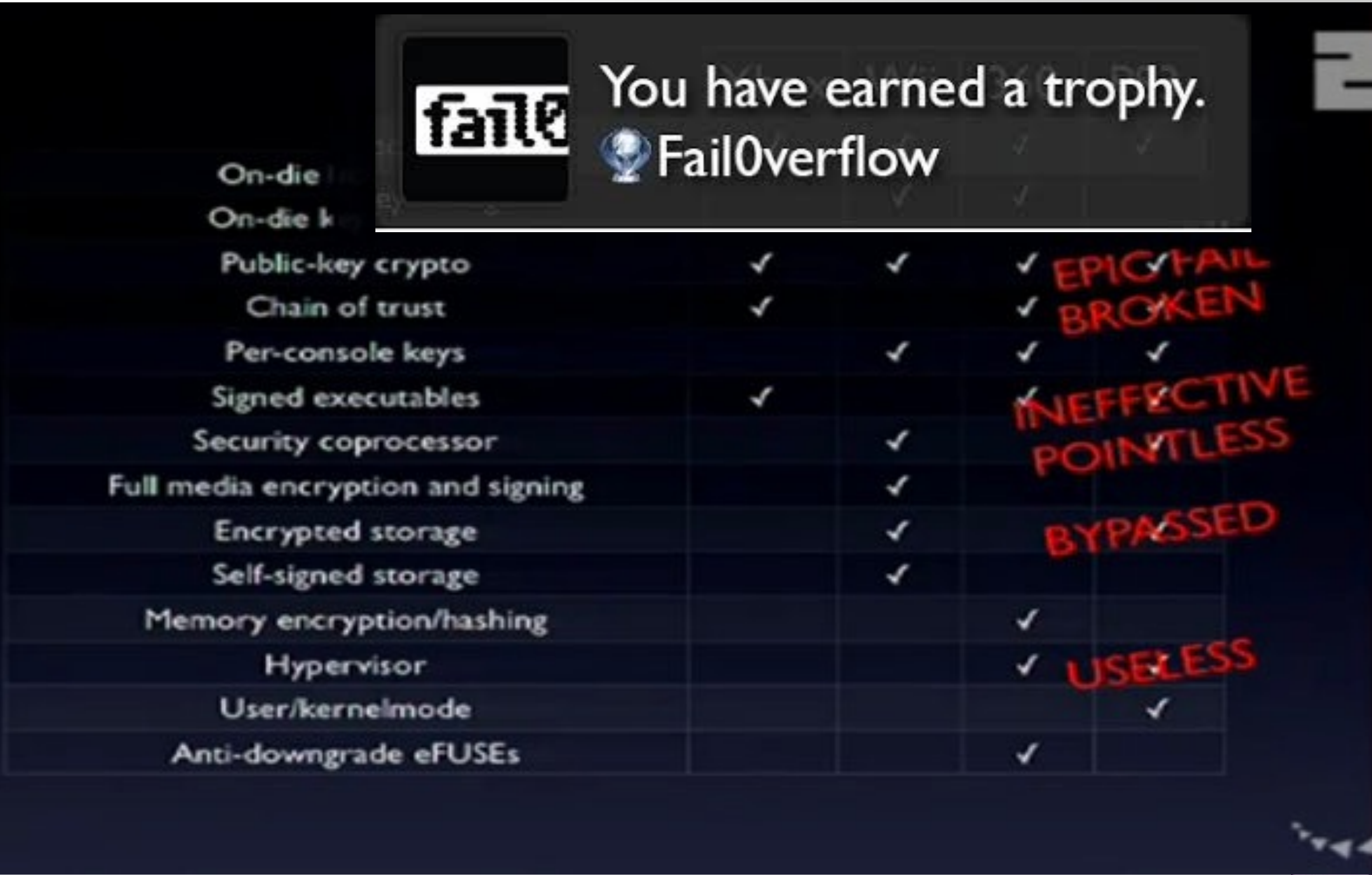
When computing s, the string z resulting from $\mathrm{HASH}(m)$ shall be converted to an integer. Note that z can be *greater* than n but not *longer*.[1]

As the standard notes, it is not only required for k to be secret, but it is also crucial to select different k for different signatures, otherwise the equation in step 6 can be solved for d_A, the private key: Given two signatures (r, s) and (r, s'), employing the same unknown k for different known messages m and m', an attacker can calculate z and z', and since $s - s' = k^{-1}(z - z')$ (all operations in this paragraph are done modulo n) the attacker can find $k = \frac{z - z'}{s - s'}$. Since $s = k^{-1}(z + rd_A)$, the attacker can now calculate the private key $\boxed{d_A = \frac{sk - z}{r}}$. This implementation failure was used, for example, to extract the signing key used for the PlayStation 3 gaming-console.[2] Another way ECDSA signature may leak private keys is when k is generated by a faulty random number generator. Such a failure in random number

Effects of Missteps

- With only TWO signatures from the Crypto SPE, you can compute Sony’s Private ECDSA Key

- With the ECDSA Key, the floodgates are opened
  - You can sign anything as Sony
  - This key is embedded in hardware



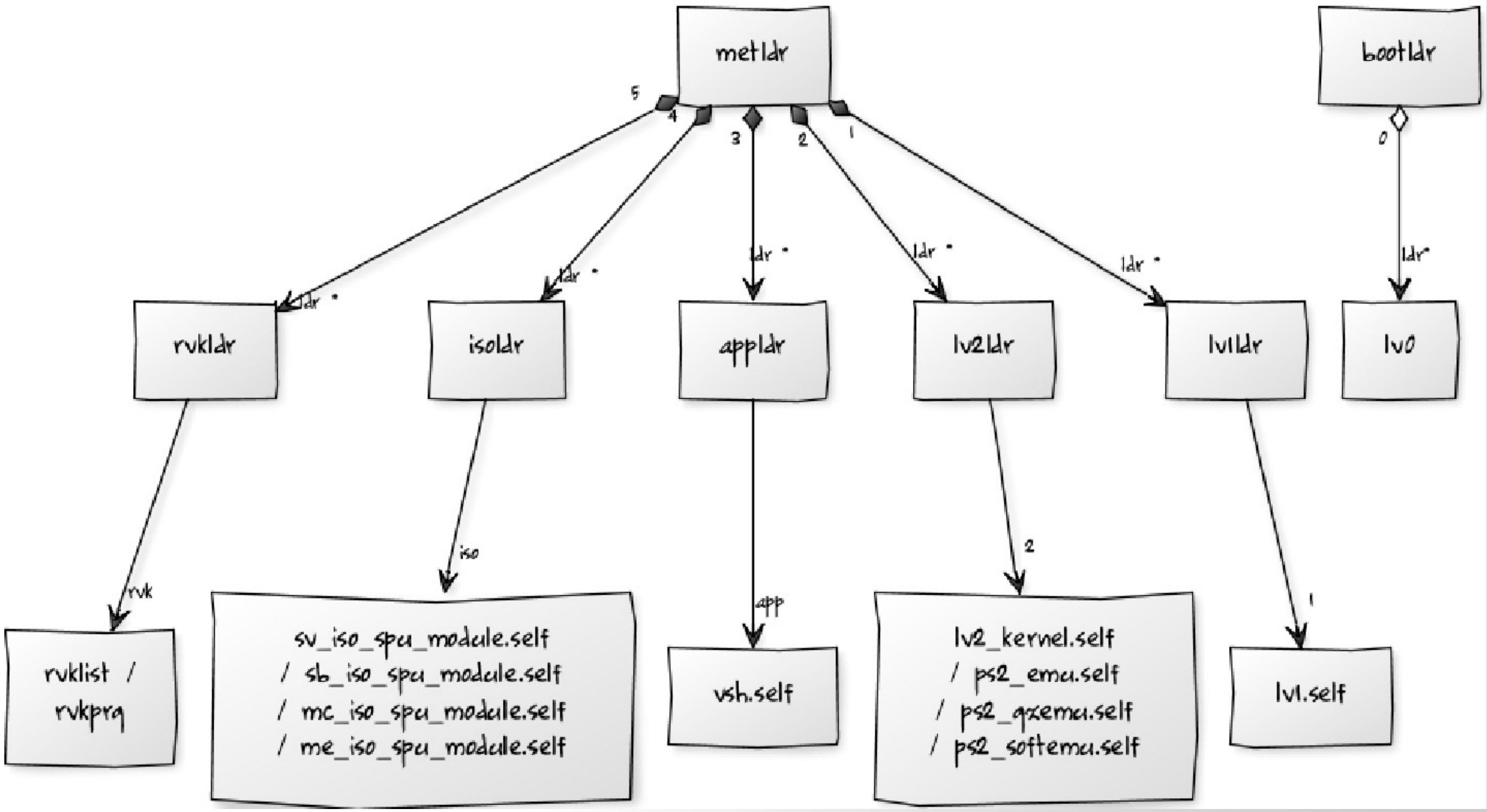
You have earned a trophy.

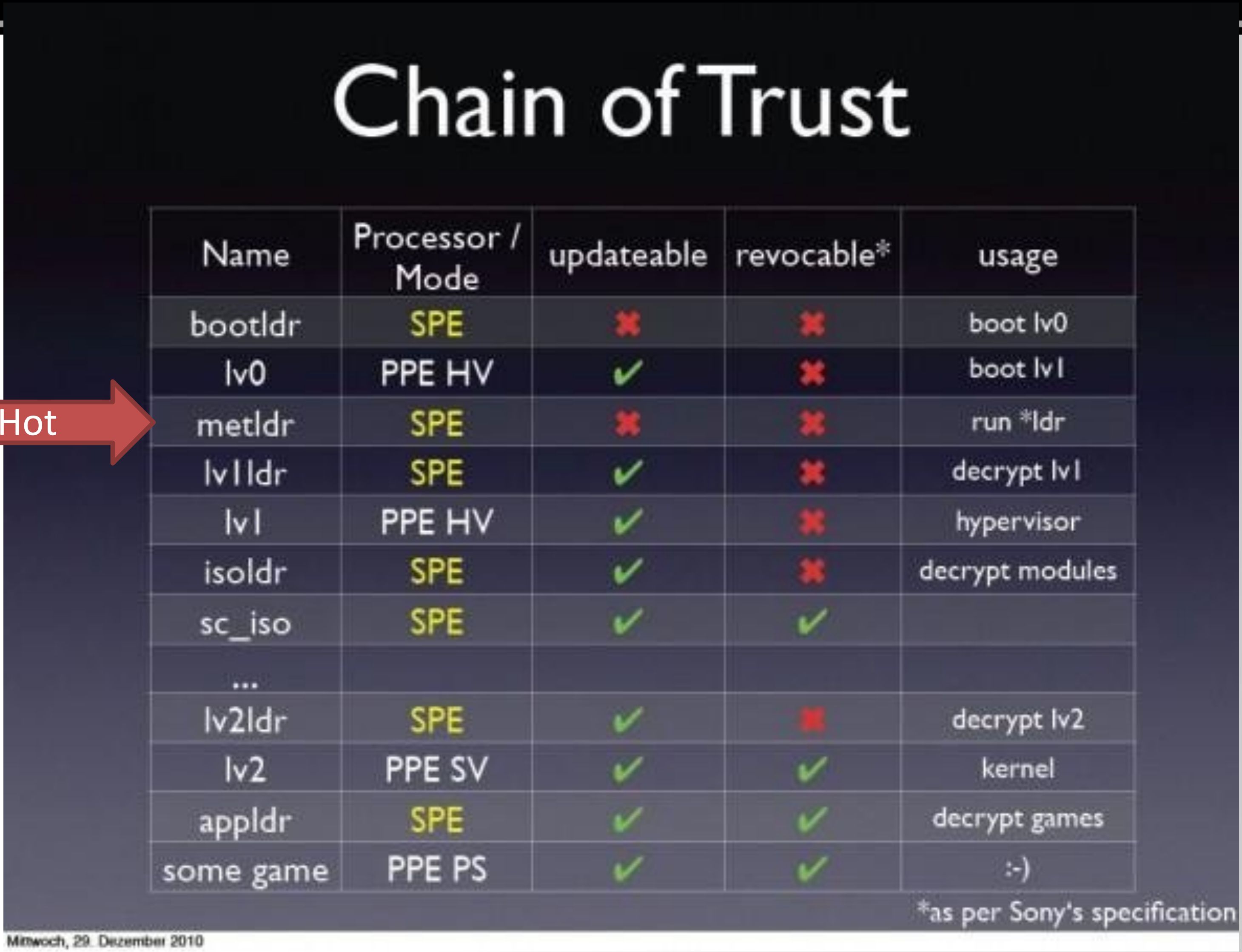
Fail0verflow

	Xbox	Wii	360	PS3
On-die [illegible]	✓	✓	✓	✓
On-die k[illegible]		✓	✓	
Public-key crypto	✓	✓	✓	✓
Chain of trust	✓		✓	✓
Per-console keys		✓	✓	✓
Signed executables	✓		✓	✓
Security coprocessor		✓		✓
Full media encryption and signing		✓		
Encrypted storage		✓		✓
Self-signed storage		✓		
Memory encryption/hashing			✓	
Hypervisor			✓	✓
User/kernelmode				✓
Anti-downgrade eFUSEs			✓	

metldr Owned

- Geohot releases metldr decryption keys


metldr
bootldr
5
4
3
2
1
0
ldr *
ldr *
ldr *
ldr *
ldr *
ldr*
rvkldr
isoldr
appldr
lv2ldr
lv1ldr
lv0
rvk
iso
app
2
1
rvklist / rvkprg
sv_iso_spu_module.self / sb_iso_spu_module.self / mc_iso_spu_module.self / me_iso_spu_module.self
vsh.self
lv2_kernel.self / ps2_emu.self / ps2_gxemu.self / ps2_softemu.self
lv1.self



