Privilege Escalation using DOP in x86-64 macOS

Yoochan Lee, Sangjun Song, Junoh Lee, and Jeongsu Choi

Whoami?

Team **GYG**

We focus on **CTF** and **Bug Hunting.**

Yoochan Lee
- Ph.D student
- Linux, macOS

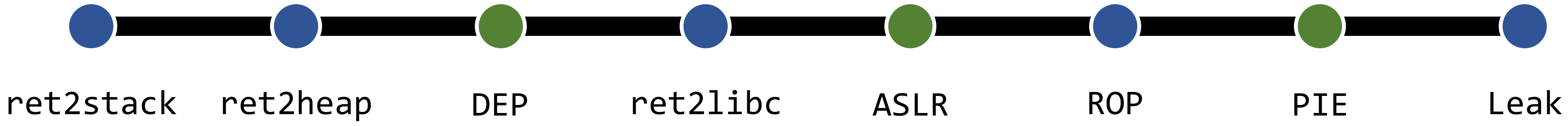
Sangjun Song
- Security Researcher
- Web3

Junoh Lee
- Security Researcher
- Windows

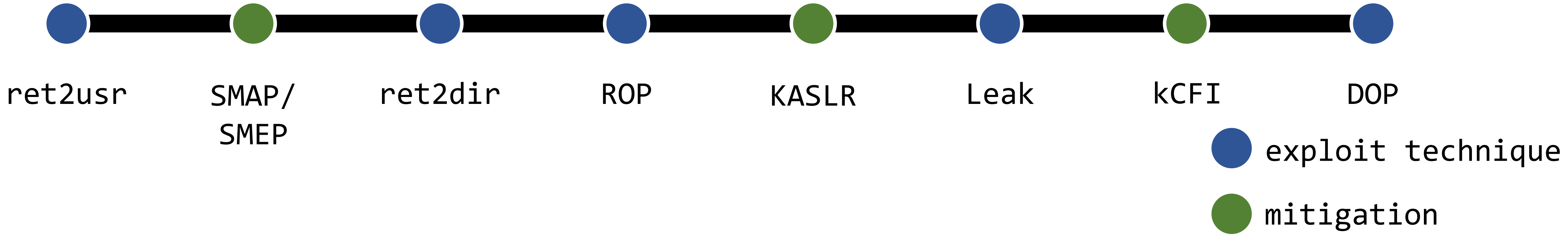
Jeongsu Choi
- Security Researcher
- Web

