

[Become a reader](#)

I'm gonna CTF.

I'll write anything other than CTF.

Structure collections for Kernel Exploit


Exploit

At first

It's been about two months since I first studied Kernel Exploit. What I'm doing is that kernel land UAF, heap overflow, etc. are very powerful. However, in addition to the features provided by vulnerable drivers, they often take advantage of the features provided by the kernel and other drivers, so without a lot of knowledge, they can't attack well or have a lower probability of success. For example, KUAF is very useful, but if the size of the object to be UAF is fixed, you need to find an object that matches the size and is "usable" in kmalloc. In this article, I'm going to put together such a "using" object. However, I'm an amateur when it comes to Kernel Exploit, so I may make a lot of mistakes, and I may have missed something like a fixed stone. Please tell us about the wrong part or if there are other objects you know in comments or on Twitter.

The experimental environment and code you write are summarized in the following repositories:

Bitbucket

 bitbucket.org

bitbucket.org

Linux Kernel 4.19.98 was used for verification.

- [At first](#)
- [List of structures that can be used for Leak/AAR/AAW/RIP control](#)
 - [shm_file_data](#)
 - [seq_operations](#)
 - [msg_msg_\(+user-supplied data\)](#)
 - [subprocess_info](#)
 - [cred](#)
 - [file](#)
 - [timerfd_ctx](#)
 - [tty_struct](#)
- [Structures that can be used for arbitrary data writes/Heap Sprays](#)
 - [msg_msg](#)
 - [setxattr](#)
 - [sendmsg](#)
- [bibliography](#)

List of structures that can be used for Leak/AAR/AAW/RIP control

There are rotten structures, but I want to cover what can be used for each size anyway. Please let me know if you know below. Currently kcalloc-8, kcalloc-16, kcalloc-64, per kcalloc-512 are unexplored.

shm_file_data

.....

- Size: 0x20 (kcalloc-32)

- Base:, points to the kernel data area and can be leaked. `ns` `vm_ops`
- Heap: can be leaked because it points to heap space. `file`
- stack: Cannot leak.
- RIP: I probably can't take it.
- Reserve: Map shared memory in . `shmat`
- Release: Or do you want to destroy it? `shmctl`
- Note: I tried rewriting, but there was no particular appearance that a function pointer of a fake vtable was called by etc. `vm_ops` `shmget`
- Reference: <https://elixir.bootlin.com/linux/v4.19.98/source/ipc/shm.c#L74>

```
0x0000: 0x0000000000000000
0x0008: 0xfffffffff82292ae0
0x0010: 0xfffff88800ea09700
0x0018: 0xfffffffff81e15540
[+] kbase = 0xfffffffff81000000
[+] kheap = 0xfffff88800ea09700
```

seq_operations

- Size: 0x20 (kmalloc-32)
- base: Leakable using whatever you like from 4 function pointers.
- heap: Cannot leak.
- stack: Cannot leak.
- RIP: For example, if you rewrite and call read, RIP will be a pong! It becomes. `start`
- Secure: Open the file that uses . And such? `single_open` `/proc/self/stat`
- Release: To. `close`
- Reference: https://elixir.bootlin.com/linux/v4.19.98/source/include/linux/seq_file.h#L32

```
0x0000: 0xfffffffff811c5f70
0x0008: 0xfffffffff811c5f90
0x0010: 0xfffffffff811c5f80
0x0018: 0xfffffffff8120c3f0
[+] kbase = 0xfffffffff81000000
Press enter to continue...
[ 6.801190] BUG: unable to handle kernel paging request at
00000000deadbeef
```

msg_msg (+user-supplied data)