Chain of Trust

Name	Processor / Mode	updateable	revocable*	usage
bootldr	SPE	✗	✗	boot lv0
lv0	PPE HV	✓	✗	boot lv1
metldr	SPE	✗	✗	run *ldr
lv1ldr	SPE	✓	✗	decrypt lv1
lv1	PPE HV	✓	✗	hypervisor
isoldr	SPE	✓	✗	decrypt modules
sc_iso	SPE	✓	✓	
...				
lv2ldr	SPE	✓	✗	decrypt lv2
lv2	PPE SV	✓	✓	kernel
appldr	SPE	✓	✓	decrypt games
some game	PPE PS	✓	✓	:-)

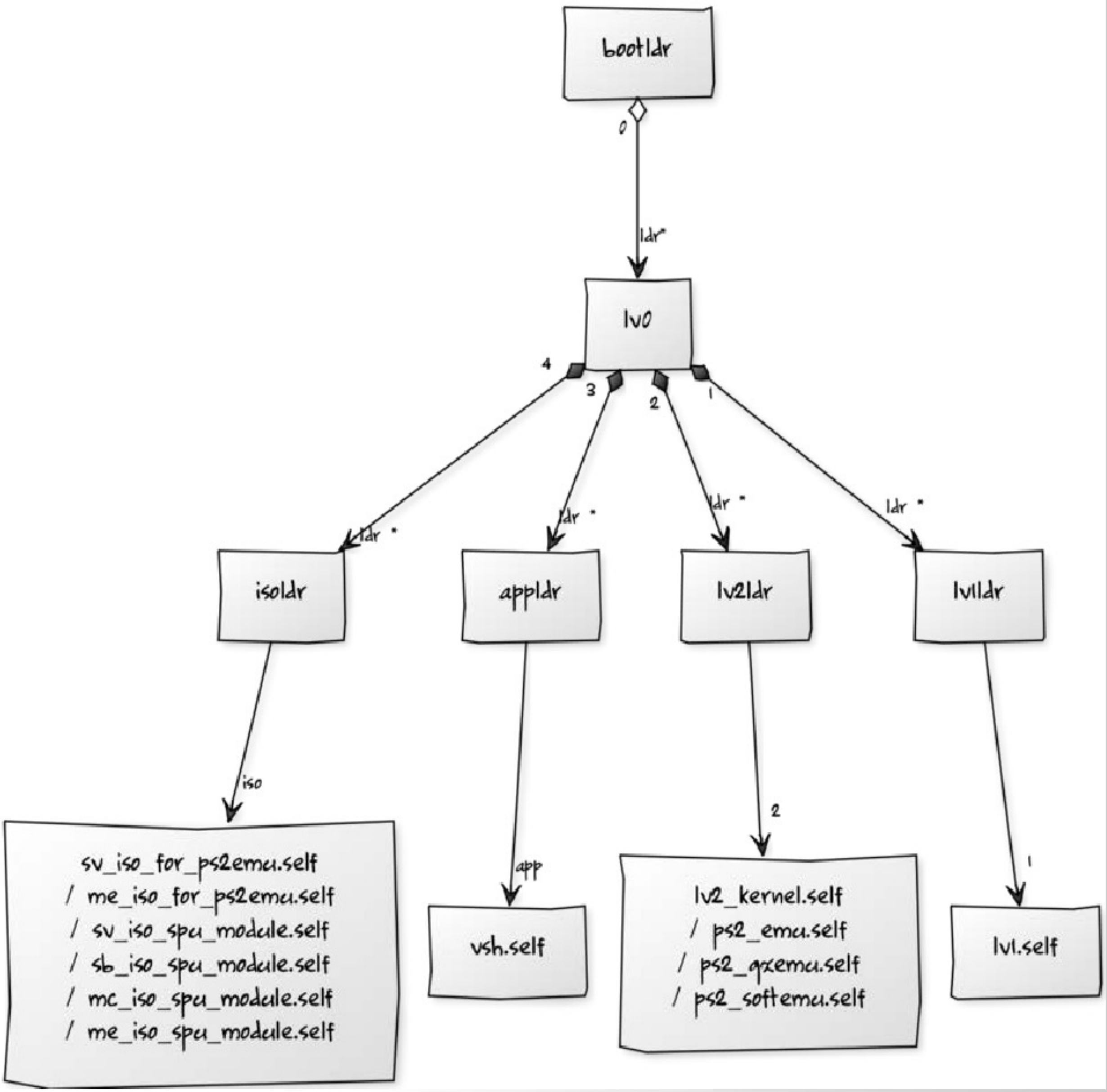
*as per Sony's specification

Mittwoch, 29. Dezember 2010

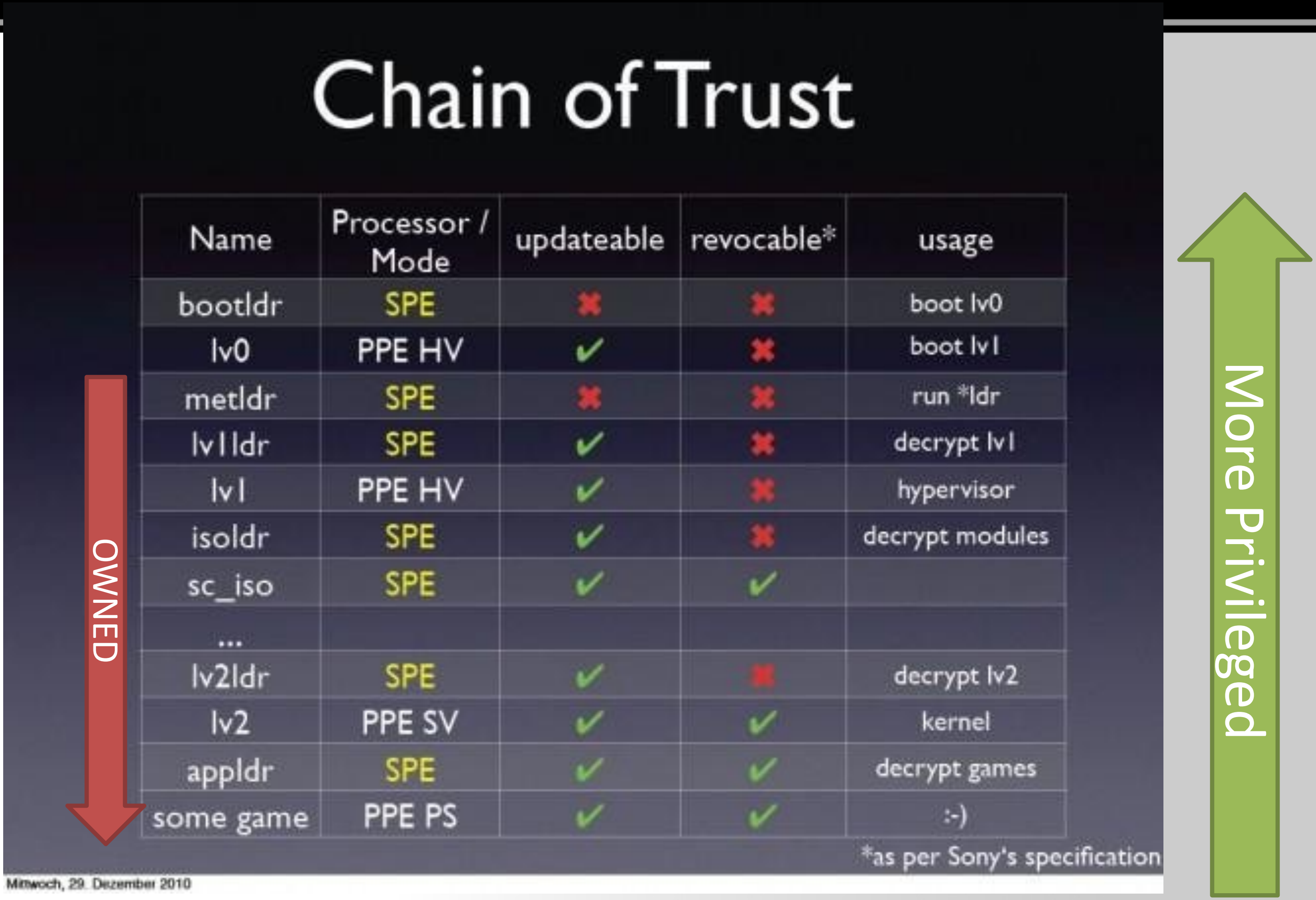
GeoHot

More Privileged

Sony Nukes metldr



Sony Sues Geohot – Jan. 2011



Chain of Trust

Name	Processor / Mode	updateable	revocable*	usage
bootldr	SPE	✗	✗	boot lv0
lv0	PPE HV	✓	✗	boot lv1
metldr	SPE	✗	✗	run *ldr
lv1ldr	SPE	✓	✗	decrypt lv1
lv1	PPE HV	✓	✗	hypervisor
isoldr	SPE	✓	✗	decrypt modules
sc_iso	SPE	✓	✓	
...				
lv2ldr	SPE	✓	✗	decrypt lv2
lv2	PPE SV	✓	✓	kernel
appldr	SPE	✓	✓	decrypt games
some game	PPE PS	✓	✓	:-)

*as per Sony's specification

Mittwoch, 29. Dezember 2010

Owning the lv0

- metldr is gone, so you need to own the lv0

- lv0 blobs can be signed, but they're encrypted and we don't have the keys to decrypt them

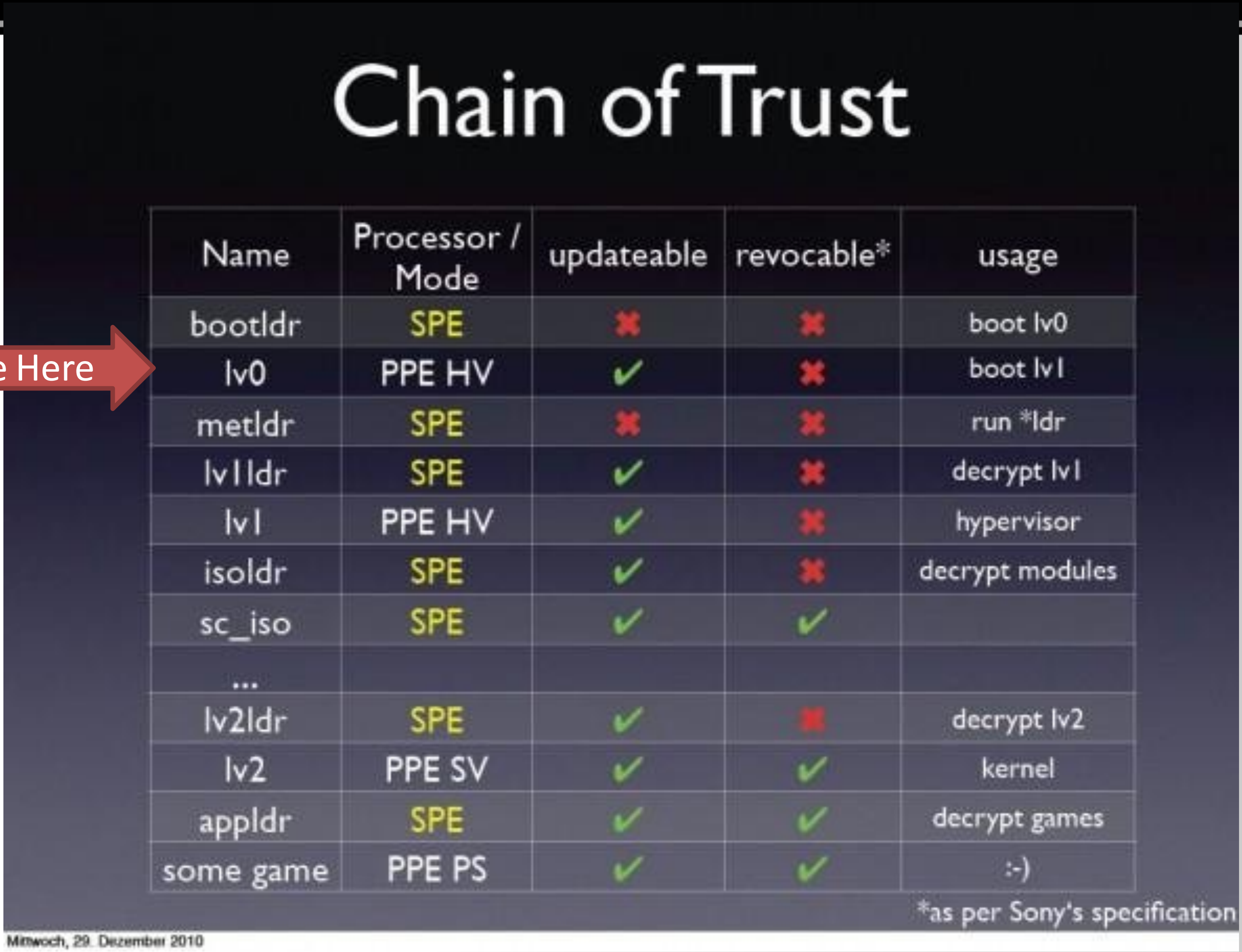
- What do you do?????