The history

In user application



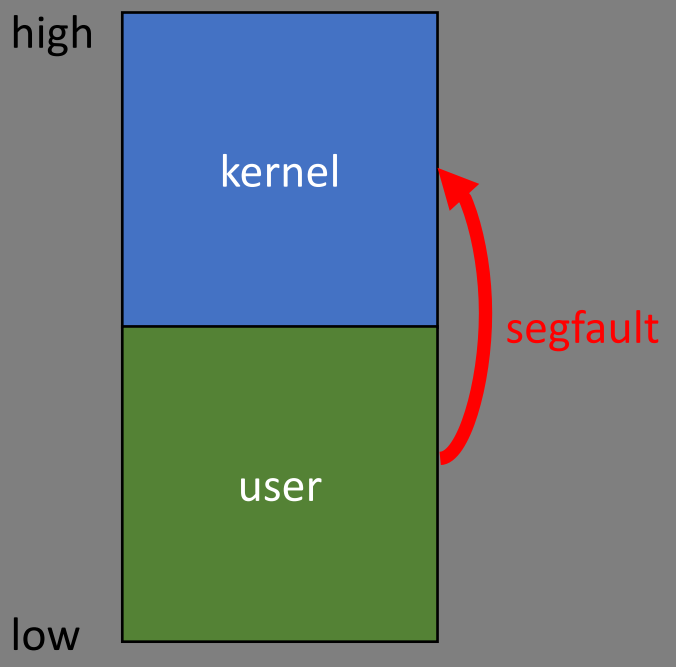
In kernel



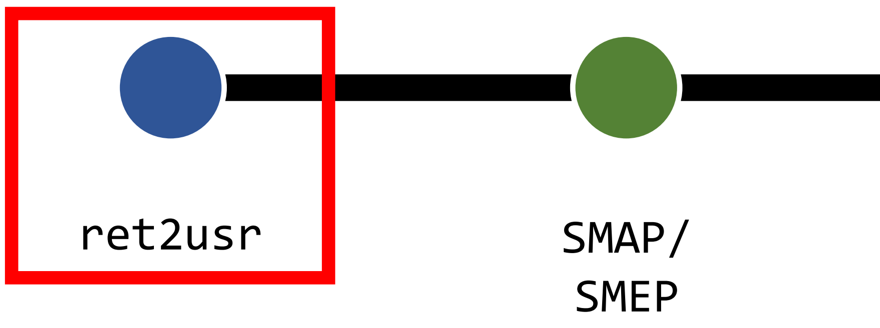
The history

- ret2usr

Change RIP register to user space address



In kernel

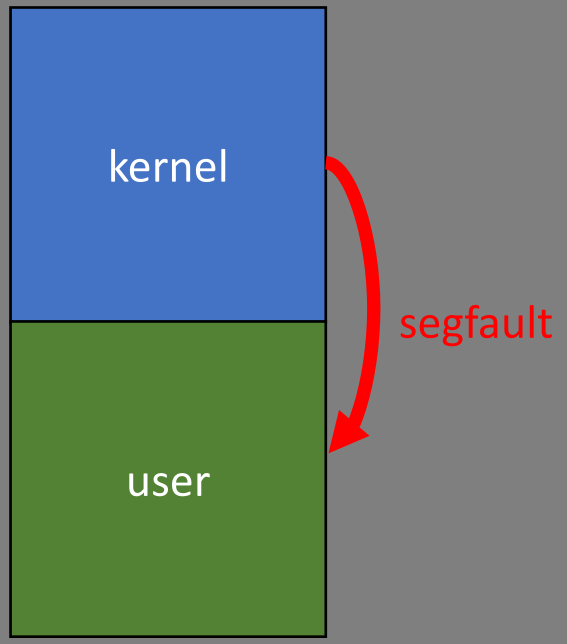


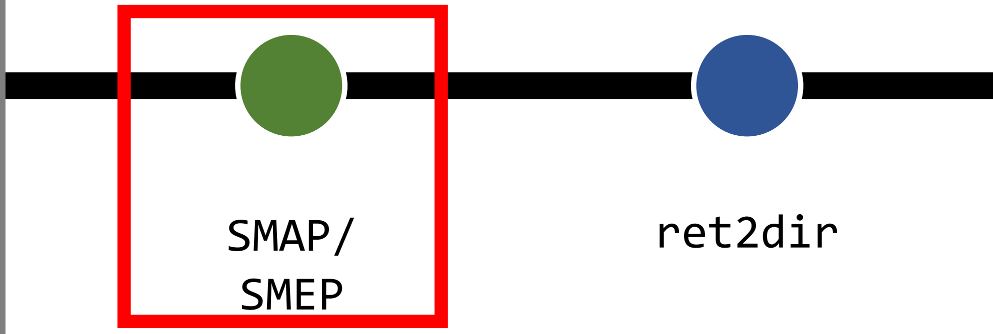
The history

In kernel

- SMAP/SMEP

Prevent user memory access when kernel runs

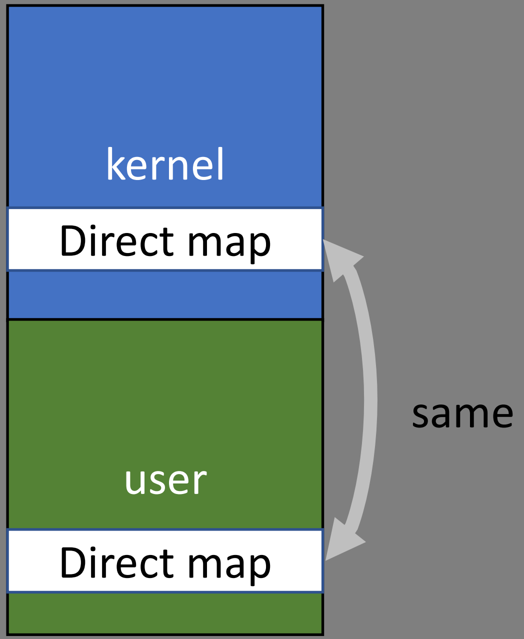




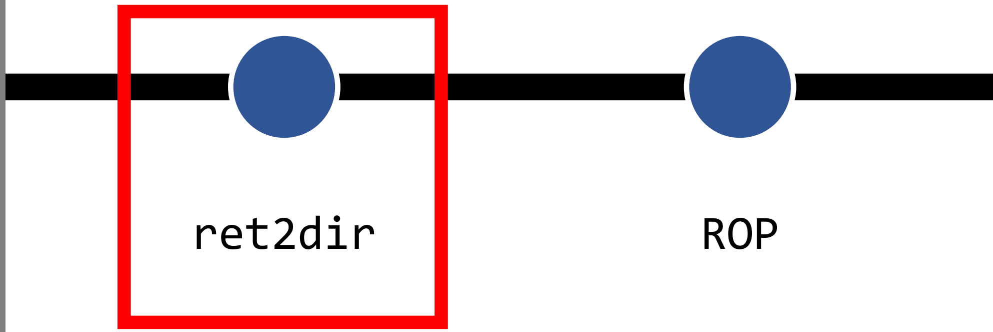
The history

- ret2dir

Using direct mapping area for executing shellcode



In kernel



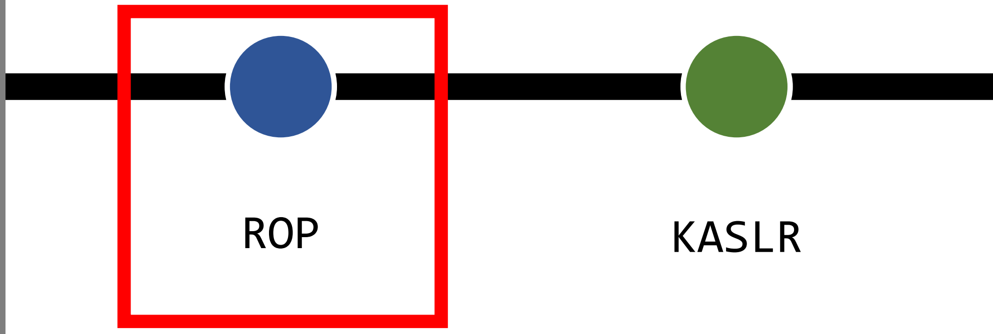
The history

- ROP

Return-Oriented Programming

Manipulating control-flow to execute code snippets (ROP gadget) sequentially.

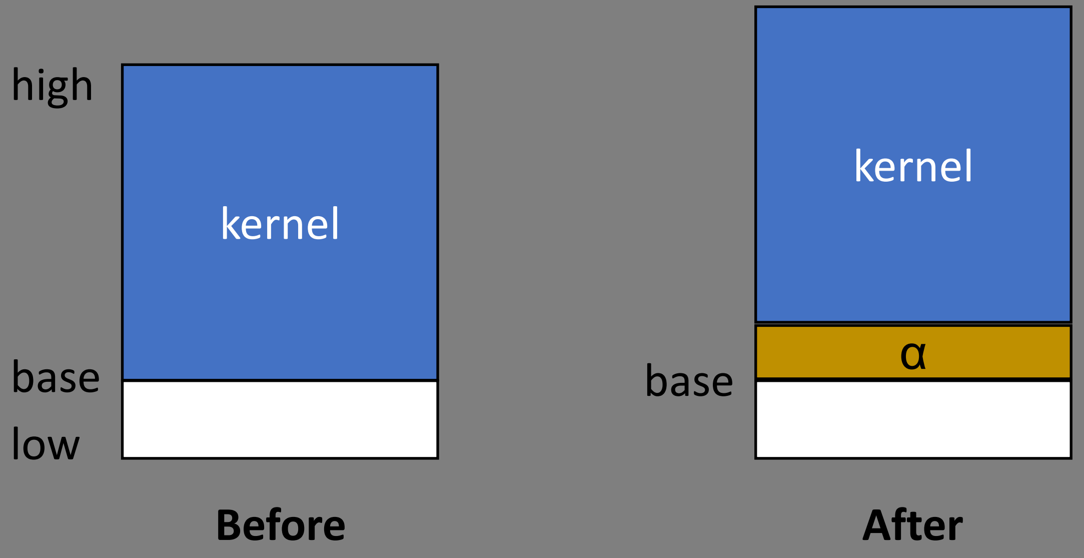
In kernel



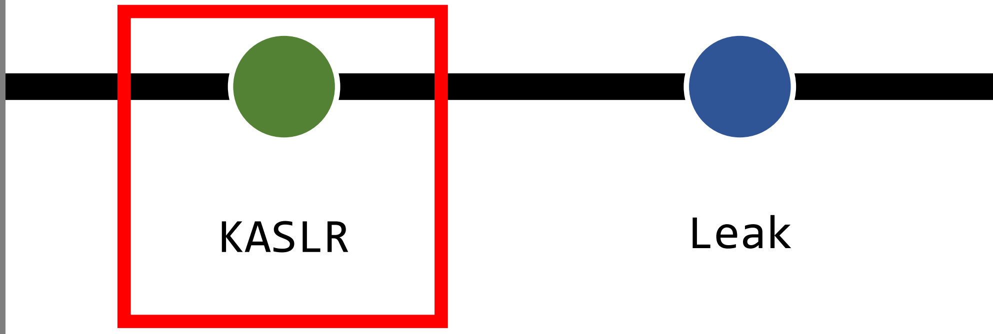
The history

- Kernel ASLR

For preventing the execution of ROP gadget, the kernel randomizes the kernel memory address at boot time



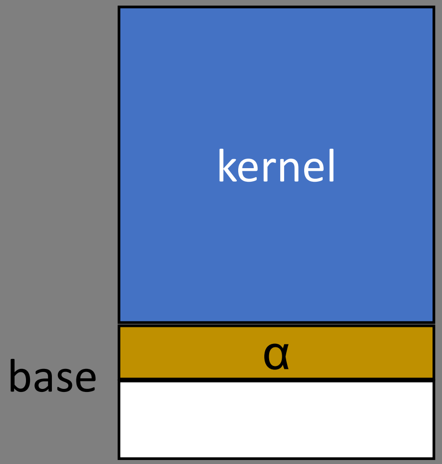
In kernel



The history

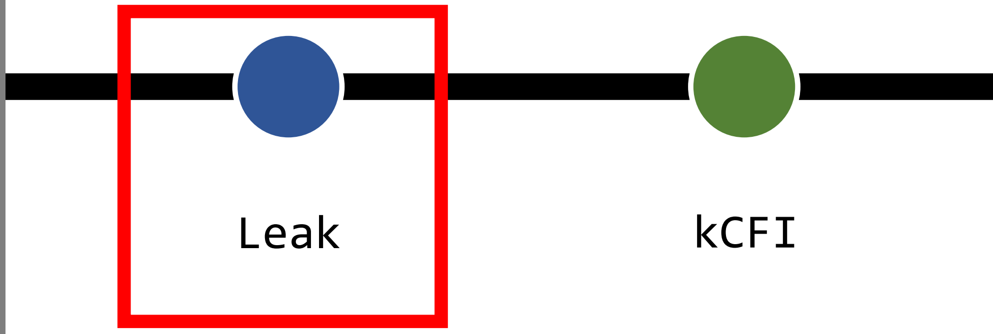
- Information Leakage

For bypassing KASLR, the attacker needs to leak a kernel address for calculating changed address



