CVE-2021-22555 Analysis Report

Analysis report on Local Privilege Escalation
Caused by Out-of-bounds in the Linux Netfilter module

- First Author : Florian Westphal (fw@strlen.de)
- Author : iCAROS7

- Data Created : 2022.07.18 Mon
- Data Version : 1.0.0

Index

1. Introduce

CVE-2021-22555는 user namespace 내에서 contraning 되어있는 application이 heap aera Out-of-bounds (이하 oob)를 통해 privilege escalation이 가능한 exploit 이다.

Linux kernel의 network 관련 handler type의 framework인 `netfilter` 내에서 trigger 된다. 2006년경 modified 된 `x_tables` module 내 `xt_compat_match`에서 type-safe 의 목적으로 split 된 function으로부터 시작 된다. `xt_compat_match_from_user` 및 `xt_compat_target_from_user` function내 integer 형으로 선언 된 incompleteness한 `pad` variable 의하여 induce 된다.

Linux kernel version ~ 4.4.265, ~ 4.9.265, ~ 4.14.229, ~ 4.19.186, ~ 5.4.111, ~ 5.10.29, ~ 5.12 까지의 광범위한 범위를 지닌다.

2021년 4월 6일 `Florian Westphal`에 의해 최초 보고 되었으며, 동년 동월 13일 Linux main upstream branch에 patch가 merge되며 각 vendor에게 release 되었다. 이후 동년 7월 7일 public release 되며 CVSS 3.0 기준 `7.8 High`로 scoring 되었다.

2. Analysis of crash occurrence function

2-1. Basic knowledge

2-1-1. `netfilter`

2-1-2. Access to heap aera

2-1-3. Out-of-bound of heap aera

2-2. Code audit

```
# /net/netfilter/x_tables.c

void xt_compat_match_from_user(struct xt_entry_match *m, void **dstptr,
int *size)
{
    struct xt_match *match = m->u.kernel.match;              // [1]
    struct compat_xt_entry_match *cm = (struct compat_xt_entry_match *)m;
// [2]
    int pad, off = xt_compat_match_offset(match);               // [3]
    u_int16_t msize = cm->u.user.match_size;                 // [4]

    m = *dstptr;                          // [5]
    memcpy(m, cm, sizeof(*cm));              // [6]
    if (match->compat_from_user)                 // [7]
        match->compat_from_user(m->data, cm->data);
    else
        memcpy(m->data, cm->data, msize - sizeof(*cm));
    pad = XT_ALIGN(match->matchsize) - match->matchsize;    // [8]
    if (pad > 0)                          // [9]
        memset(m->data + match->matchsize, 0, pad);

    msize += off;                         // [10]
    m->u.user.match_size = msize;                // [11]

    *size += off;                         // [12]
    *dstptr += msize;                     // [13]
}
```

우선 [1]과 같이 argument로 넘어온 `xt_entry_match` struct를 User Kernel Match data로 pointer 시킨다. [2]에서 `compat_xt_entry_match` type struct pointer `cm`에 arguments의 `m`을 `compat_xt_entry_match`으로 type conversion 하여 대입한다. 이후 [3]에서 integer type의 `pad`를 선언하며 `off`를 `xt_compat_match_offset()` 의 return data로 `compatsize`의 value에 따라 (`matchsize` + `__alignof__` - 1) -`compatsize`의 연산을 통해 allocating 한다. 여기서 `__alignof__`로 define 된 value는 amd64 archtecture 에 따라 8 byte 이므로 `matchsize + 7 -` compatsize`가 대입된다.

[4]에서 unsigned 16bit integer type `msize`에 argument의 user.match_size를 가져온 이후, `m` pointer 의 address 를 argument의 `dstptr`로 change 한다. [6]과 같이 `memcpy` function을 통해 `dstptr`의 data를 다시 `cm`의 address 로 `cm`의 size 만큼 copy 한다.

[7]의 if 문에서 kernel space 상에서 별도의 align이 필요할 경우 call & use 되며 Kernel config 상의 `CONFIG_COMPAT`에 따라 존재가 결정되는 `compat_from_user`가 존재한다면 해당 function을, 존재하지 않는 다면 단순 `memcpy`로 `cm` struct의 data를 `m` struct의 `data`로 copy 한다.

이후 [8]에서 `pad` variable에 위에서 연산이 한번 되었던 `XT_ALIGN(matchsize)`와 `matchsize`를 subtraction한다.

이때 [9]의 if 문의 condition에 따라 [8]의 `pad` 값이 0보다 큰 경우에는 `memset` function을 통해 `m` struct의 `data` address 와 `match` struct의 `match` address의 합에 `0`을 `pad` value 만큼 memory를 initalization 한다.