Owning the lv0

- metldr is gone, so you need to own the lv0
- lv0 blobs can be signed, but they’re encrypted and we don’t have the keys to decrypt them
- What do you do?????
  - Sign random data blobs, and hope the instruction at the entry point ‘decrypt’ to a jmp/call to code that you control

lv0 Owned – Oct. 2012

- Trying randomly signed blobs eventually works and execution is achieved at level of lv0



Chain of Trust

Name	Processor / Mode	updateable	revocable*	usage
bootldr	SPE	✗	✗	boot lv0
lv0	PPE HV	✓	✗	boot lv1
metldr	SPE	✗	✗	run *ldr
lv1ldr	SPE	✓	✗	decrypt lv1
lv1	PPE HV	✓	✗	hypervisor
isoldr	SPE	✓	✗	decrypt modules
sc_iso	SPE	✓	✓	
...				
lv2ldr	SPE	✓	✗	decrypt lv2
lv2	PPE SV	✓	✓	kernel
appldr	SPE	✓	✓	decrypt games
some game	PPE PS	✓	✓	:-)

*as per Sony's specification

Mittwoch, 29. Dezember 2010

You are Here

More Privileged

lv0 Owned – Oct. 2012

- Decryption keys are retrieved as lv0. Now you can create meaningful lv0 blobs, encrypt them, and sign them

- bootldr also exploited and dumped for fun
  - Not updateable anyway, so it doesn't matter much

GAME OVER
INSERT COIN

PS3 Aftermath

- Sony drops lawsuit against Geohot
  - Must never hack Sony products again

- No more updateable seeds of trust exist on the PS3 that Sony can utilize
  - PS3 totally broken

XBOX
Y
X
B
A

NINTENDO SWITCH
NINTENDO SWITCH
X
Y
A
B

PS4™
SONY
PS4

Introduction

So far, we have been exploiting binaries running in userspace.

Userspace is an *abstraction* that runs "on top" of the kernel.

1. Filesystem I/O
2. Privilege Levels (Per User/Per Group)
3. Syscalls
4. Processes
5. And so much more

Introduction

So far, we have been exploiting binaries running in userspace.

Userspace is an *abstraction* that runs "on top" of the kernel.

1. Filesystem I/O
2. Privilege Levels (Per User/Per Group)
3. Syscalls
4. Processes
5. And so much more

These are all "services" provided by the Kernel

Introduction

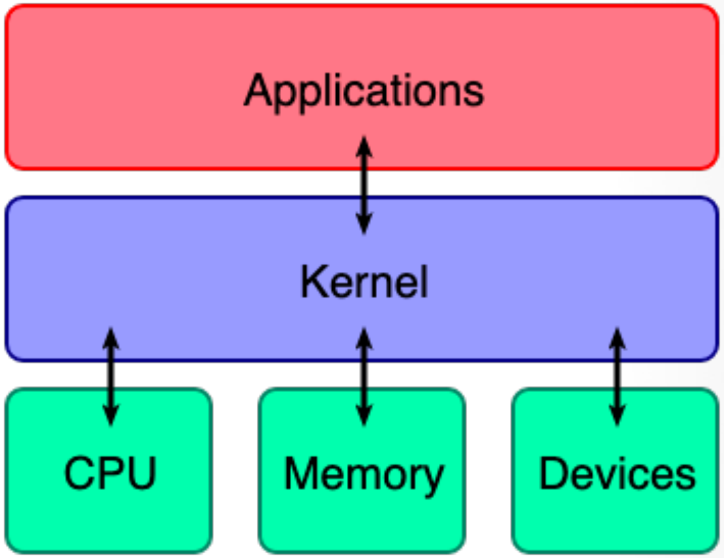
What’s a Kernel?

Low Level code with two major responsibilities

1. Interact with and control hardware components
2. Provide an Environment in which Applications can run

The Kernel is the core of the operating system

Introduction



The kernel is also a **program** that:

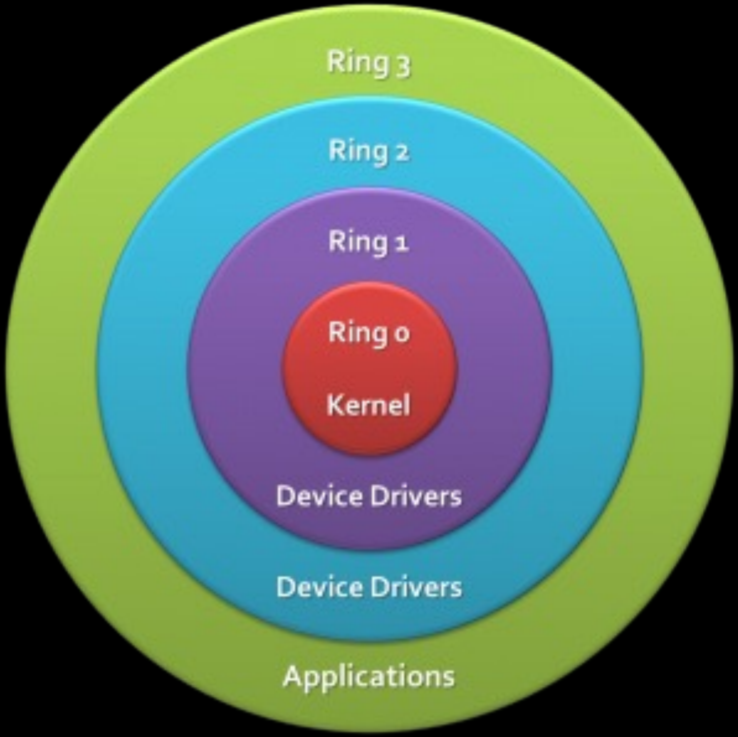
- Manages the data I/O requirements issued by the software
- Escaping these requirements into instructions
- Handing them over to the CPU

Ring Model

Hardware Enforced Model

0: Privileged, Kernelspace

3: Restricted, Userspace



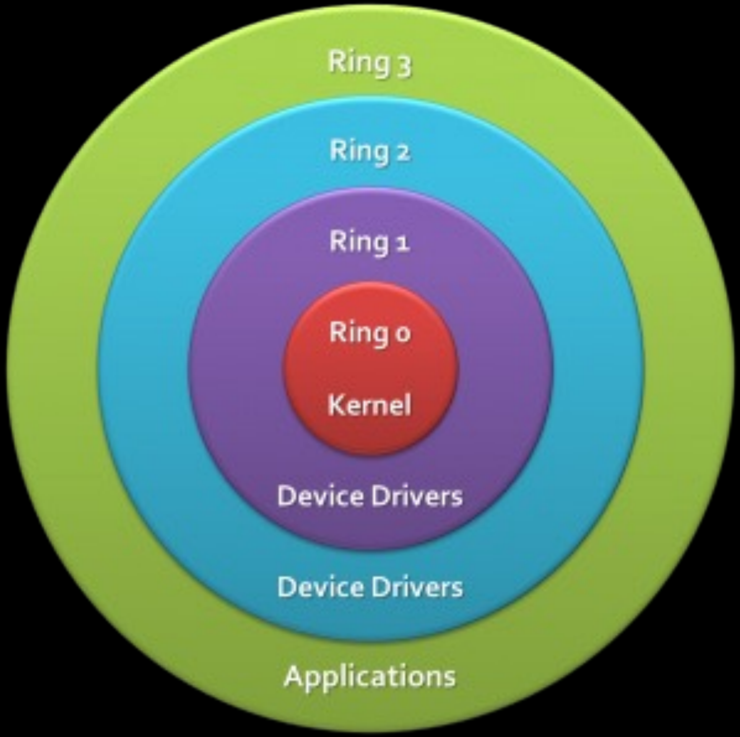
Ring Model

Hardware Enforced Model

0: Privileged, Kernelspace

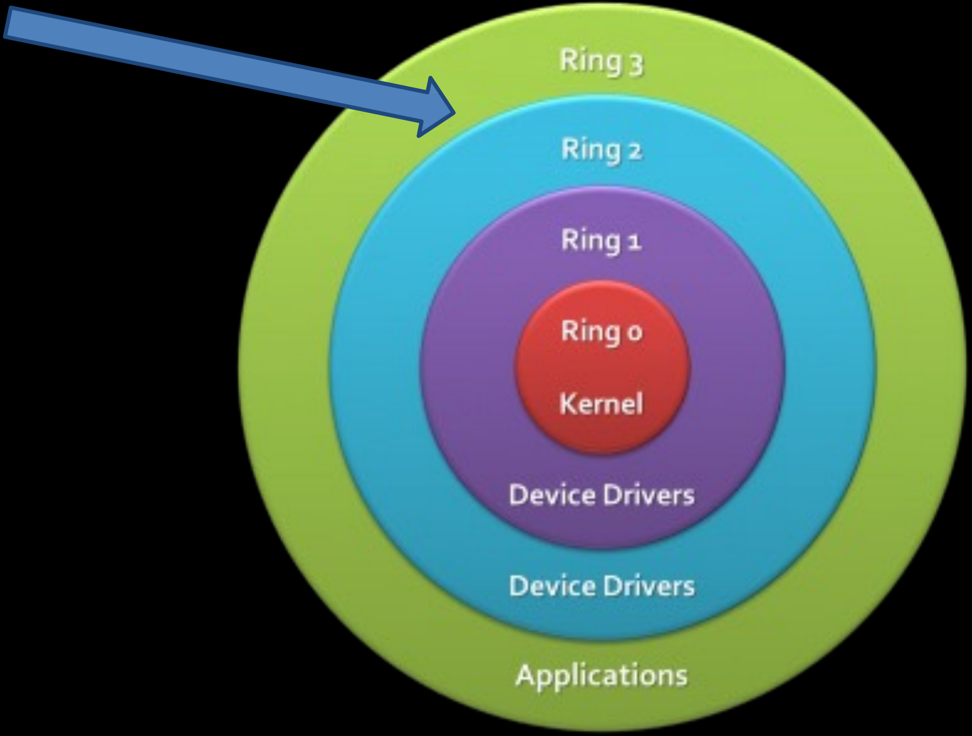
3: Restricted, Userspace

Ring 1 and Ring 2 are not utilized by most popular/modern Operating Systems (Linux / Windows / OSX)