α = Leaked pointer - offset - base

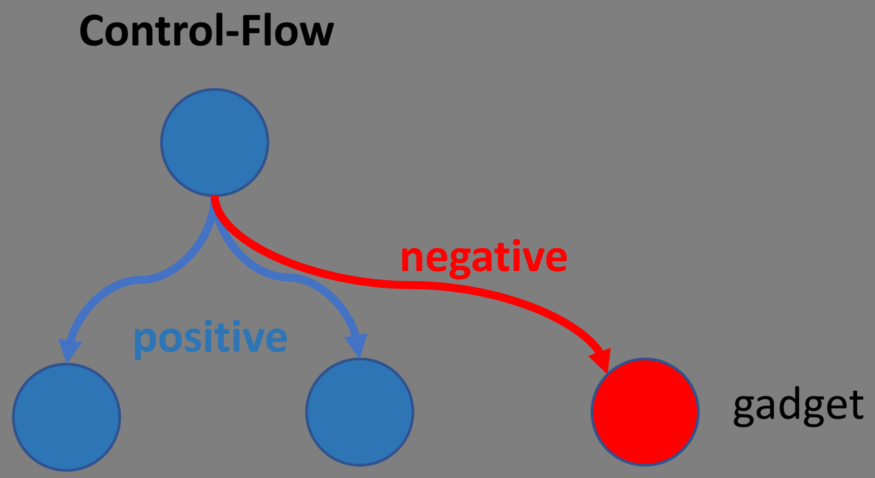
In kernel



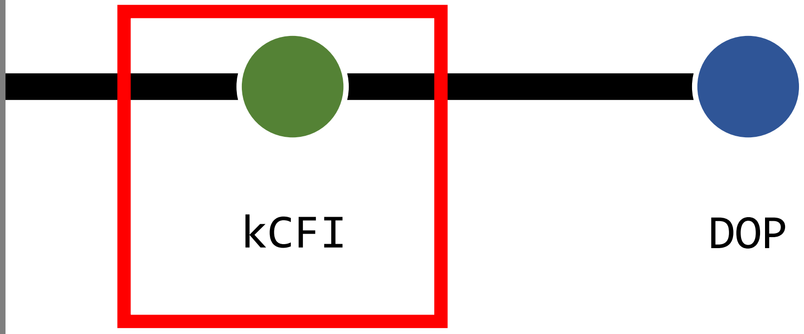
The history

- Kernel CFI

Restrict when control-flow change



In kernel

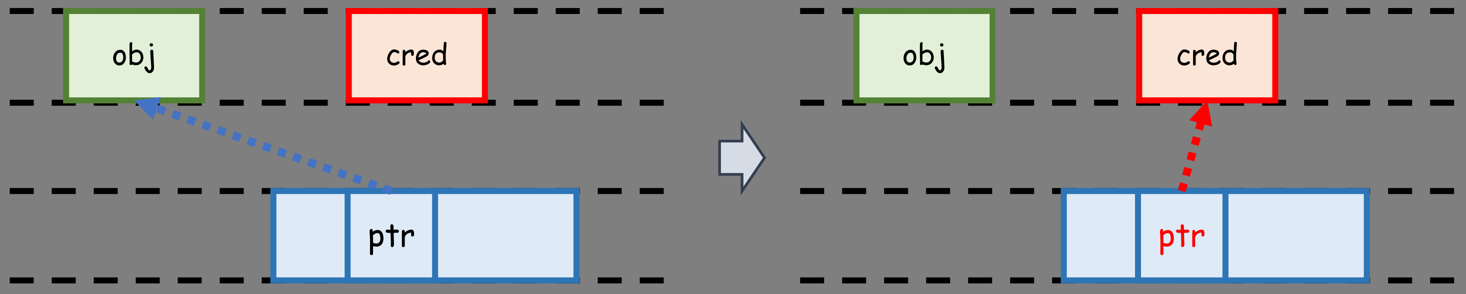


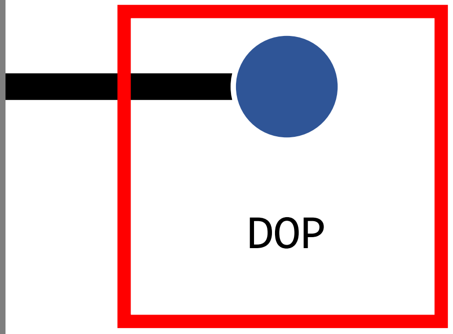
The history

In kernel

- DOP

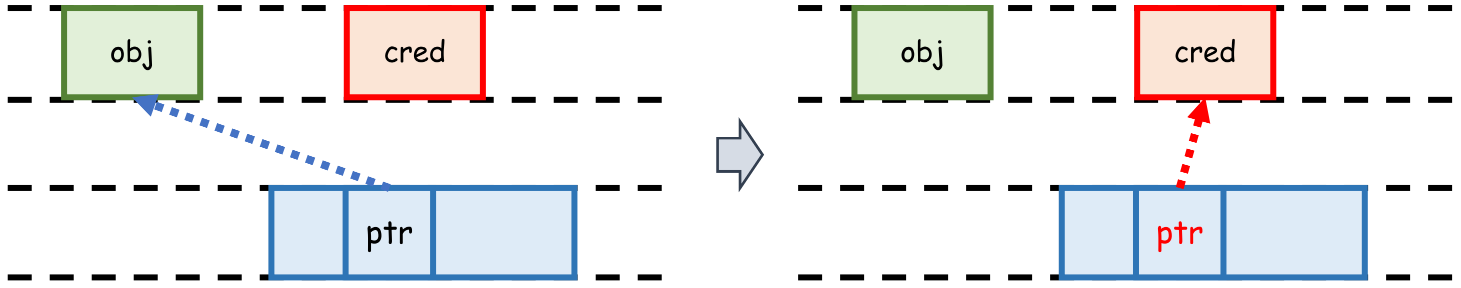
Data-Oriented Programming





Data-Oriented Programming

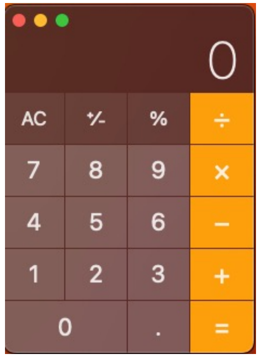
- Manipulate the data-flow to read/write a **target data**.
- That is, it has advantage when corrupting specific data.