[10]에서 `msize`의 값을 [3]에서 계산된 `off` value 만큼 increase 시킨 뒤, [11]에서 user.match_size를 `msize`에 대입한다. 이후 [12]에서 argument의 `size` pointer의 값 역시 `off` value 만큼 increase 시키고, [13]에서 `dstptr` pointer 역시 `msize` 만큼 increase 한다.

```
void xt_compat_target_from_user(struct xt_entry_target *t, void **dstptr,
                unsigned int *size)
{
    const struct xt_target *target = t->u.kernel.target;
    struct compat_xt_entry_target *ct = (struct compat_xt_entry_target
*)t;
    int pad, off = xt_compat_target_offset(target);
    u_int16_t tsize = ct->u.user.target_size;
    char name[sizeof(t->u.user.name)];          // [1]

    t = *dstptr;
    memcpy(t, ct, sizeof(*ct));
    if (target->compat_from_user)
        target->compat_from_user(t->data, ct->data);
    else
        memcpy(t->data, ct->data, tsize - sizeof(*ct));
    pad = XT_ALIGN(target->targetsize) - target->targetsize;
    if (pad > 0)
        memset(t->data + target->targetsize, 0, pad);

    tsize += off;
    t->u.user.target_size = tsize;

    strlcpy(name, target->name, sizeof(name));      // [2]
    module_put(target->me);                 // [3]
    strncpy(t->u.user.name, name, sizeof(t->u.user.name));  // [4]
    *size += off;
    *dstptr += tsize;
}
```

```
Temp Code Aera for [3]
void module_put(struct module *module)
{
    int ret;

    if (module) {
```

```
        preempt_disable();
        ret = atomic_dec_if_positive(&module->refcnt);
        WARN_ON(ret < 0);   /* Failed to put refcount */
        trace_module_put(module, _RET_IP_);
        preempt_enable();
    }
}
```

`xt_compat_target_from_user` 역시 `xt_compat_match_from_user`와 비슷한 mechanism으로 동작하지만 차이점은 다음과 같다.

[1]에서 char 형식으로 `user.name`의 size 만큼 array를 구성한다. 이후 `memset` function으로 memory initalization 까지 동일하게 진행된다.

[2] `strlcpy` function으로 `\0`을 포함하여 `target` struct의 `name`을 [1]에서 declaration 된 `name`에 copy 한다. 이후 [3]에서

[4]에서 다시금 `strncpy` function ^왜 strlcpy를 안쓰는가?^ 을 통해 user.name에 `name`의 address에 u.user.name 의 size만큼 copy 한다.

2-3. Vulnerability analysis

실질적으로 vulnerability 한 code는 하기와 같다.

```
# xt_compat_match_from_user
    int pad, off = xt_compat_match_offset(match);

    pad = XT_ALIGN(match->matchsize) - match->matchsize;
    if (pad > 0)
        memset(m->data + match->matchsize, 0, pad);

# xt_compat_target_from_user
    int pad, off = xt_compat_target_offset(target);

    pad = XT_ALIGN(target->targetsize) - target->targetsize;
    if (pad > 0)
        memset(t->data + target->targetsize, 0, pad);
```

Integer type으로 declaration 된 `pad` variable에 `XT_ALIGN` macro를 통해 [code audit의 xt_compat_match_from_user [8]

```
Temp Area for vulnerability
이후 [3]에서 integer type의 `pad`를 선언하며 `off`를 `xt_compat_match_offset()`
의 return data로 `compatsize`의 value에 따라 (`matchsize` + `__alignof__` - 1)
-`compatsize`의 연산을 통해 allocating 한다. 여기서 `__alignof__`로 define 된
value는 amd64 archtecture에 따라 8 byte 이므로 `matchsize + 7 - `compatsize`가
대입된다.
```

3. Crash-inducement

3-1. Host PC specification

- OS Version : Ubuntu 20.04.04 LTS (Focal Fossa)
- Kernel Version: Linux 5.13.0-52-generic amd64
- CPU : Intel(R) Core(TM) i7-2620M CPU @ 2.70GHz
- RAM : PC3L-12800 1600MHz 12GB
- Swap File Size: 16GB

3-2. Fuzzer specification

- Kernel version : Linux 5.10.28-generic
- Fuzzer : Syzkaller revision 1434eec0 (Coverage-guided kernel fuzzer)
- Enabled Sanitizer :
    1. KASAN (Kernel Address Sanitizer)
    2. UBSAN (Undefined Behavior Sanitizer) KCOV (Kernel Coverage)
    3. Kmemleak (Kernel Memory Leak Detector)

3-3. Vulnerability analysis

4. Proof of concept

5. Conclusion

6. Reference

- https://www.cvedetails.com/cve/CVE-2021-22555/
- https://ubuntu.com/security/CVE-2021-22555
- https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/net/netfilter/x_tables.c?id=b29c457a6511435960115c0f548c4360d5f4801d
- https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/net/netfilter/x_tables.c?id=9fa492cdc160cd27ce1046cb36f47d3b2b1efa21
- https://github.com/torvalds/linux/blob/v2.6.19/net/netfilter/x_tables.c
- https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-22555
- https://nvd.nist.gov/vuln/detail/CVE-2021-22555
- https://access.redhat.com/security/cve/CVE-2021-22555
- https://en.wikipedia.org/wiki/Iptables
- https://vulners.com/cve/CVE-2021-22555
- https://elixir.bootlin.com