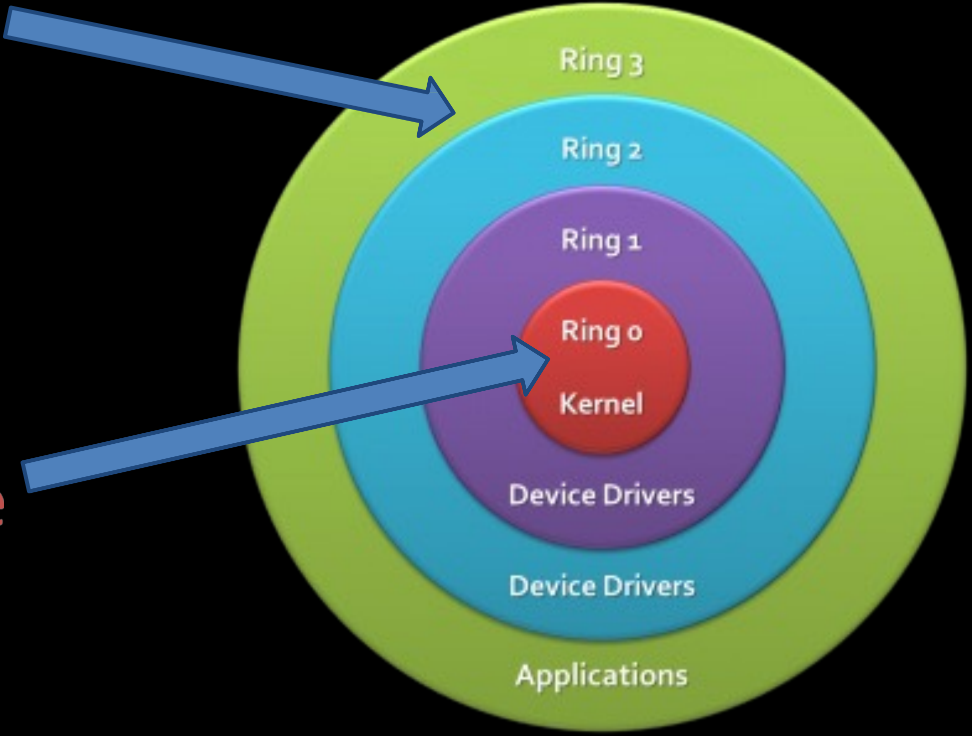
Ring Model

We've Been Here



Ring Model

We’ve Been Here



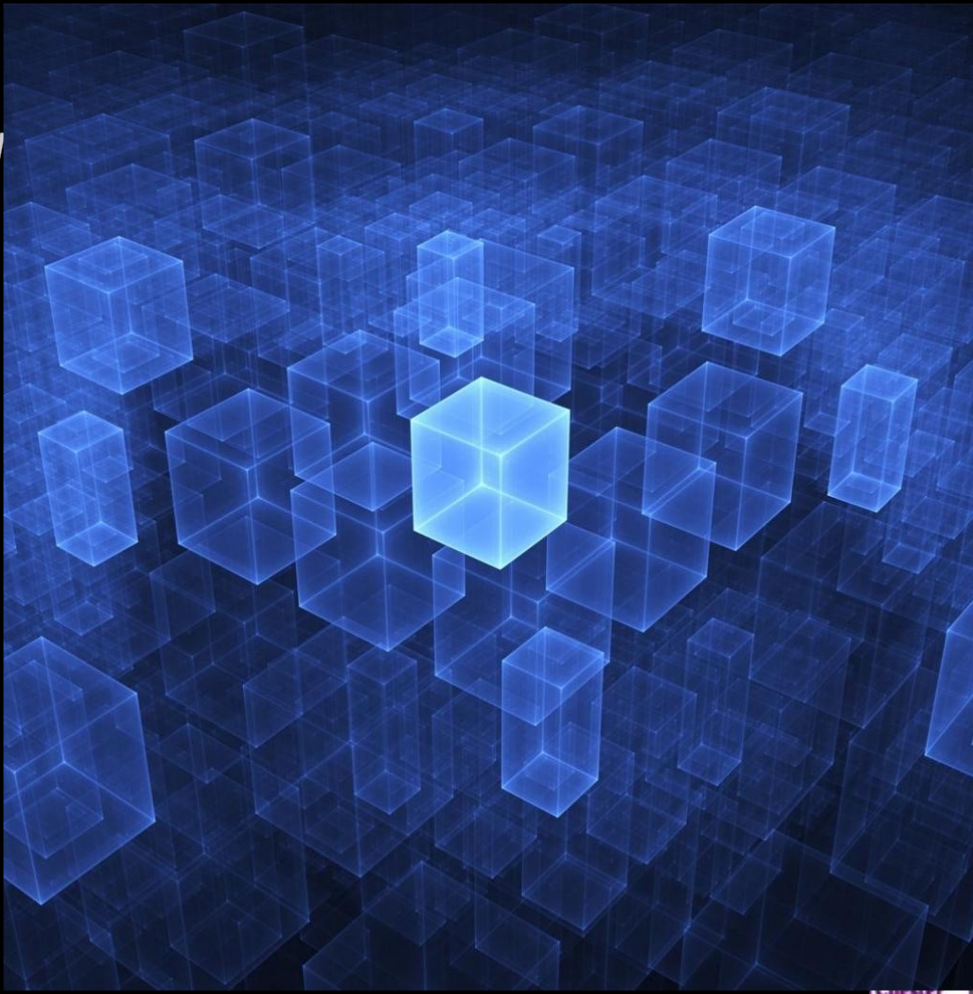
We’re Going Here

“Matrix”

“The Matrix is the world that has been pulled over your eyes to blind you from the truth.” - Morpheus

The kernel provides the “matrix” your programs run in

Break out of the Matrix, and you pwn the entire system



Kernel Pwning

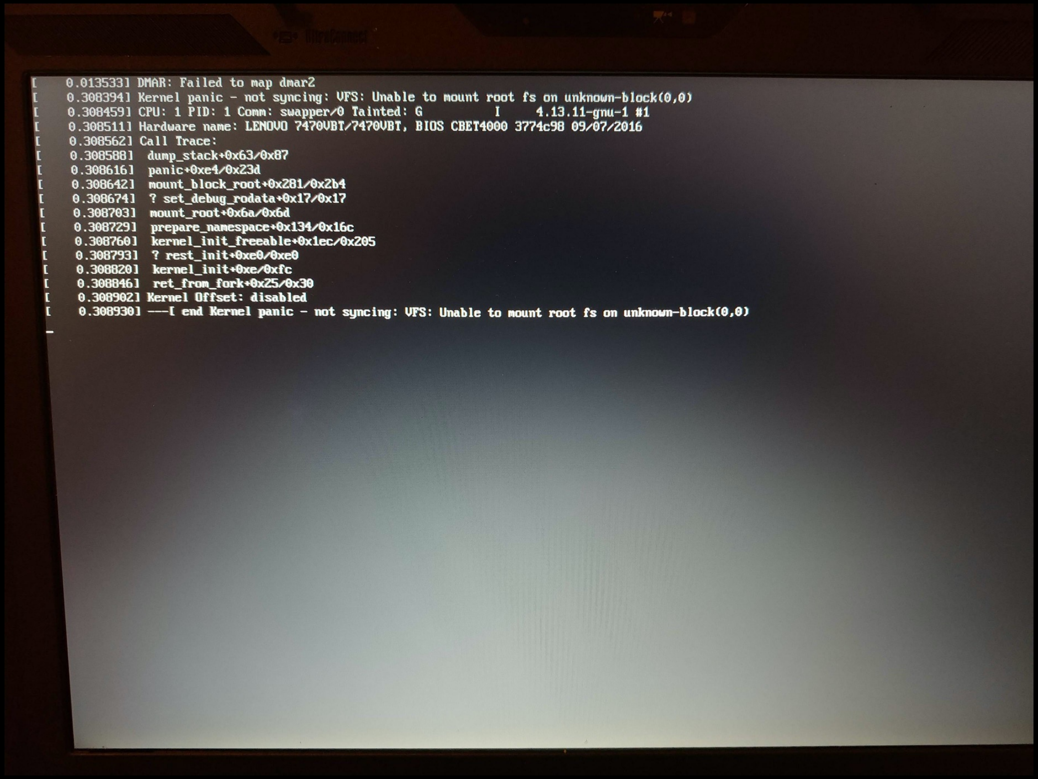
“Jailbreaking” or “rooting” devices often depends on finding and leveraging Kernel bugs

Remember JailbreakMe?

It used a remote code execution primitive inside Safari to trigger a kernel-level exploit to bypass Apple’s code-signing protection

Jailbreak Game Console

Kernel Basics



```
[    0.013533] DMAR: Failed to map dmar2
[    0.308394] Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)
[    0.308459] CPU: 1 PID: 1 Comm: swapper/0 Tainted: G           I     4.13.11-gnu-1 #1
[    0.308511] Hardware name: LENOVO 7470VBT/7470VBT, BIOS CBET4000 3774c98 09/07/2016
[    0.308562] Call Trace:
[    0.308588]  dump_stack+0x63/0x87
[    0.308616]  panic+0xe4/0x23d
[    0.308642]  mount_block_root+0x281/0x2b4
[    0.308674]  ? set_debug_rodata+0x17/0x17
[    0.308703]  mount_root+0x6a/0x6d
[    0.308729]  prepare_namespace+0x134/0x16c
[    0.308760]  kernel_init_freeable+0x1ec/0x205
[    0.308793]  ? rest_init+0xe0/0xe0
[    0.308820]  kernel_init+0xe/0xfc
[    0.308846]  ret_from_fork+0x25/0x30
[    0.308902] Kernel Offset: disabled
[    0.308930] ---[ end Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)
_
```

Your Kernel is:

Managing your Processes
Managing your Memory
Coordinating your Hardware

A crash oftentimes means a reboot!

In general, we want to spend as little time there as possible.

Kernel Basics

The Kernel is typically *the most powerful* place we can find bugs

But, how do we go from “vulnerability” to “privileged execution” *without bringing down the rest of the system?*

Kernel Exploitation Strategy

1. Find vulnerability in kernel code
2. Manipulate it to gain code execution
3. Elevate our process's privilege level
4. Survive the "trip" back to userland
5. Enjoy our root privileges

Kernel Exploitation Strategy

You already know how to find these!

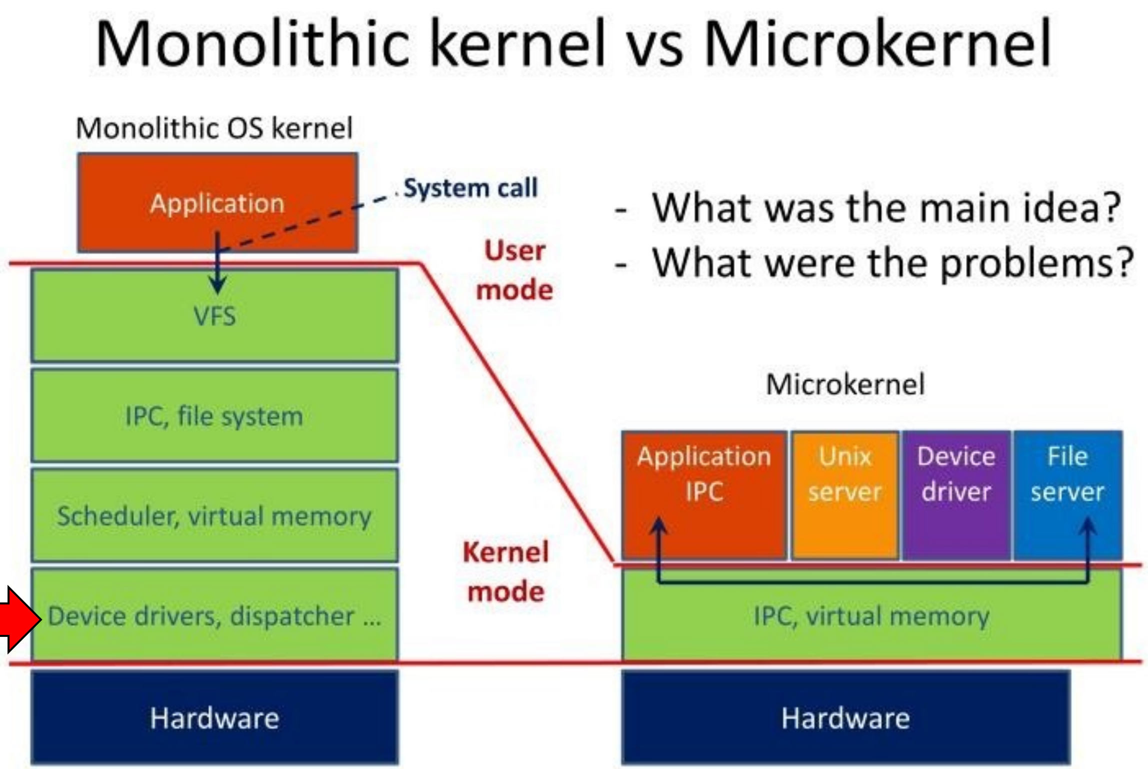
Kernel vulnerabilities are almost *exactly* the same as userland vulnerabilities.

1. Stack Overflows
2. Heap Overflows

By now, finding these should be a familiar process

Kernel Exploitation Strategy

- **Monolithic Kernel**
  - Monolithic kernel is a single large processes running entirely in a single address space. It is a single static binary file. All kernel services exist and execute in kernel address space. The kernel can invoke functions directly.



Monolithic kernel vs Microkernel

- What was the main idea?
- What were the problems?

LKMs

Kernel Exploitation Strategy

The most common place to find vulnerabilities is inside of Loadable Kernel Modules (LKMs).