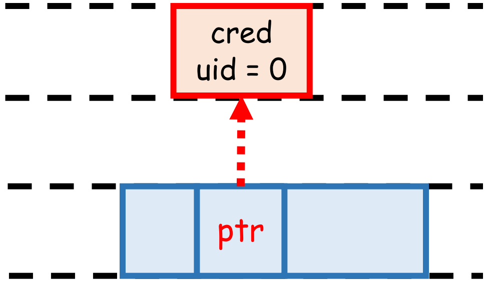
Strength of DOP

Specialized in kernel exploit

- DOP is effective not in User App but in Kernel



The goal of user application exploit

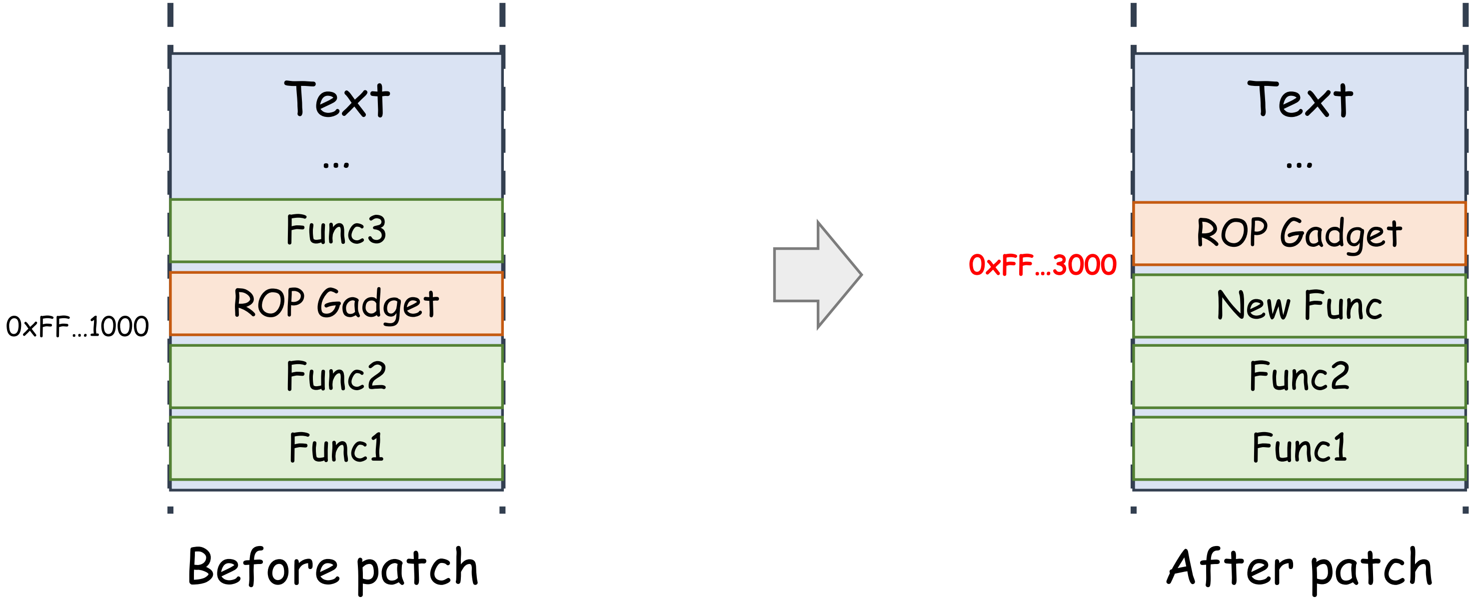


The goal of kernel exploit

Strength of DOP

Patch-agnostic exploits

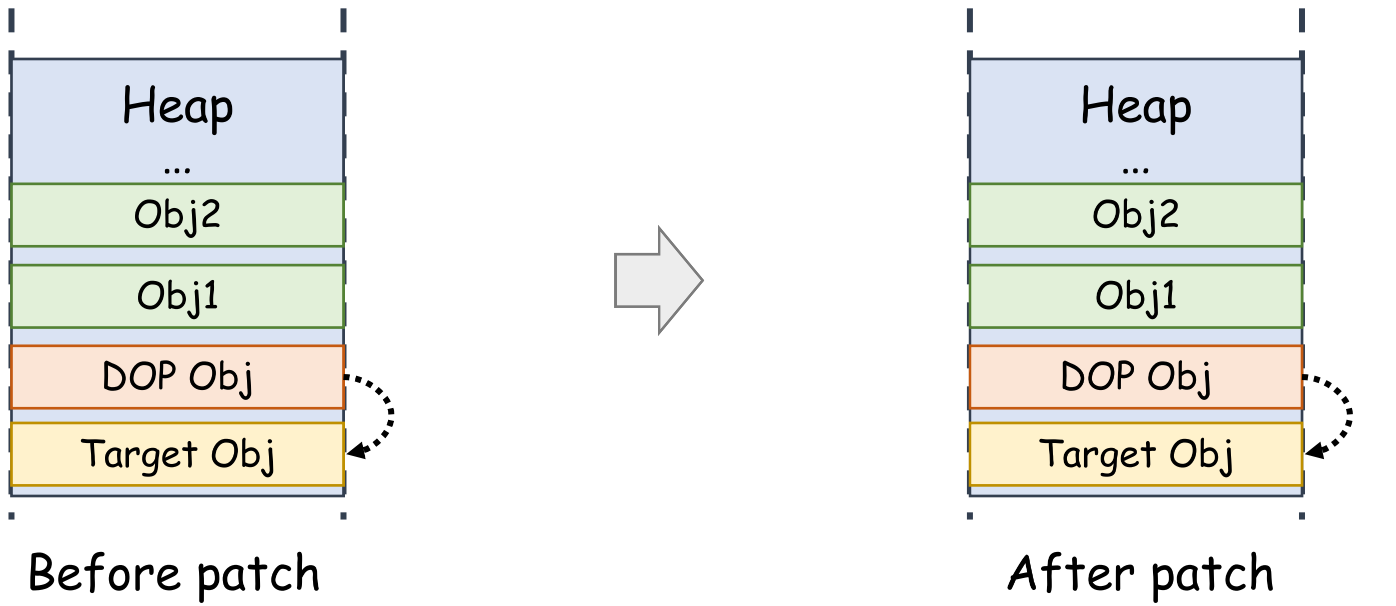
- ROP gadget is highly affected by the patch.
- Because the patch makes the offset of the ROP gadget **changes**.



Strength of DOP

Patch-agnostic exploits

- DOP is less affected by the patch.
- Unless the object used in the exploit is changed, the exploit hasn't changed.

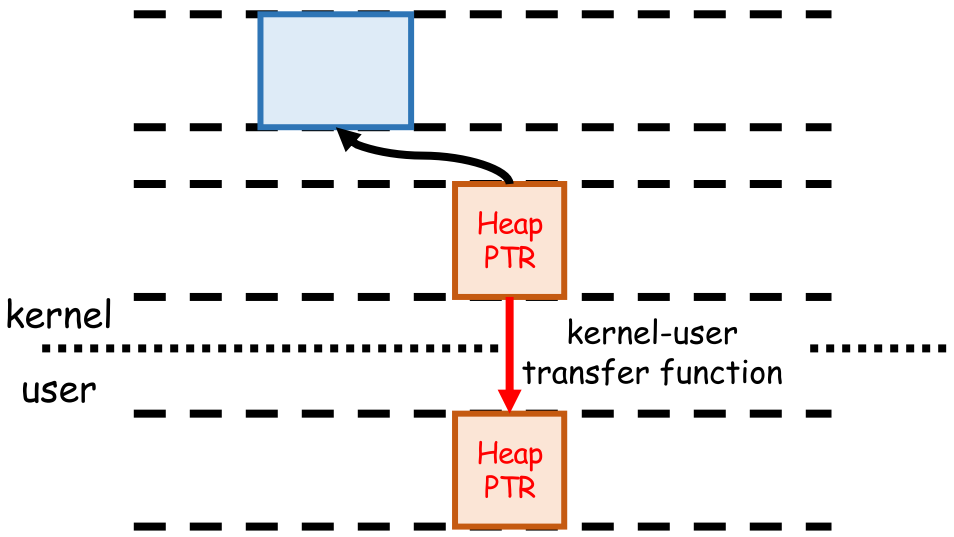


Requirements of DOP

- Privilege Escalation using DOP needs three exploit primitives.
  - Information Leakage
  - Arbitrary Address Read
  - Arbitrary Address Write

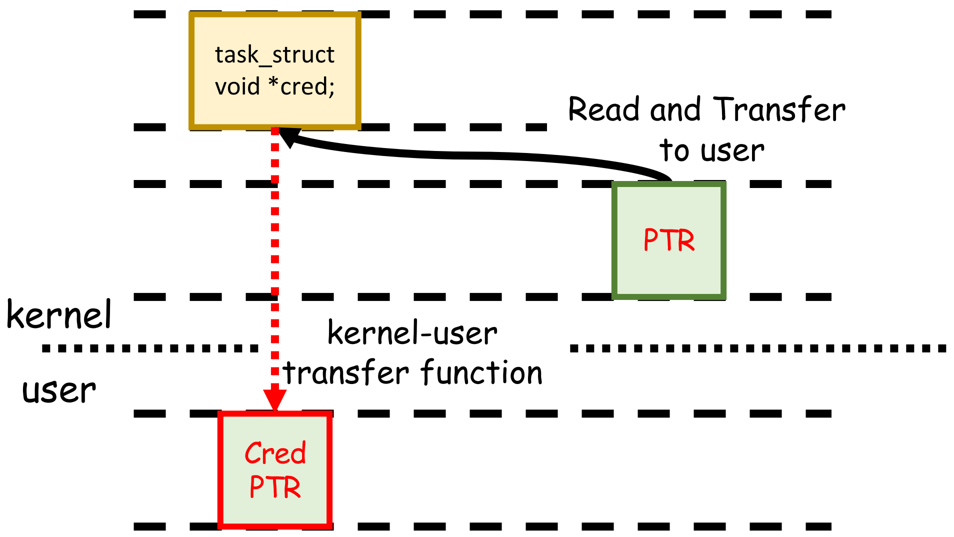
Requirements of DOP

- Privilege Escalation using DOP needs three exploit primitives.
  - **Information Leakage**
  - Arbitrary Address Read
  - Arbitrary Address Write