- Size: 0x31 to 0x1000 (kmalloc-64 or higher)
- base: Cannot leak.
- Leap: can be leaked because it points to a previously msgsnd message. Useful because it points to an address on SLUB that corresponds to the size of the previously msgsnd data. `next`
- stack: Cannot leak.
- RIP: I can't take it.
- Secure: And so on. `msgget` `msgsnd`
- Release: To. However, since it is received in order from the first one, the order of kfree is also the order in which it was sent. `msgrcv` `msgsnd`
- Note: It is convenient because arbitrary data can be written by 4 sizes, but the first 48 bytes are used in the structure, so it cannot be rewritten. When read is only possible in UAF, it is possible to prepare data on the heap after leaking the address of the heap. It is often used for heap spray.
- Reference: <https://elixir.bootlin.com/linux/v4.19.98/source/include/linux/msg.h#L9>

```

0x0000: 0xfffff88800e1661c0
0x0008: 0xfffff88800e1661c0
0x0010: 0x0000000000000001
0x0018: 0x0000000000000020
0x0020: 0x0000000000000000
0x0028: 0xfffff88800e8c7f90
0x0030: 0x4242424241414141
0x0038: 0x4444444443434343
0x0040: 0x4646464645454545
0x0048: 0x0000000000000046
[+] kheap = 0xfffff88800e1661c0

```

subprocess_info

- Size: 0x60 (kmalloc-128)
- base : がを指しているのでリーク可能。
`work.func` `call_usermodehelper_exec_work`
- heap : リーク可能だがどのSLUBのデータかは未検証。
- stack : リークできない。
- RIP: With race condition, etc., RIP can be taken by rewriting while securing. (Isn't this trivia?))
`cleanup`

- Secure: A method of specifying a protocol that seems to have been confirmed but is not known. `socket(22, AF_INET, 0);`
- Release: Freed with the same path as securing.
- Remarks: I thought that I could not write data and could use it, but since there is time until after setting up, verification succeeded in taking RIP with a high probability with race condition. However, for some reason, rop also dies in places where fs is used and syscall after returning to userland. If SMAP is disabled, I think it is possible to rewrite fd to userland and combine it with userfaultfd to overwrite cleanup. `info->cleanup if (info->cleanup) info->cleanup(info);`
- Reference: <https://elixir.bootlin.com/linux/v4.19.98/source/include/linux/umh.h#L19>

```

0x0000: 0xfffff88800f254380
0x0008: 0xfffff88800f254e88
0x0010: 0xfffff88800f254e88
0x0018: 0xfffffffff81071380
0x0020: 0x0000000000000000
0x0028: 0xfffffffff82242260
0x0030: 0xfffff88800e0e3b40
0x0038: 0xfffffffff82242180
0x0040: 0x0000000000000000
0x0048: 0x0000010000000006
0x0050: 0x00000000000000407
0x0058: 0x0000000000000000
[+] kbase = 0xfffffffff81000000
[+] kheap = 0xfffff88800f254380
Press enter to continue...
[ 6.801190] BUG: unable to handle kernel paging request at
00000000deadbeef

```

cred

.....

- Size: 0xa8 (kmalloc-192)
- base: It seems not to be leaked.
- heap: leakable from etc. The target SLUB has not been investigated. `session_keyring`
- stack: It seems that it can't be leaked.
- RIP: I can't take it.
- Secure: fork.
- Release: Exit the process you made.
- Note: Since it has uid, gid, etc., it can be promoted to authority if it can be filled with 0.

However, I feel that it is not in `kmalloc` in 4.19.98 verified. `cred`

- Reference: <https://elixir.bootlin.com/linux/v4.19.98/source/include/linux/cred.h#L116>

file

- size:? (`kmalloc-256`)
- Base: points to the kernel data area and can be leaked. `f_op`
- heap: Unt validation. Well, I can go.
- stack: Unt validated.
- Reserve: Create shared memory in . `shmget`
- Release: Discard in . `shmctl`
- Remarks: RIP control is possible by rewriting and calling etc. However, since the file structure did not overlap for some reason after UAF, verification failed. `f_op` `shmctl`
- Reference: <https://elixir.bootlin.com/linux/v4.19.98/source/include/linux/fs.h#L891>