LKMs are like executables that run in Kernel Space.
A few common uses are listed below:

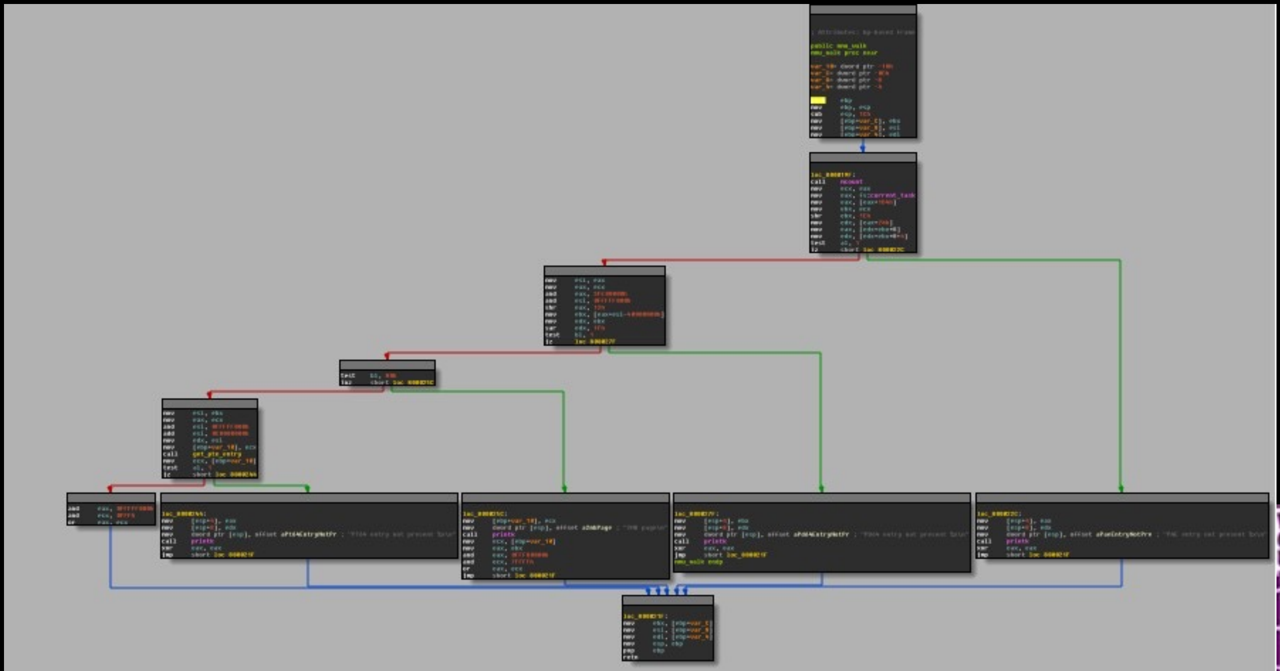
> Device Drivers
> Filesystem Drivers
> Networking Drivers
> Executable Interpreters
> Kernel Extensions
> (rootkits :P)

Kernel Exploitation Strategy

LKMs are just binary blobs like your familiar ELF’s, EXE’s and MACH-O’s. (On Linux, they even use the ELF format)

You can drop them into GDB and reverse-engineer them like you’re used to already.

WHERE THE KERNEL MEETS THE HARDWARE
3rd Edition
LINUX
DEVICE DRIVERS
JONATHAN CORBET, ALESSANDRO RUBINI & GREG KROAH-HARTMAN
O'REILLY



Kernel Exploitation Strategy

There's a few useful commands that deal with LKMs on Linux.

insmod	--->	Insert a module into the running kernel
rmmod	--->	Remove a module from the running kernel
lsmod	--->	List currently loaded modules

A general familiarity with these is helpful

Kernel Exploitation Strategy

The same basic exploitation techniques apply to Kernelspace
(After all, it's just x86 code!)

Shellcoding, ROP, Pointer Overwrites,
, etc can all be used to execute code in Kernel Land.

Kernel Functions

Common Library calls are sometimes *different* , so there is a slight learning curve involved.

printf()	--->	printk()
memcpy()	--->	copy_from_user()/copy_to_user()
malloc()	--->	kmalloc() (slab/slub allocator)
free()	--->	kfree()

Typically, whatever you want to know is a quick google-search or man page away.

Kernel Debugging

Debugging kernel code can be difficult

We can’t just run the kernel in gdb

You will often have to rely on stack dumps, error messages, and other “black box” techniques to infer what’s going on inside the kernel.

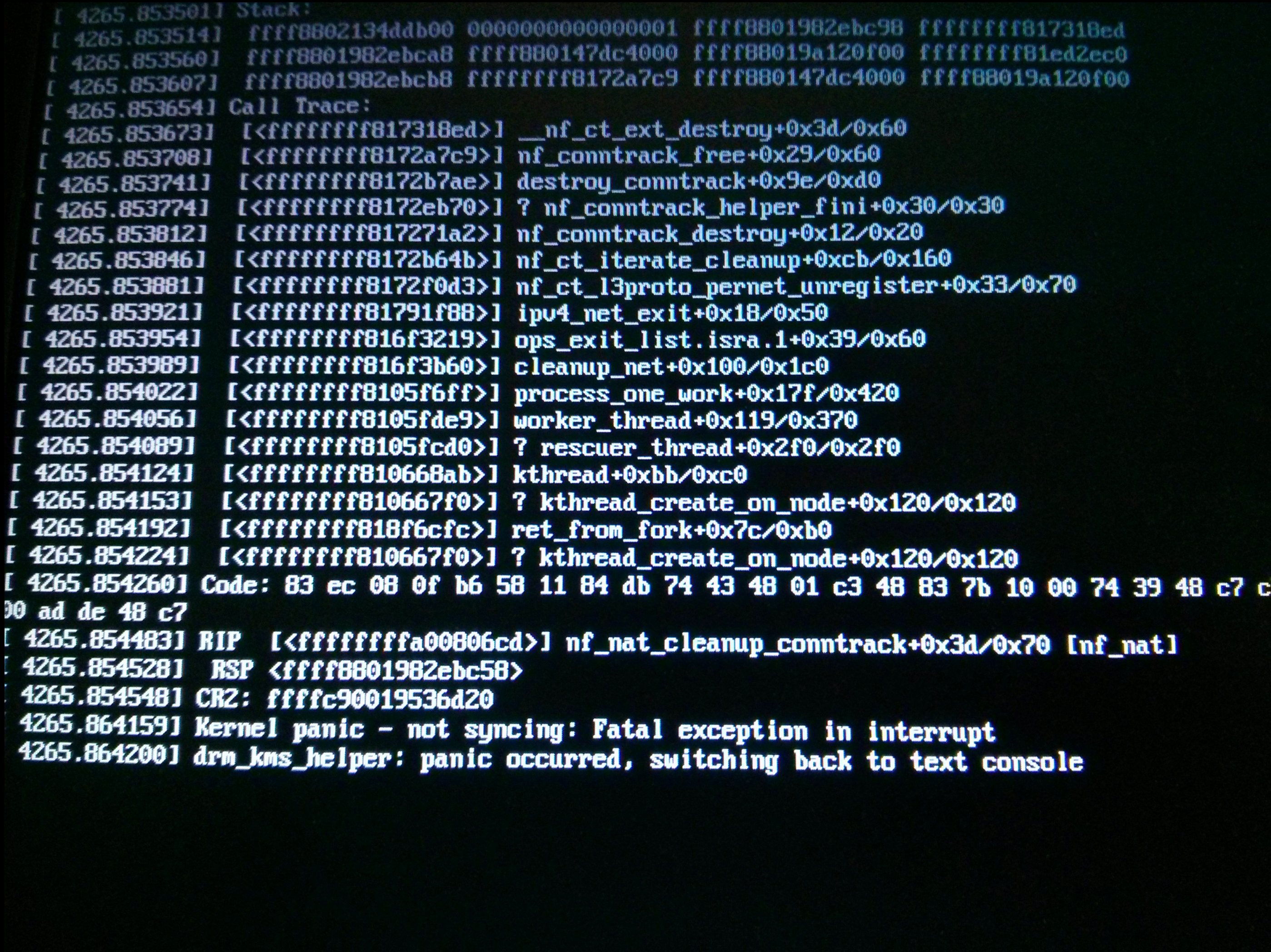
Kernel Debugging

This is an example of what you might see if you get a crash in the kernel.

Stack Dump

Call Trace

Register Dump



```
[ 4265.853501] Stack:
[ 4265.853514]  ffff8802134ddb00 0000000000000001 ffff8801982ebc98 ffffffff817318ed
[ 4265.853560]  ffff8801982ebca8 ffff880147dc4000 ffff88019a120f00 ffffffff81ed2ec0
[ 4265.853607]  ffff8801982ebcb8 ffffffff8172a7c9 ffff880147dc4000 ffff88019a120f00
[ 4265.853654] Call Trace:
[ 4265.853673]  [<ffffffff817318ed>] __nf_ct_ext_destroy+0x3d/0x60
[ 4265.853708]  [<ffffffff8172a7c9>] nf_conntrack_free+0x29/0x60
[ 4265.853741]  [<ffffffff8172b7ae>] destroy_conntrack+0x9e/0xd0
[ 4265.853774]  [<ffffffff8172eb70>] ? nf_conntrack_helper_fini+0x30/0x30
[ 4265.853812]  [<ffffffff817271a2>] nf_conntrack_destroy+0x12/0x20
[ 4265.853846]  [<ffffffff8172b64b>] nf_ct_iterate_cleanup+0xcb/0x160
[ 4265.853881]  [<ffffffff8172f0d3>] nf_ct_l3proto_pernet_unregister+0x33/0x70
[ 4265.853921]  [<ffffffff81791f88>] ipv4_net_exit+0x18/0x50
[ 4265.853954]  [<ffffffff816f3219>] ops_exit_list.isra.1+0x39/0x60
[ 4265.853989]  [<ffffffff816f3b60>] cleanup_net+0x100/0x1c0
[ 4265.854022]  [<ffffffff8105f6ff>] process_one_work+0x17f/0x420
[ 4265.854056]  [<ffffffff8105fde9>] worker_thread+0x119/0x370
[ 4265.854089]  [<ffffffff8105fcd0>] ? rescuer_thread+0x2f0/0x2f0
[ 4265.854124]  [<ffffffff810668ab>] kthread+0xbb/0xc0
[ 4265.854153]  [<ffffffff810667f0>] ? kthread_create_on_node+0x120/0x120
[ 4265.854192]  [<ffffffff818f6cfc>] ret_from_fork+0x7c/0xb0
[ 4265.854224]  [<ffffffff810667f0>] ? kthread_create_on_node+0x120/0x120
[ 4265.854260] Code: 83 ec 08 0f b6 58 11 84 db 74 43 48 01 c3 48 83 7b 10 00 74 39 48 c7 c
00 ad de 48 c7
[ 4265.854483] RIP  [<ffffffffa00806cd>] nf_nat_cleanup_conntrack+0x3d/0x70 [nf_nat]
[ 4265.854528]  RSP <ffff8801982ebc58>
[ 4265.854548] CR2: ffffc90019536d20
[ 4265.864159] Kernel panic - not syncing: Fatal exception in interrupt
[ 4265.864200] drm_kms_helper: panic occurred, switching back to text console
```

Kernel Debugging

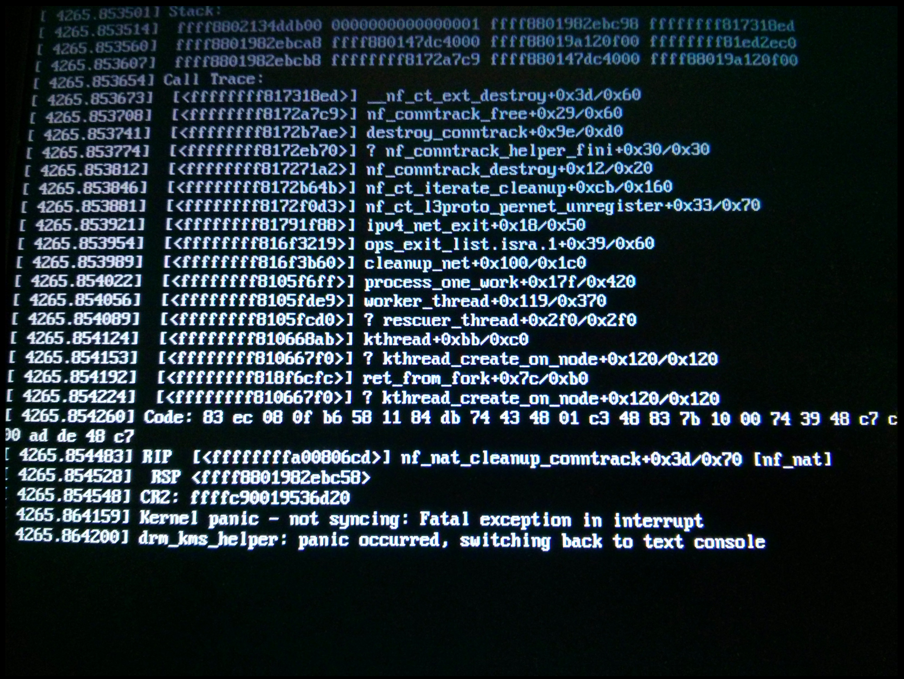
This is an example of what you might see if you get a crash in the kernel.

Stack Dump

Call Trace

Register Dump

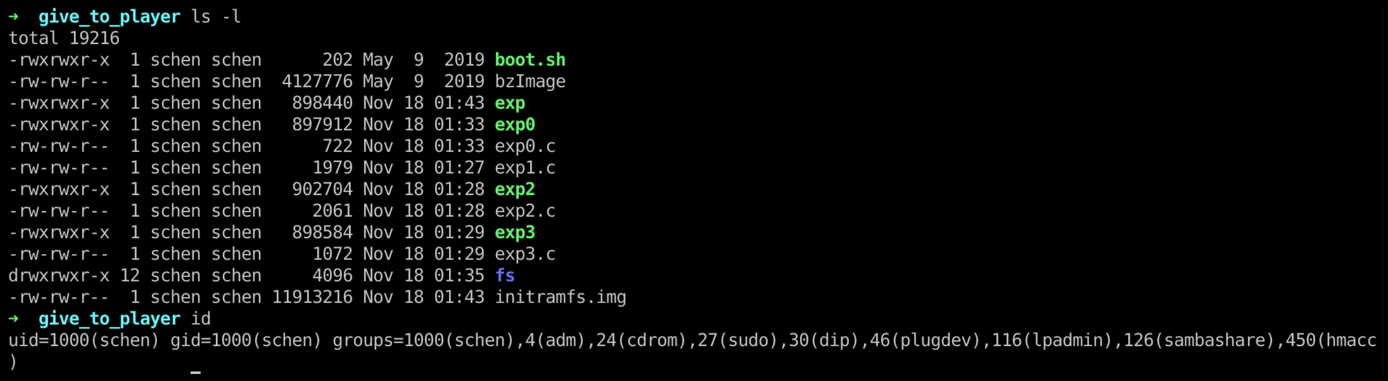
You might be able to see this with dmesg if the crash is not fatal.