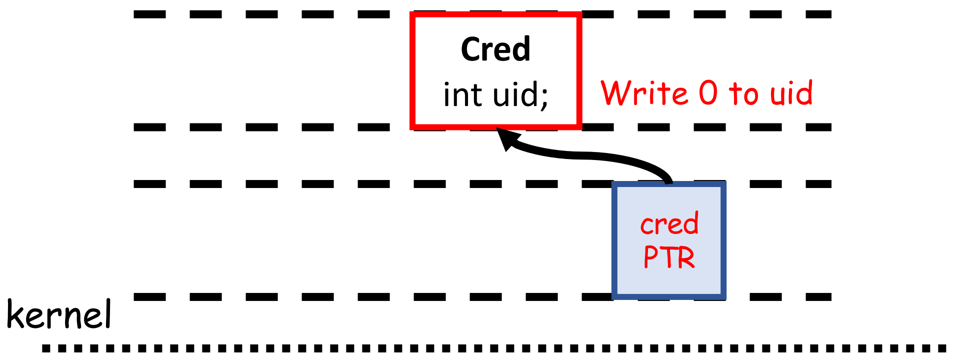
Requirements of DOP

- Privilege Escalation using DOP needs three exploit primitives.
  - **Information Leakage**
  - Arbitrary Address Read
  - Arbitrary Address Write



Requirements of DOP

- Privilege Escalation using DOP needs three exploit primitives.
  - **Information Leakage**
  - Arbitrary Address Read
  - Arbitrary Address Write



CVE-2021-31077

- One Heap Overflow
- Vulnerability Timeline
  - Found this vulnerability in late 2018
  - Exploit this vulnerability in 2020. 05
  - Report to the vendor in 2020. 05
  - Bug bounty reward in 2022. 06
  - Upload at patch note in 2023. 03

Attack Surface

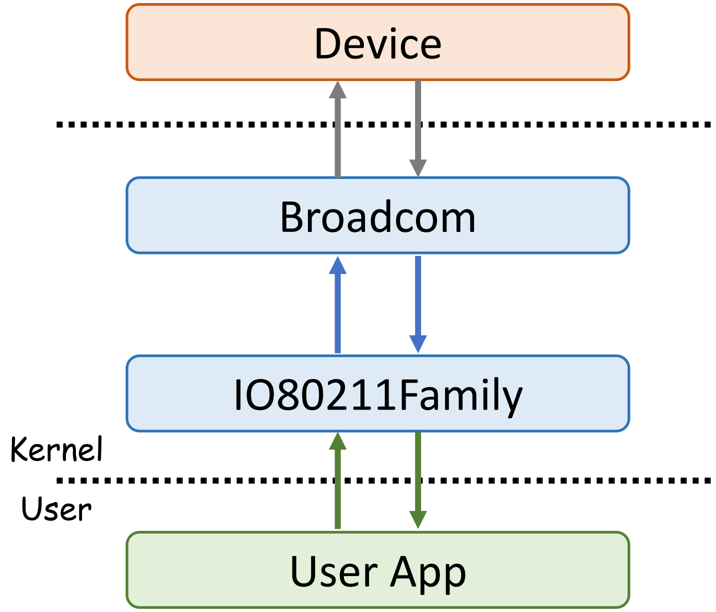
- IO80211, Broadcom

  - IO80211Family.kext
  - AirPort.BrcmNIC.kext

  x86 macOS

  - IO80211Family.kext
  - AppleBCMWLANCore.kext

  ARM macOS



Attack Surface

- IO80211, Broadcom
- Our attack surface is disclosed a very few times.

- ZDI blog in 2018
Based on my report

- BlackHat 2020 by Wang

Attack Surface

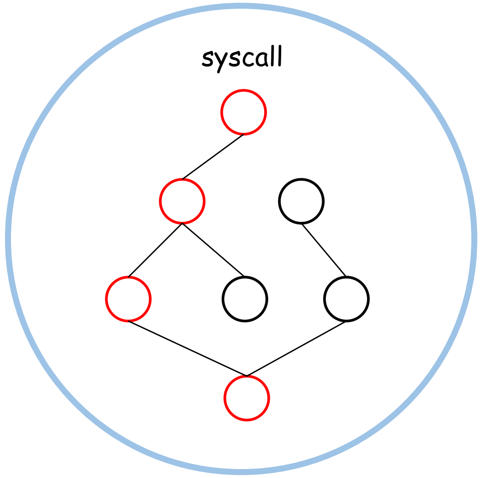
- IO80211, Broadcom
- We found a number of bugs and vulnerabilities.

  - CVE-2018-4084 : Information Leakage
  - CVE-2018-4338 : Information Leakage
  - CVE-2020-3839 : Information Leakage
  - CVE-2021-31077 : Local Privilege Escalation

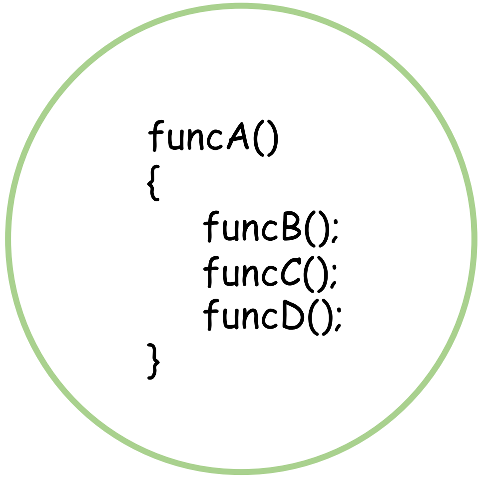
Reward about $40,000

Attack Surface

- How to know this module can be called by user.



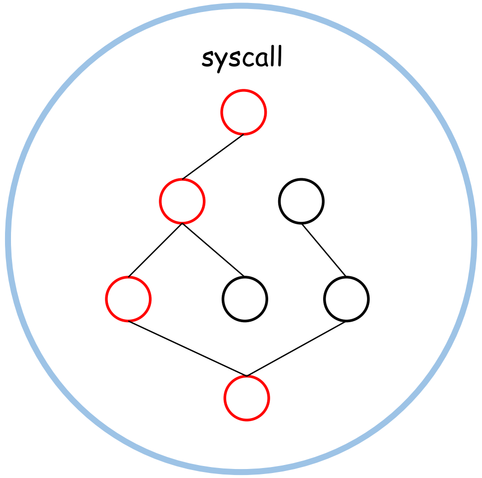
1) Tracking root function



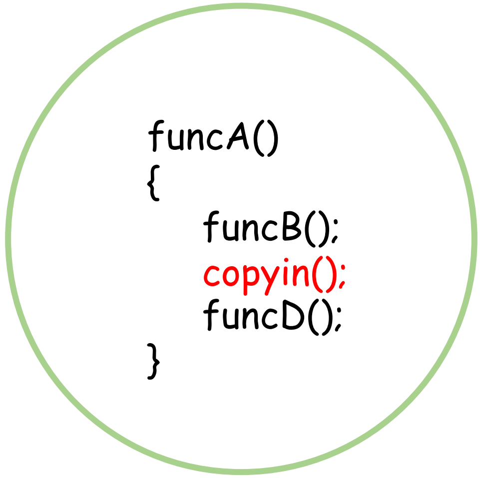
2) Finding the specific function

Attack Surface

- How to know this module can be called by user.