timerfd_ctx

- size:? (`kmalloc-256`)
- Base: can be leaked because it points to . `tmr.function` `timerfd_tmrproc`
- Heap: Can be leaked from, etc. `tmr.base`
- stack: Cannot leak.
- RIP: I can't take it.
- Secure: Call. `timerfd_create`
- Release: `tfd?` `close`
- Reference: <https://elixir.bootlin.com/linux/v4.19.98/source/fs/timerfd.c#L30>

```
0x0000: 0xfffff88800e216100
0x0008: 0x00000000000000000
0x0010: 0x00000000000000000
0x0018: 0x0000000183ca77938
0x0020: 0x0000000183ca77938
0x0028: 0xfffffffff811e7ef0
0x0030: 0xfffff88800f41ba80
...
[+] kbase = 0xfffffffff81000000
[+] kbase = 0xfffff88800f41ba80
```

tty_struct

- Size: 0x2e0 (kmalloc-1024)
- Base: points to , so it can be leaked. In addition, it pointed to the data region of kernel in about two places. ops ptm_unix98_ops
- Heap:, and so many objects point to the heap or its own members, so they can be leaked. The target SLUB has not been investigated. dev driver
- stack: It seems that it can't be leaked.
- Secure: Open . /dev/ptmx
- Release: Close the open ptmx.
- Remarks: RIP can be controlled by rewriting. ops
- Reference: <https://elixir.bootlin.com/linux/v4.19.98/source/include/linux/tty.h#L283>

```

0x0000: 0x00000000100005401
0x0008: 0x0000000000000000
0x0010: 0xfffff88800f1ed840
0x0018: 0xfffffffff81e65900
0x0020: 0x0000000000000000
...
[+] kbase = 0xfffffffff81000000
[+] kheap = 0xfffff88800f1ed840
Press enter to continue...
[ 5.413411] BUG: unable to handle kernel paging request at
00000000deadbeef

```

Structures that can be used for arbitrary data writes/Heap Sprays

msg_msg

Explained

setxattr

- Size: Optional (< = 65536)
- Secure: Call. Put a pointer to the value you want to write to and specify . setxattr value size
- Release: Freed with the same path as securing.
- Note: Used in combination with userfaultfd. Msgsnd does not rewrite the first 48 bytes, so it is useful

if you want to support it. It can also be used for heap spraying.

sendmsg

- Size: Optional (≥ 2)
- Secure: Call. pointer, put the size
in. `sendmsg` `msg.msg_control` `msg.msg_controllen`
- Release: Freed with the same path as securing.
- Note: Same as `setxattr` and combined with `userfaultfd`.

bibliography

- [1] <https://elixir.bootlin.com/linux/v4.19.98/source>
- [2] <https://duasynt.com/blog/linux-kernel-heap-spray>
- [3] <https://hama.hatenadiary.jp/entry/2018/12/01/000000>
- [4] <https://smallkirby.hatenablog.com/entry/2019/11/19/225504>



Write a comment

[« SuSeC CTF 2020 Pwn Writeups](#)

[CONFidence CTF 2020 Teaser Writeups »](#)

profile



sushi

[Become a reader](#)

69

search

Monthly Archives

- ▶ 2021 (12)
- ▼ 2020 (43)
 - 2020 / 12 (5)
 - 2020 / 11 (2)
 - 2020 / 10 (1)
 - 2020 / 9 (4)
 - 2020 / 8 (2)
 - 2020 / 7 (4)
 - 2020 / 6 (2)
 - 2020 / 5 (2)
 - 2020 / 4 (6)
 - 2020 / 3 (7)
 - 2020 / 2 (3)
 - 2020 / 1 (5)
- ▶ 2019 (55)
- ▶ 2018 (9)

Let's start a ythysy blog!

ptr-yudai uses a ythysyn blog. Why don't you start a Yythysner blog?

Start a Ythythnables Blog (Free)

[What is a Ythythy world blog?](#)



[CTFするぞ](#)

Powered by [Hatena Blog](#) | [Report a blog](#)