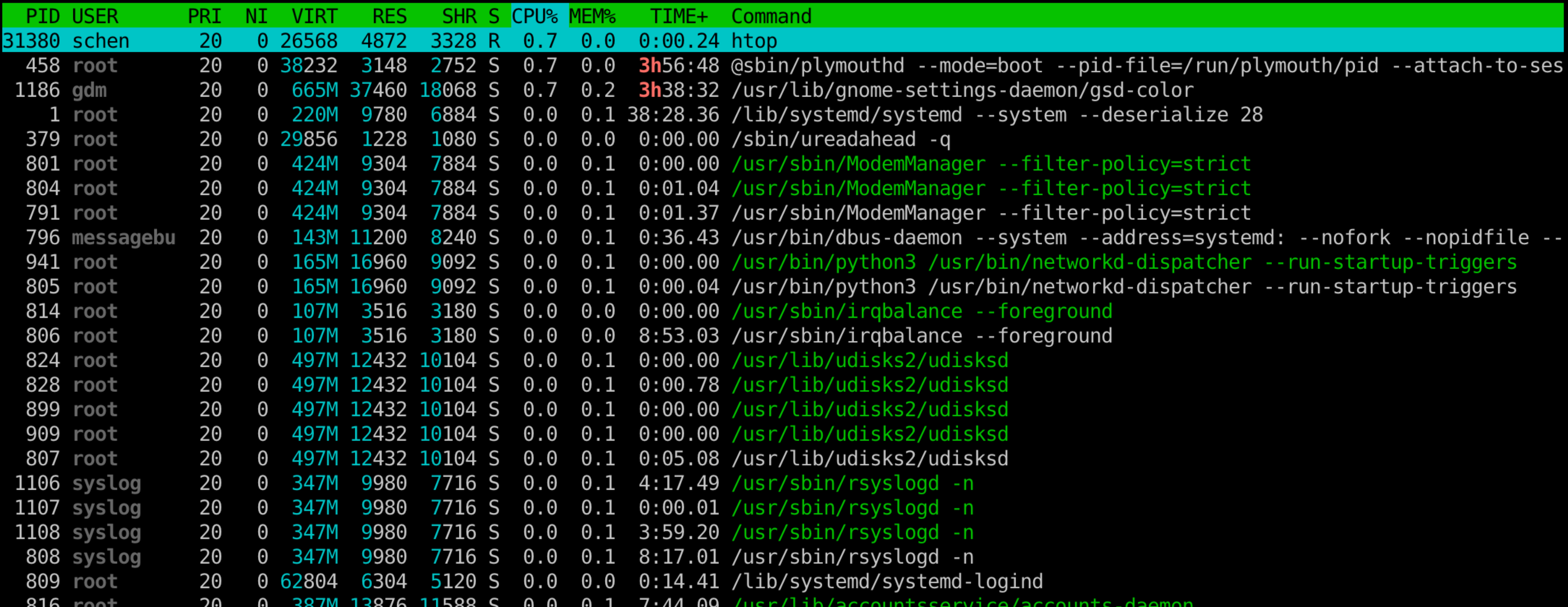
```
[ 4265.853501] Stack:
[ 4265.853514]  ffff8802134ddb00 0000000000000001 ffff8801982ebc98 ffffffff817318ed
[ 4265.853560]  ffff8801982ebca8 ffff880147dc4000 ffff88019a120f00 ffffffff81ed2ec0
[ 4265.853607]  ffff8801982ebcb8 ffffffff8172a7c9 ffff880147dc4000 ffff88019a120f00
[ 4265.853654] Call Trace:
[ 4265.853673]  [<ffffffff817318ed>] __nf_ct_ext_destroy+0x3d/0x60
[ 4265.853708]  [<ffffffff8172a7c9>] nf_conntrack_free+0x29/0x60
[ 4265.853741]  [<ffffffff8172b7ae>] destroy_conntrack+0x9e/0xd0
[ 4265.853774]  [<ffffffff8172eb70>] ? nf_conntrack_helper_fini+0x30/0x30
[ 4265.853812]  [<ffffffff817271a2>] nf_conntrack_destroy+0x12/0x20
[ 4265.853846]  [<ffffffff8172b64b>] nf_ct_iterate_cleanup+0xcb/0x160
[ 4265.853881]  [<ffffffff8172f0d3>] nf_ct_l3proto_pernet_unregister+0x33/0x70
[ 4265.853921]  [<ffffffff81791f88>] ipv4_net_exit+0x18/0x50
[ 4265.853954]  [<ffffffff816f3219>] ops_exit_list.isra.1+0x39/0x60
[ 4265.853989]  [<ffffffff816f3b60>] cleanup_net+0x100/0x1c0
[ 4265.854022]  [<ffffffff8105f6ff>] process_one_work+0x17f/0x420
[ 4265.854056]  [<ffffffff8105fde9>] worker_thread+0x119/0x370
[ 4265.854089]  [<ffffffff8105fcd0>] ? rescuer_thread+0x2f0/0x2f0
[ 4265.854124]  [<ffffffff810668ab>] kthread+0xbb/0xc0
[ 4265.854153]  [<ffffffff810667f0>] ? kthread_create_on_node+0x120/0x120
[ 4265.854192]  [<ffffffff818f6cfc>] ret_from_fork+0x7c/0xb0
[ 4265.854224]  [<ffffffff810667f0>] ? kthread_create_on_node+0x120/0x120
[ 4265.854260] Code: 83 ec 08 0f b6 58 11 84 db 74 43 48 01 c3 48 83 7b 10 00 74 39 48 c7 c
00 ad de 48 c7
[ 4265.854483] RIP  [<ffffffffa00806cd>] nf_nat_cleanup_conntrack+0x3d/0x70 [nf_nat]
[ 4265.854528]  RSP <ffff8801982ebc58>
[ 4265.854548] CR2: ffffc90019536d20
 4265.864159] Kernel panic - not syncing: Fatal exception in interrupt
 4265.864200] drm_kms_helper: panic occurred, switching back to text console
```

Traditional UNIX credentials.

- Real User ID
- Real Group ID



```
➜  give_to_player ls -l
total 19216
-rwxrwxr-x  1 schen schen      202 May  9  2019 boot.sh
-rw-rw-r--  1 schen schen  4127776 May  9  2019 bzImage
-rwxrwxr-x  1 schen schen   898440 Nov 18 01:43 exp
-rwxrwxr-x  1 schen schen   897912 Nov 18 01:33 exp0
-rw-rw-r--  1 schen schen      722 Nov 18 01:33 exp0.c
-rw-rw-r--  1 schen schen     1979 Nov 18 01:27 exp1.c
-rwxrwxr-x  1 schen schen   902704 Nov 18 01:28 exp2
-rw-rw-r--  1 schen schen     2061 Nov 18 01:28 exp2.c
-rwxrwxr-x  1 schen schen   898584 Nov 18 01:29 exp3
-rw-rw-r--  1 schen schen     1072 Nov 18 01:29 exp3.c
drwxrwxr-x 12 schen schen     4096 Nov 18 01:35 fs
-rw-rw-r--  1 schen schen 11913216 Nov 18 01:43 initramfs.img
➜  give_to_player id
uid=1000(schen) gid=1000(schen) groups=1000(schen),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),116(lpadmin),126(sambashare),450(hmacc
)
```

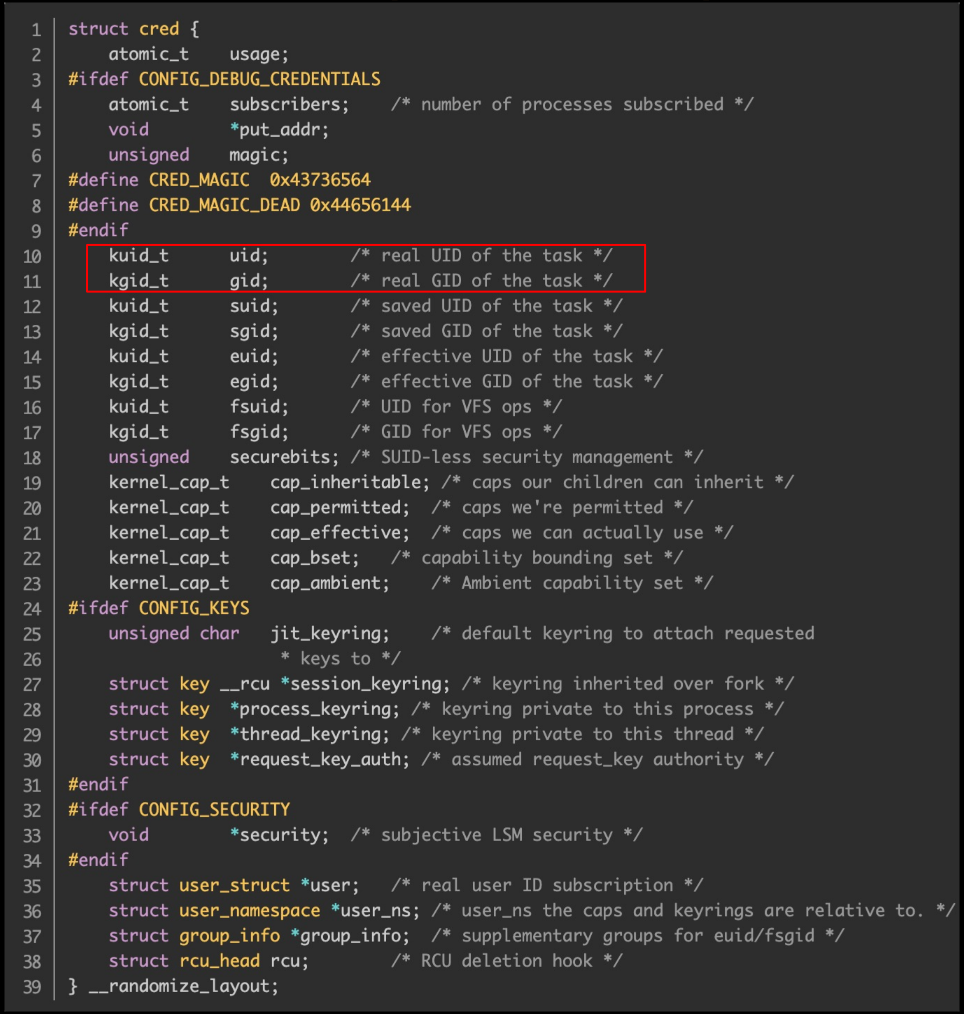


```
  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
31380 schen      20   0 26568  4872  3328 R  0.7  0.0  0:00.24 htop
  458 root       20   0 38232  3148  2752 S  0.7  0.0  3h56:48 @sbin/plymouthd --mode=boot --pid-file=/run/plymouth/pid --attach-to-ses
 1186 gdm        20   0  665M 37460 18068 S  0.7  0.2  3h38:32 /usr/lib/gnome-settings-daemon/gsd-color
    1 root       20   0  220M  9780  6884 S  0.0  0.1 38:28.36 /lib/systemd/systemd --system --deserialize 28
  379 root       20   0 29856  1228  1080 S  0.0  0.0  0:00.00 /sbin/ureadahead -q
  801 root       20   0  424M  9304  7884 S  0.0  0.1  0:00.00 /usr/sbin/ModemManager --filter-policy=strict
  804 root       20   0  424M  9304  7884 S  0.0  0.1  0:01.04 /usr/sbin/ModemManager --filter-policy=strict
  791 root       20   0  424M  9304  7884 S  0.0  0.1  0:01.37 /usr/sbin/ModemManager --filter-policy=strict
  796 messagebu  20   0  143M 11200  8240 S  0.0  0.1  0:36.43 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --
  941 root       20   0  165M 16960  9092 S  0.0  0.1  0:00.00 /usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-triggers
  805 root       20   0  165M 16960  9092 S  0.0  0.1  0:00.04 /usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-triggers
  814 root       20   0  107M  3516  3180 S  0.0  0.0  0:00.00 /usr/sbin/irqbalance --foreground
  806 root       20   0  107M  3516  3180 S  0.0  0.0  8:53.03 /usr/sbin/irqbalance --foreground
  824 root       20   0  497M 12432 10104 S  0.0  0.1  0:00.00 /usr/lib/udisks2/udisksd
  828 root       20   0  497M 12432 10104 S  0.0  0.1  0:00.78 /usr/lib/udisks2/udisksd
  899 root       20   0  497M 12432 10104 S  0.0  0.1  0:00.00 /usr/lib/udisks2/udisksd
  909 root       20   0  497M 12432 10104 S  0.0  0.1  0:00.00 /usr/lib/udisks2/udisksd
  807 root       20   0  497M 12432 10104 S  0.0  0.1  0:05.08 /usr/lib/udisks2/udisksd
 1106 syslog     20   0  347M  9980  7716 S  0.0  0.1  4:17.49 /usr/sbin/rsyslogd -n
 1107 syslog     20   0  347M  9980  7716 S  0.0  0.1  0:00.01 /usr/sbin/rsyslogd -n
 1108 syslog     20   0  347M  9980  7716 S  0.0  0.1  3:59.20 /usr/sbin/rsyslogd -n
  808 syslog     20   0  347M  9980  7716 S  0.0  0.1  8:17.01 /usr/sbin/rsyslogd -n
  809 root       20   0 62804  6304  5120 S  0.0  0.0  0:14.41 /lib/systemd/systemd-logind
  816 root       20   0  387M 13876 11588 S  0.0  0.1  7:44.09 /usr/lib/accountsservice/accounts-daemon
```