1) Tracking root function



2) Finding the specific function

Attack Surface

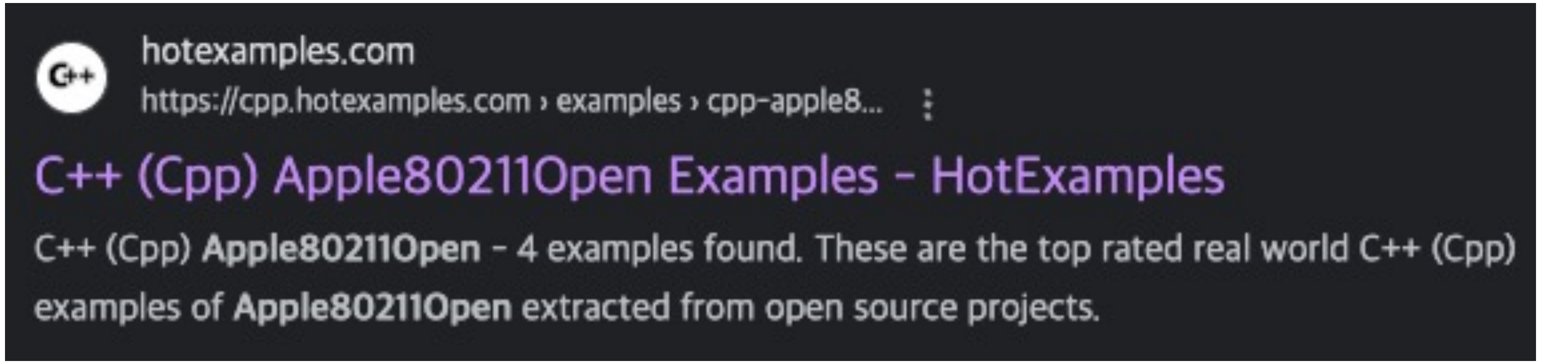
- How to know this module can be called by user.

```
funcA()
{
    funcB();
    copyin();
    funcD();
}
```

2) Finding the specific function

Attack Surface

- How to connect and trigger
  - Answer is in Google



CVE-2021-31077

- The kernel extension has two functions: setIE, getIE.
- Two functions can be called by ioctl().
- Two function treats storing and getting Information Element.
- The bug is triggered when executing getIE.
- However, to understand the bug, we have to understand the mechanism of setIE and getIE.

CVE-2021-31077

- setIE stores Information Element in vndr_ie.

```
int AirPort_BrcmNIC::setIE(a1, a2, apple80211_ie_data *input)
{
    uint8_t *ptr = osl_mallocz(*(a1 + 2528), 10000);
    ...
    strncpy_chk(ptr, "add", 4, 4);
    ptr[12] = input->data->id;
    memcpy(ptr+14, &input->data->len, input->ie_len-1);

    /* Point 0. this value is the key point of triggering overflow */
    ptr[13] = BYTE(input->ie_len-1);

    // store the buffer to "vndr_ie" variable
    err = wlIovarOp(a1, "vndr_ie", 0, 0, ptr, v18 + 14);
}
```

CVE-2021-31077

- *getIE in IO80211Familly allocates the heap buffer.*

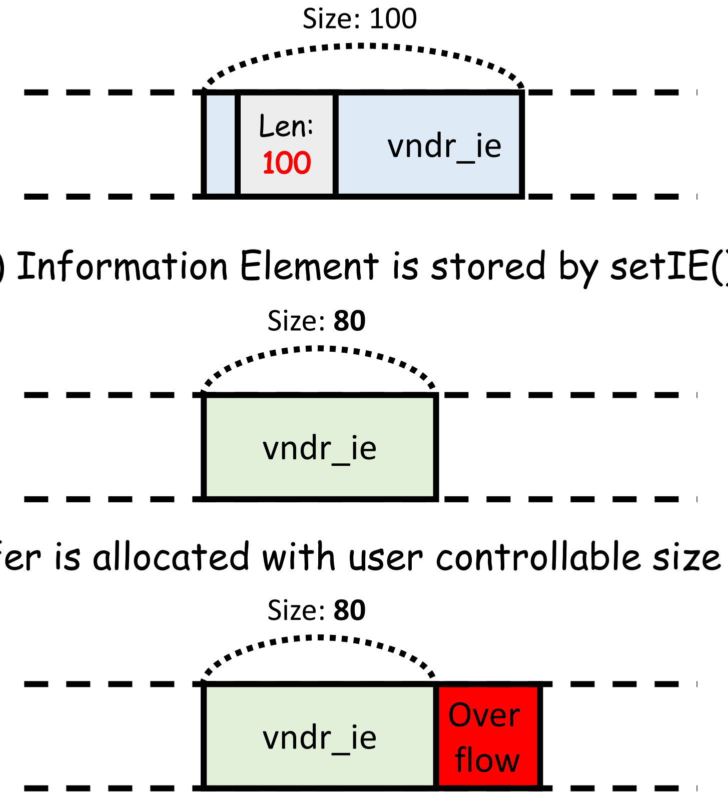
```
int getIE(a1, a2, a3, a4, input)
{
    struct apple80211_ie_data data;
    vndr_ie *ptr;
    copyIn(*(input + 32), &data, 0x20uLL);
    ...
    /* Point 1. allocate with size that user input */
    ptr = IOMalloc(data.ie_len);
    data.ie_data = ptr;
    ...
    // this function calls AirPort_BrcmNIC::getIE() internally.
    apple80211RequestIoctl(this, 0xC03069C9, 85, a2, &data);
    ...
    err = copyOut(&data, *(input + 32), 32);
    if(!err)
        copyOut(data.ie_data, user_ptr, data.ie_len);
}
```

CVE-2021-31077

- A heap overflow bug is triggered in AirPort_BrcmNIC::getIE.
  - input == allocated buffer & stored == stored buffer in setIE

```
int AirPort_BrcmNIC::getIE(a1, a2, apple80211_ie_data *input)
{
    ...
    void *ptr = osl_mallocz(*(a1 + 2528), 10000LL);
    // store the buffer to "vndr_ie" variable
    err = wlIovarOp(a1, "vndr_ie", 0LL, 0LL, ptr, 10000LL);
    vndr_ie *stored = ptr+8;
    ...
    /* Point 2. overflow will be occured when the size of input-
   >data is smaller than stored->len */
    memcpy(input->data + input->some_other_len, \
            &stored->data[0] + input->some_other_len, \
            stored->len - input->someotherlen + 2);
    input->ie_len = stored->len + 1;
}
```

CVE-2021-31077



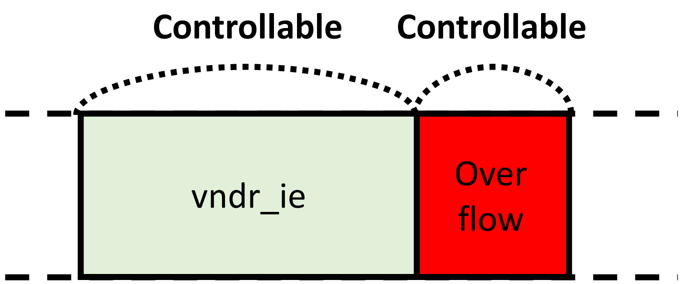
1) Information Element is stored by setIE().

2) The buffer is allocated with user controllable size by getIE().

3) The memory copy is triggered with a stored length size.
That is, if the allocated buffer's size is smaller than stored length,
the heap buffer overflow is triggered.

CVE-2021-31077

In summary, this vulnerability can control the size of buffer and the size of overflow.

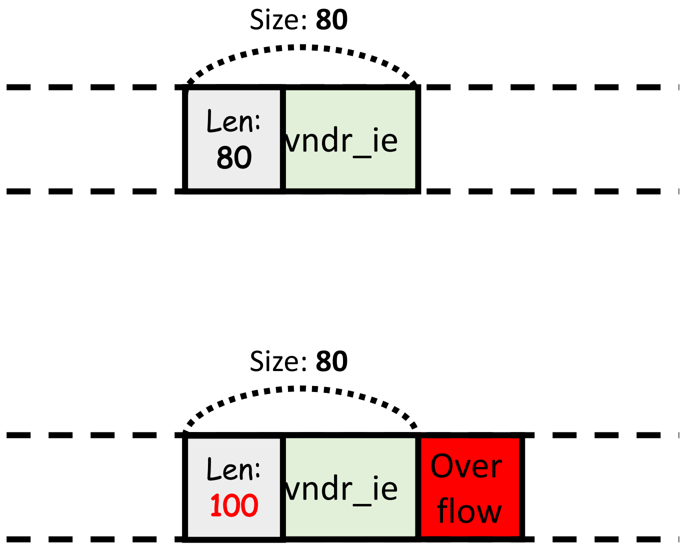


Exploit

- After overflow, kernel panic occurs because of hardened copy.
- This is because data.ie_len is overwritten to be larger than allocated.