Elevate Privileges

Remember: The Kernel manages running processes

Therefore: The Kernel keeps track of permissions



```
struct cred {
	atomic_t	usage;
#ifdef CONFIG_DEBUG_CREDENTIALS
	atomic_t	subscribers;	/* number of processes subscribed */
	void		*put_addr;
	unsigned	magic;
#define CRED_MAGIC	0x43736564
#define CRED_MAGIC_DEAD	0x44656144
#endif
	kuid_t		uid;		/* real UID of the task */
	kgid_t		gid;		/* real GID of the task */
	kuid_t		suid;		/* saved UID of the task */
	kgid_t		sgid;		/* saved GID of the task */
	kuid_t		euid;		/* effective UID of the task */
	kgid_t		egid;		/* effective GID of the task */
	kuid_t		fsuid;		/* UID for VFS ops */
	kgid_t		fsgid;		/* GID for VFS ops */
	unsigned	securebits;	/* SUID-less security management */
	kernel_cap_t	cap_inheritable; /* caps our children can inherit */
	kernel_cap_t	cap_permitted;	/* caps we're permitted */
	kernel_cap_t	cap_effective;	/* caps we can actually use */
	kernel_cap_t	cap_bset;	/* capability bounding set */
	kernel_cap_t	cap_ambient;	/* Ambient capability set */
#ifdef CONFIG_KEYS
	unsigned char	jit_keyring;	/* default keyring to attach requested
					 * keys to */
	struct key __rcu *session_keyring; /* keyring inherited over fork */
	struct key	*process_keyring; /* keyring private to this process */
	struct key	*thread_keyring; /* keyring private to this thread */
	struct key	*request_key_auth; /* assumed request_key authority */
#endif
#ifdef CONFIG_SECURITY
	void		*security;	/* subjective LSM security */
#endif
	struct user_struct *user;	/* real user ID subscription */
	struct user_namespace *user_ns; /* user_ns the caps and keyrings are relative to. */
	struct group_info *group_info;	/* supplementary groups for euid/fsgid */
	struct rcu_head	rcu;		/* RCU deletion hook */
} __randomize_layout;
```

https://code.woboq.org/linux/linux/include/linux/cred.h.html#cred

Elevate Privileges

Conveniently, the Linux Kernel has two wrapper functions for updating process credentials and generating process credentials!

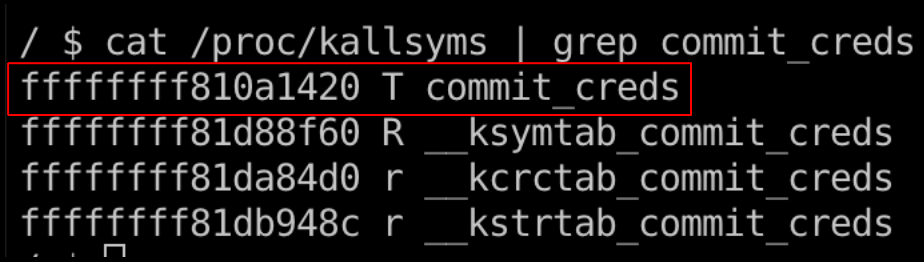
```
 int commit_creds(struct cred *new) {
       …
 }


struct cred *prepare_kernel_cred(struct task_struct *daemon) {

}
```

Elevate Privileges

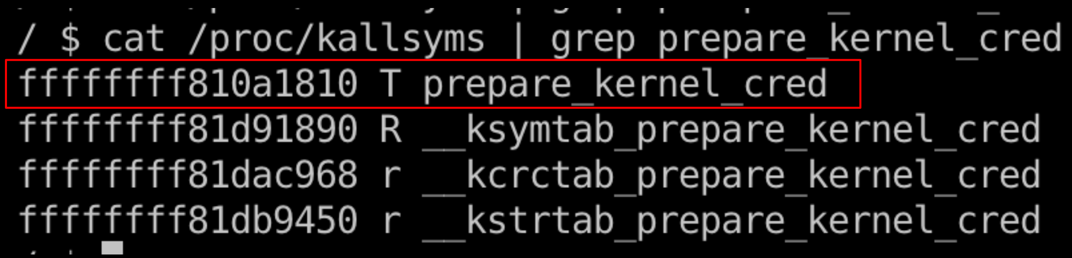
Now we can map out what we need to do

```
commit_creds(prepare_kernel_cred(0));
```

We can find their addresses in **/proc/kallsyms**



```
/ $ cat /proc/kallsyms | grep commit_creds
ffffffff810a1420 T commit_creds
ffffffff81d88f60 R __ksymtab_commit_creds
ffffffff81da84d0 r __kcrctab_commit_creds
ffffffff81db948c r __kstrtab_commit_creds
```



```
/ $ cat /proc/kallsyms | grep prepare_kernel_cred
ffffffff810a1810 T prepare_kernel_cred
ffffffff81d91890 R __ksymtab_prepare_kernel_cred
ffffffff81dac968 r __kcrctab_prepare_kernel_cred
ffffffff81db9450 r __kstrtab_prepare_kernel_cred
```

Returning to UserSpace

Why bother returning to Userspace?

Most useful things we want to do are *much* easier from userland.

In KernelSpace, there’s no easy way to:

> Modify the filesystem
> Create a new process
> Create network connections

Returning to UserSpace

How does the kernel do it?

```
push    $SS_USER_VALUE
push    $USERLAND_STACK
push    $USERLAND_EFLAGS
push    $CS_USER_VALUE
push    $USERLAND_FUNCTION_ADDRESS
swapgs
iretq
```

This *will usually* get you out of “Kernel Mode” safely.

Returning to UserSpace

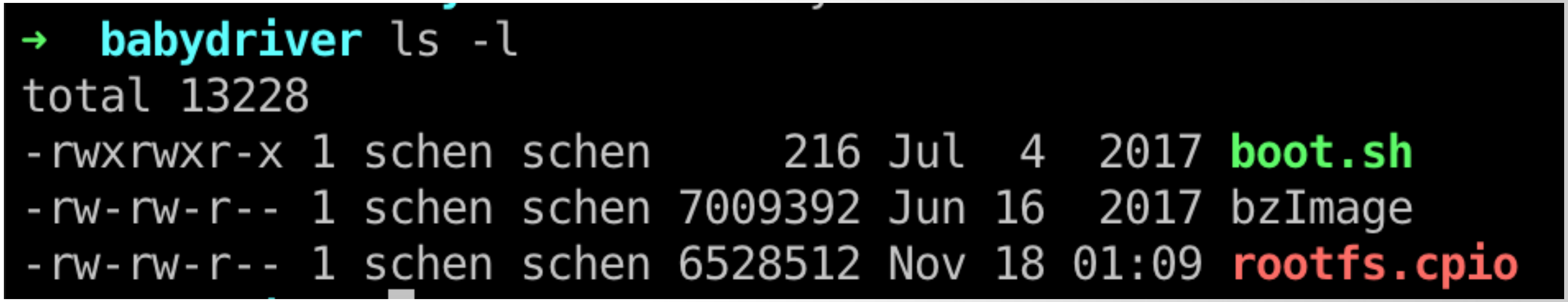
For exploitation, the easiest strategy is highjacking execution, and letting the kernel return by itself.

> Function Pointer Overwrites
> Syscall Table Highjacking
> Use-After-Free

You need to be very careful about destroying Kernel state.

A segfault probably means a reboot!

Example: Babydriver



```
→  babydriver ls -l
total 13228
-rwxrwxr-x 1 schen schen     216 Jul  4  2017 boot.sh
-rw-rw-r-- 1 schen schen 7009392 Jun 16  2017 bzImage
-rw-rw-r-- 1 schen schen 6528512 Nov 18 01:09 rootfs.cpio
```

https://github.com/ctf-wiki/ctf-challenges/tree/master/pwn/kernel

Kernel Space Protections

By now, you're familiar with the alphabet soup of exploit mitigations

DEP
ASLR
Canaries
etc...

Green: Present in Kernel Space
Yellow: Present, with caveats

There's a whole new alphabet soup for Kernel Mitigations!

Kernel Space Protections

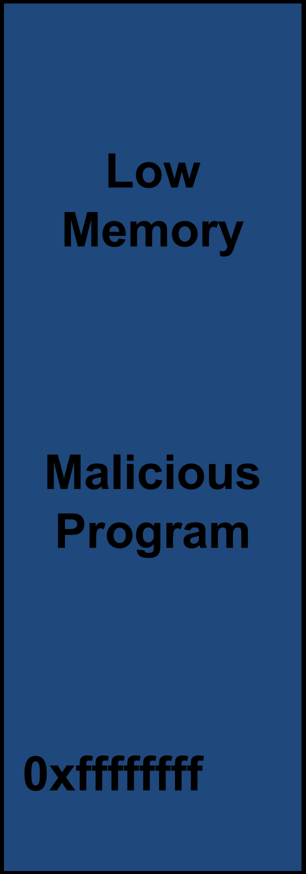
Some new words in our soup (There's plenty more...)

MMAP_MIN_ADDR
KALLSYMS
RANDSTACK
STACKLEAK
SMEP / SMAP

Most of these will be off for the labs!

MMAP_MIN_ADDR

This makes exploiting NULL pointer dereferences harder.



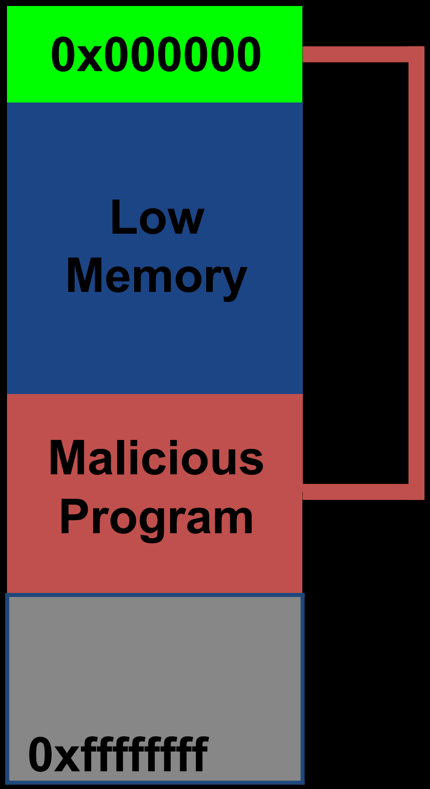
Low Memory

Malicious Program

0xffffffff

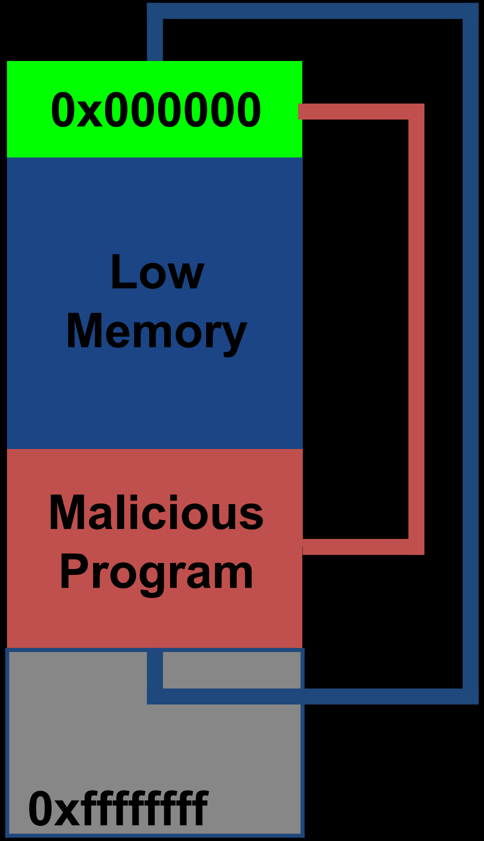
MMAP_MIN_ADDR

This makes exploiting NULL pointer dereferences harder.