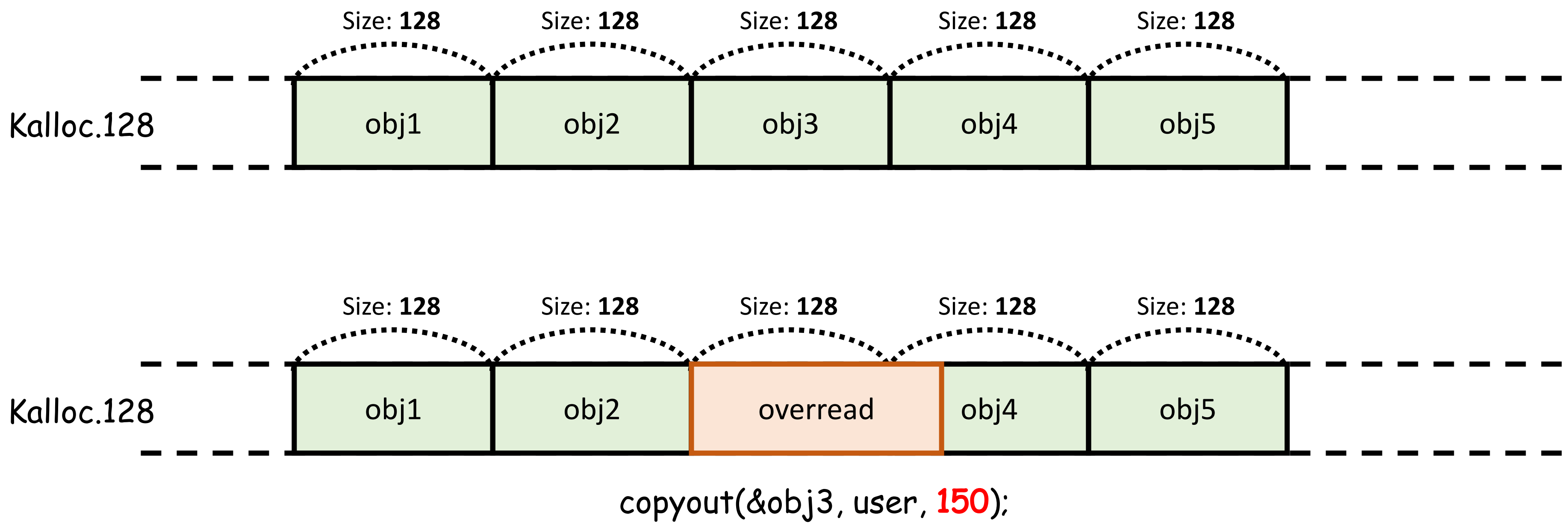
```
int getIE(a1, a2, a3, a4, input)
{
    // this function calls AirPort_BrcmNIC::getIE() internally.

    apple80211RequestIoctl(this, 0xC03069C9, 85, a2, &data);
    ...
    err = copyOut(&data, *(input + 32), 32);
    if(!err)
        copyOut(data.ie_data, user_ptr, data.ie_len);
}
```

e.g., 100



Hardened Copy

- It is a mitigation that prevents overread.
- If the copied size is bigger than the size of the object, it triggers kernel panic.



Exploit: Bypass Hardened Copy

- We thought the kernel panic by hardened copy must be triggered if the attacker tries to cause heap overflow.

```
int getIE(a1, a2, a3, a4, input)
{
    // this function calls AirPort_BrcmNIC::getIE() internally.

    apple80211RequestIoctl(this, 0xC03069C9, 85, a2, &data);
    ...
    err = copyOut(&data, *(input + 32), 32);
    if(!err)
        copyOut(data.ie_data, user_ptr, data.ie_len);
}
```

Exploit: Bypass Hardened Copy

- Here, we found a simple trick that prevents the second copyOut function.
- The second copyOut is not executed if the first copyOut is failed.

```
int getIE(a1, a2, a3, a4, input)
{
    // this function calls AirPort_BrcmNIC::getIE() internally.

    apple80211RequestIoctl(this, 0xC03069C9, 85, a2, &data);
    ...
    err = copyOut(&data, *(input + 32), 32);
    if(!err)
        copyOut(data.ie_data, user_ptr, data.ie_len);
}
```

Exploit: Bypass Hardened Copy

- copyOut function returns failed when it has a problem to copy the data to user space memory.
  - If the user space memory address is not assigned.
  - If the user space memory is **read-only**.
  - Etc.

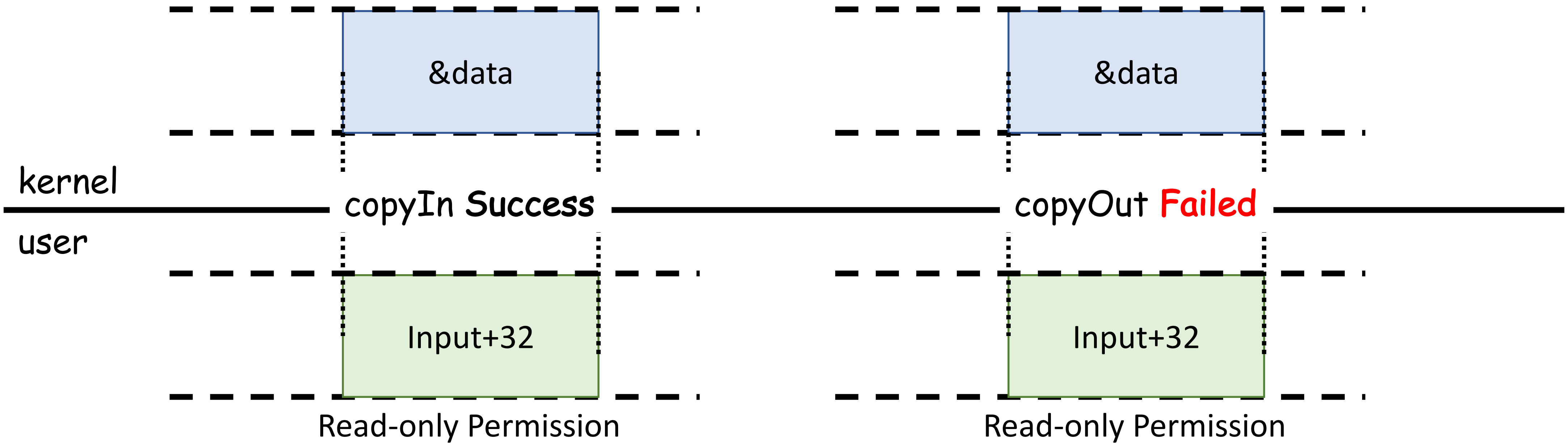
Exploit: Bypass Hardened Copy

- If the user space memory address is not assigned.

```
int getIE(a1, a2, a3, a4, input)
{
    struct apple80211_ie_data data;
    vndr_ie *ptr;
    copyIn(*(input + 32), &data, 0x20uLL);
    ...

                    Same user space memory

    ...
    err = copyOut(&data, *(input + 32), 32);
    if(!err)
        copyOut(data.ie_data, user_ptr, data.ie_len);
}
```

Exploit: Bypass Hardened Copy

- If the user space memory is **read-only**.



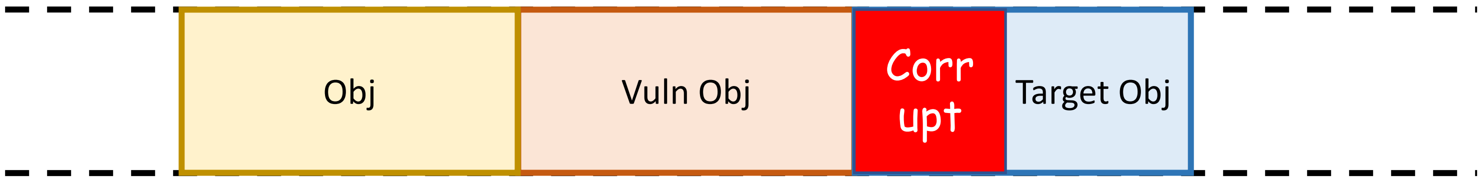
Exploit: Bypass Hardened Copy

- Since the trick makes first copyOut failed, the second copyOut is not executed.

```
int getIE(a1, a2, a3, a4, input)
{
    // this function calls AirPort_BrcmNIC::getIE() internally.

    apple80211RequestIoctl(this, 0xC03069C9, 85, a2, &data);
    ...
    err = copyOut(&data, *(input + 32), 32); // return fail
    if(!err) // goto else
        copyOut(data.ie_data, user_ptr, data.ie_len); // Panic!
}
```

Exploit

Now, we can corrupt the target object which is placed right after the vulnerable object.



Exploit

Target object : According to the target object, the vulnerability can be turned into various exploit primitives.

DOP needs 3 primitives: Information Leakage, Arbitrary Address Read, Arbitrary Address Write

Exploit

Target object : According to the target object, the vulnerability can be turned into various exploit primitives.

Tuesday, December 10, 2019

SockPuppet: A Walkthrough of a Kernel Exploit for iOS 12.4

Posted by Ned Williamson, 20% on Project Zero

Exploit

Target object : According to the target object, the vulnerability can be turned into various exploit primitives.

```
struct  ip6_pktopts {
  struct  mbuf *ip6po_m;
  int     ip6po_hlim;
  struct  in6_pktinfo *ip6po_pktinfo;
  struct  ip6po_nhinfo ip6po_nhinfo;
  struct  ip6_hbh *ip6po_hbh;
  struct  ip6_dest *ip6po_dest1;
  struct  ip6po_rhinfo ip6po_rhinfo;
  struct  ip6_dest *ip6po_dest2;
  int     ip6po_tclass;
  int     ip6po_minmtu;
  int     ip6po_prefer_tempaddr;
  int ip6po_flags;
};
```

```
struct in6_pktinfo {
  struct in6_addr ipi6_addr;
  unsigned int ipi6_ifindex;
};
```

Exploit

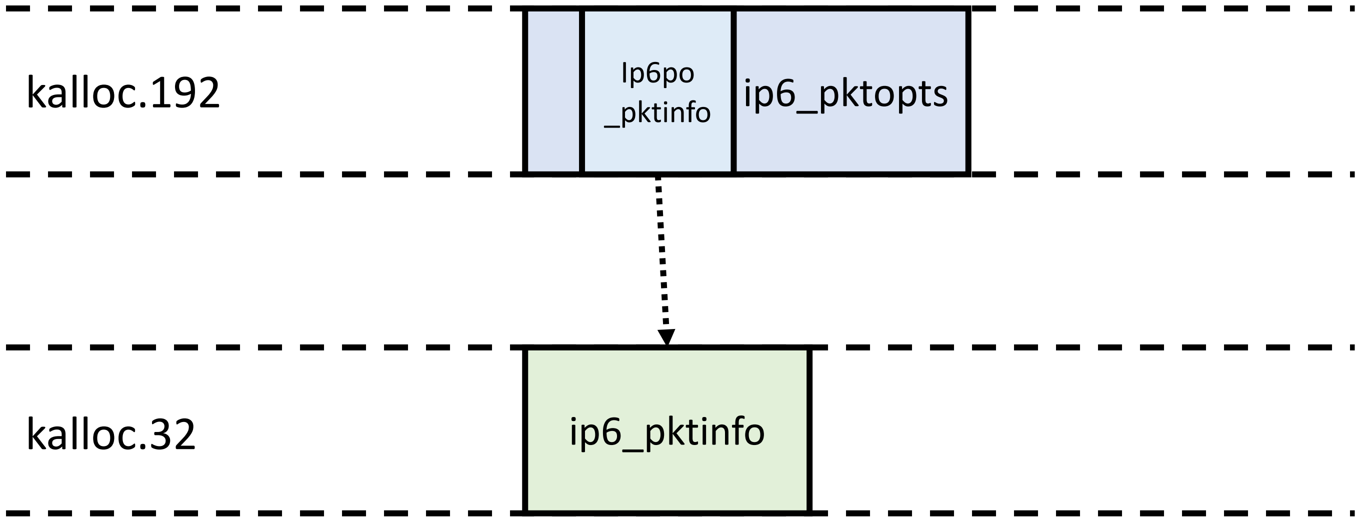
We only use ip6po_pktinfo for Information Leakage, AAR, AAW

```
struct  ip6_pktopts {
  struct  mbuf *ip6po_m;
  int     ip6po_hlim;
  struct  in6_pktinfo *ip6po_pktinfo;
  struct  ip6po_nhinfo ip6po_nhinfo;
  struct  ip6_hbh *ip6po_hbh;
  struct  ip6_dest *ip6po_dest1;
  struct  ip6po_rhinfo ip6po_rhinfo;
  struct  ip6_dest *ip6po_dest2;
  int     ip6po_tclass;
  int     ip6po_minmtu;
  int     ip6po_prefer_tempaddr;
  int ip6po_flags;
};
```

```
struct in6_pktinfo {
  struct in6_addr ipi6_addr;
  unsigned int ipi6_ifindex;
};
```

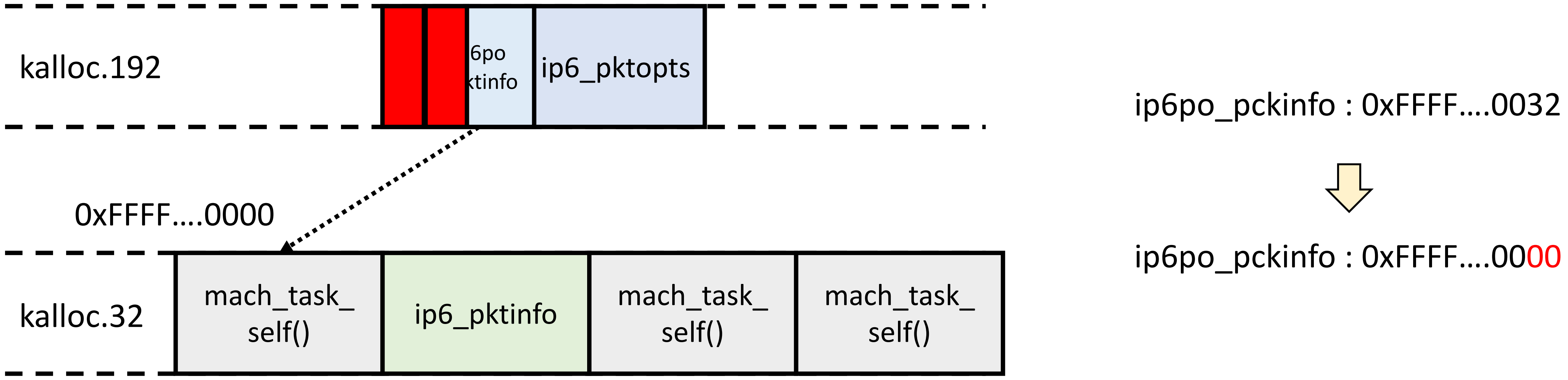
Exploit

Zone



Exploit

Information Leakage



Exploit

Recursive Arbitrary Address Read

```
struct ipc_port {
        ...
    union {
        ipc_kobject_t kobject;       // OFFSET 0x68  KOBJECT == struct task
        ipc_importance_task_t imp_task;
        ipc_port_t sync_inheritor_port;
        struct knote *sync_inheritor_knote;
        struct turnstile *sync_inheritor_ts;
    } kdata;
       ...
}
```

Exploit

Recursive Arbitrary Address Read

```
struct task {
        ...
        void *bsd_info;                // OFFSET 0x3a0 bsd_info == struct proc
        ...
}
```

Exploit

Recursive Arbitrary Address Read

```
struct proc {
        ...
        kauth_cred_t    p_ucred;     // OFFSET 0x100 p_ucred == struct ucred
        ...
}
```

Exploit

Arbitrary Address Write

```
struct ucred {
    ...
    struct posix_cred {
        uid_t   cr_uid;         /* effective user id */   // OFFSET 0x18
        uid_t   cr_ruid;        /* real user id */
        uid_t   cr_svuid;       /* saved user id */
        short   cr_ngroups;     /* number of groups in advisory list */
        gid_t   cr_groups[NGROUPS];/* advisory group list */
        gid_t   cr_rgid;        /* real group id */
        gid_t   cr_svgid;       /* saved group id */
        uid_t   cr_gmuid;       /* UID for group membership purposes */
        int     cr_flags;       /* flags on credential */
    } cr_posix;
    ...
};
```

Exploit

Proof-Of-Concept

Email: yoochan10@snu.ac.kr
Twitter: @_yoochanlee