Program does mmap(0,....)

MMAP_MIN_ADDR



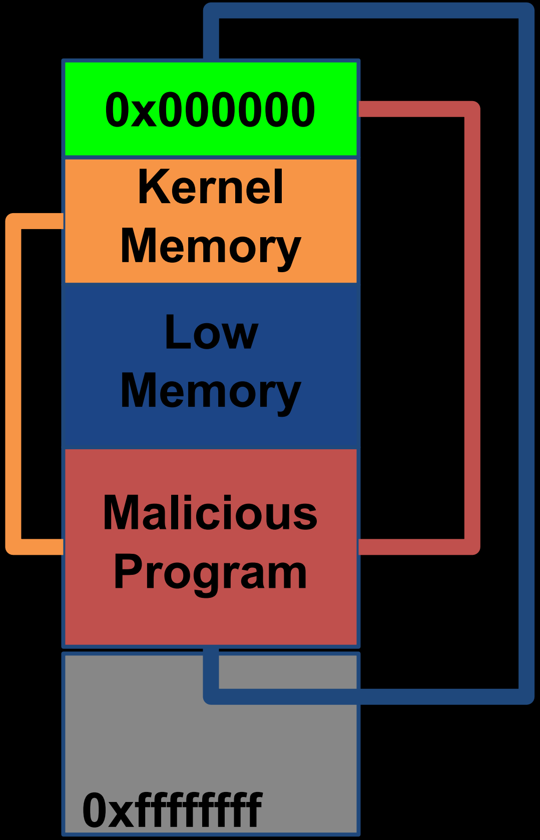
NULL pointer dereferences

- Program does mmap(0,....)

Program writes malicious Code

MMAP_MIN_ADDR

This makes exploiting NULL pointer dereferences harder.



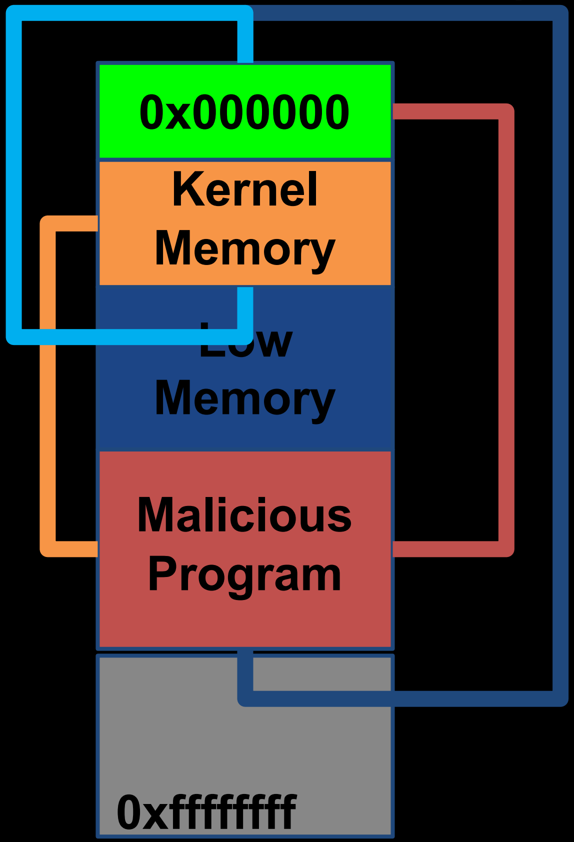
Program does mmap(0,....)

Program writes malicious Code

Program triggers Kernel Bug

MMAP_MIN_ADDR

This makes exploiting NULL pointer dereferences harder.



Program does mmap(0,....)

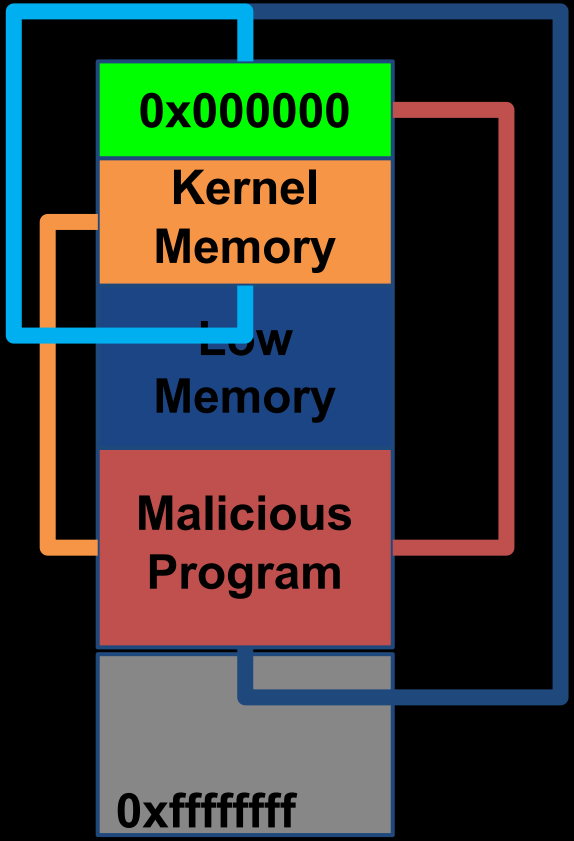
Program writes malicious Code

Program triggers Kernel Bug

Kernel starts executing malicious Code

MMAP_MIN_ADDR

This makes exploiting NULL pointer dereferences harder.



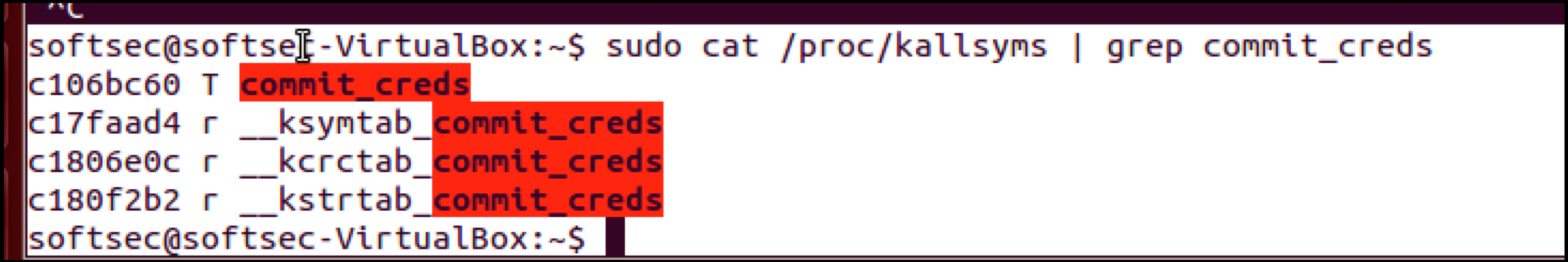
mmap_min_addr disallows programs from allocating low memory.

Makes it much more difficult to exploit a simple NULL pointer dereference in the kernel.

KALLSYMS

/proc/kallsyms gives the address of all symbols in the kernel.

We need this information to write reliable exploits without an info-leak!

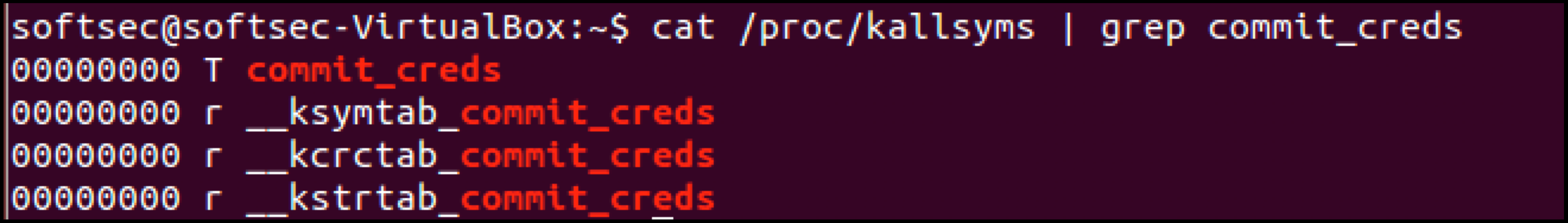


```
^C
softsec@softsec-VirtualBox:~$ sudo cat /proc/kallsyms | grep commit_creds
c106bc60 T commit_creds
c17faad4 r __ksymtab_commit_creds
c1806e0c r __kcrctab_commit_creds
c180f2b2 r __kstrtab_commit_creds
softsec@softsec-VirtualBox:~$
```

KALLSYMS

kallsyms used to be world-readable.

Now, it returns 0’s for unprivileged users



```
softsec@softsec-VirtualBox:~$ cat /proc/kallsyms | grep commit_creds
00000000 T commit_creds
00000000 r __ksymtab_commit_creds
00000000 r __kcrctab_commit_creds
00000000 r __kstrtab_commit_creds
```

Can still be a useful source of information on older systems

SMEP / SMAP

SMEP: Supervisor Mode Execution Protection

Introduced in Intel IvyBridge

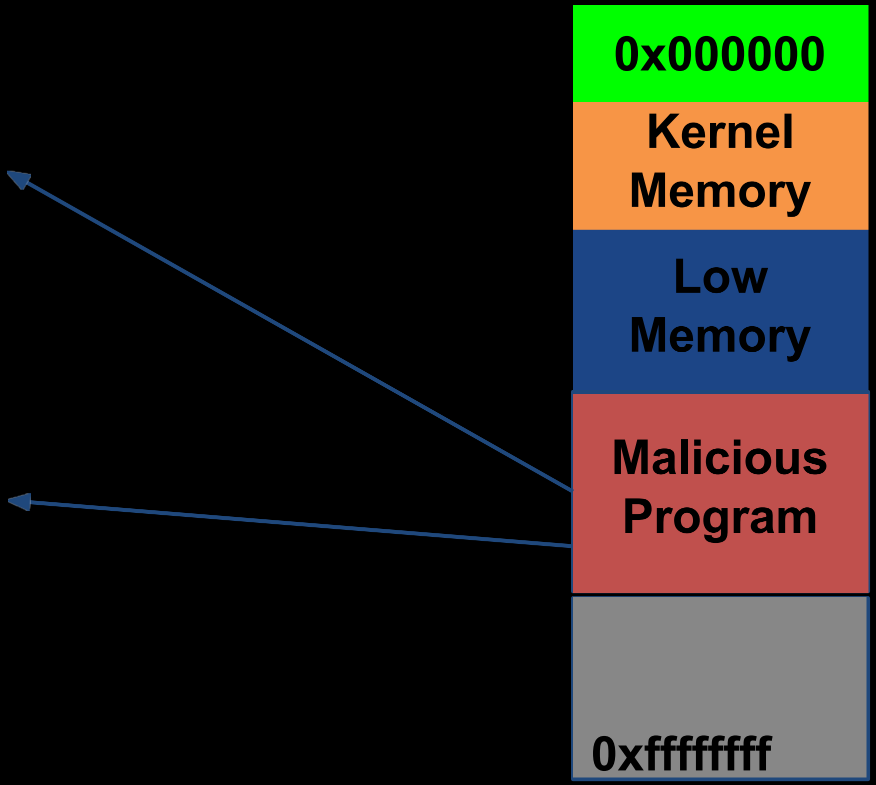
SMAP: Supervisor Mode Access Protection

Introduced in Intel Haswell

SMEP / SMAP

Common Exploitation Technique: Supply your own “get root” code.

```
void get_r00t() {
        commit_creds(prepare_kernel_cred(0));
}

int main(int argc, char * argv) {
        …
        trigger_fp_overwrite(&get_r00t);
        …
        //trigger fp use
        trigger_vuln_fp();
        // Kernel Executes get_r00t
        ...
        // Now we have root
        system(“/bin/sh”);
}
```



SMEP / SMAP

SMEP prevents this type of attack by triggering a page fault if the processor tries to execute memory that has the “user” bit set while in “ring 0”.

SMAP works similarly, but for data access in general

This doesn’t *prevent* vulnerabilities, but it adds considerable work to developing a working exploit

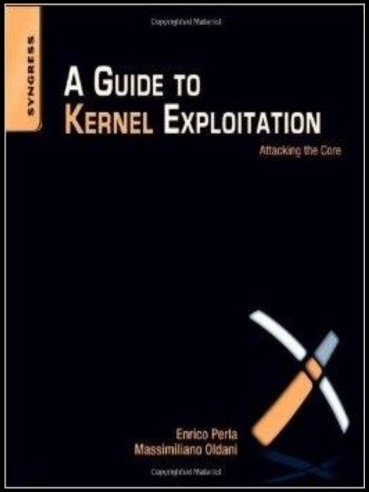
We need to use ROP, or somehow get executable code into kernel memory.

Conclusion

Kernel Exploitation is *weird*, but *extremely powerful*

As userland exploit-dev becomes more challenging and more expensive, kernelspace is becoming a more attractive target.

A single bug can be used to bypass sandboxes, and gain root privileges, which may otherwise be impossible



Q & A