CVE-2021-22555 Analysis Report

Analysis report on Local Privilege Escalation

Caused by Out-of-bounds in the Linux Netfilter module

First Author	Andy Nguyen
Author	iCAROS7 (Homin Rhee)
Data Created	2022.07.18 Mon
Data Version	1.0.1

Index

1. Introduce

CVE-2021-22555는 user namespace 내에서 contaning 되어있는 application이 heap Out-of-bounds (이하 oob)를 통해 privilege escalation이 가능한 취약점 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` variable에 의하여 induce 된다.

Linux kernel version 2.6.19 ~ 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 `Andy Nguyen` 에 의해 최초 보고 되었으며, 동년 동월 13일 Linux main upstream branch에 patch가 `Florian Westphal` 로부터 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. `__alignof__` value per architecture

`__alignof__`의 return value는 argument로 주어진 data의 alignment을 위해 필요한 byte이다. 이를 이용하여 하기와 같아 compile을 수행한 architecture의 register의 bit를 확인하는 용도로도 활용이 가능하다.

```
1  /* this is a dummy structure to find out the alignment requirement for
   a struct
2   * containing all the fundamental data types that are used in
   ipt_entry,
3   * ip6t_entry and arpt_entry.  This sucks, and it is a hack.  It will
   be my
4   * personal pleasure to remove it -HW
5   */
6
7  struct _xt_align {
8    __u8 u8;
9    __u16 u16;
10   __u32 u32;
11   __u64 u64;
12 };
13
14 #define SIZE __alignof__(struct _xt_align)
```

`u8`, `u16`, `u32`, `u64`이 declaration 된 struct의 `__alignof__`를 수행 시 가장 큰 data type를 기준으로 return 한다.

2-2. Code audit

```
1  # /net/netfilter/x_tables.c
2
3  void xt_compat_match_from_user(struct xt_entry_match *m, void **dstptr,
   int *size)
```

```
4  {
5    struct xt_match *match = m->u.kernel.match;
   // [1]
6    struct compat_xt_entry_match *cm = (struct compat_xt_entry_match *)m;
   // [2]
7    int pad, off = xt_compat_match_offset(match);
   // [3]
8    u_int16_t msize = cm->u.user.match_size;
   // [4]
9
10   m = *dstptr;                                          // [5]
11   memcpy(m, cm, sizeof(*cm));                           // [6]
12   if (match->compat_from_user)                          // [7]
13     match->compat_from_user(m->data, cm->data);
14   else
15     memcpy(m->data, cm->data, msize - sizeof(*cm));
16   pad = XT_ALIGN(match->matchsize) - match->matchsize;  // [8]
17   if (pad > 0)                                          // [9]
18     memset(m->data + match->matchsize, 0, pad);
19
20   msize += off;                    // [10]
21   m->u.user.match_size = msize;    // [11]
22
23   *size += off;                    // [12]
24   *dstptr += msize;                // [13]
25 }
```

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 compile를 진행한 architecture에 따라 결정된다. 따라서 `matchsize` + `n` - `compatsize` 로 연산하여 둘의 difference가 대입된다.

[4]에서 unsigned 16bit integer type `msize` 에 argument의 user.match_size를 가져온 이후, `m` pointer 의 address를 argument의 `dstptr` 로 change 한다. **[6]**과 같이 `memcpy` function을 통해 `dstptr` 의 data를 다시 `cm` 의 address로 `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 `compat_from_user` function을, 존재하지 않는다면 `memcpy` 로 `cm` struct의 data를 `m` struct의 `data` 로 copy 한다.

이후 **[8]**에서 `pad` variable에 위에서 1회 연산 되었던 `XT_ALIGN(matchsize)` 와 `matchsize` 를 subtraction한다. 이는 archtecture에 따라 다른 값이 나오며, 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.

```
1  void xt_compat_target_from_user(struct xt_entry_target *t, void
   **dstptr,
2          unsigned int *size)
3  {
4    const struct xt_target *target = t->u.kernel.target;
5    struct compat_xt_entry_target *ct = (struct compat_xt_entry_target
   *)t;
6    int pad, off = xt_compat_target_offset(target);
7    u_int16_t tsize = ct->u.user.target_size;
8    char name[sizeof(t->u.user.name)];                    // [1]
9
10   t = *dstptr;
11   memcpy(t, ct, sizeof(*ct));
12   if (target->compat_from_user)
13     target->compat_from_user(t->data, ct->data);
14   else
15     memcpy(t->data, ct->data, tsize - sizeof(*ct));
16   pad = XT_ALIGN(target->targetsize) - target->targetsize;
17   if (pad > 0)
```

```
18      memset(t->data + target->targetsize, 0, pad);
19
20    tsize += off;
21    t->u.user.target_size = tsize;
22
23    strlcpy(name, target->name, sizeof(name));                // [2]
24    module_put(target->me);                                   // [3]
25    strncpy(t->u.user.name, name, sizeof(t->u.user.name));    // [4]
26    *size += off;
27    *dstptr += tsize;
28  }
```

`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 User가 request한 netfilter table의 re-allocating을 위해 target indexing을 목적으로 call 되는 netfilter의 `conntrack_bridge` module의 reference counter가 1회 incrase 되었던 것을 다시 decrease 시킨다.

[4]에서 다시금 `strncpy` function을 통해 user.name에 `name`의 address에 u.user.name의 size 만큼 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);            // [1]

  pad = XT_ALIGN(match->matchsize) - match->matchsize;     // [2]
  if (pad > 0)
    memset(m->data + match->matchsize, 0, pad);            // [3]

# xt_compat_target_from_user
  int pad, off = xt_compat_target_offset(target);          // [1]

  pad = XT_ALIGN(target->targetsize) - target->targetsize; // [2]
  if (pad > 0)
    memset(t->data + target->targetsize, 0, pad);          // [3]
```

[1] 및 **[2]**는 Integer type으로 declaration 된 `pad` variable에 compile를 진행한 archtecture 별 difference를 연산한다.

```
/* /linux/net/netfilter/x_tables.h */

#define XT_ALIGN(s) __ALIGN_KERNEL((s), __alignof__(struct _xt_align))

/* /linux/include/uapi/linux/const.h */
#define __ALIGN_KERNEL(x, a)    __ALIGN_KERNEL_MASK(x, (typeof(x))(a) - 1)
#define __ALIGN_KERNEL_MASK(x, mask)  (((x) + (mask)) & ~(mask))
```

이후 **[3]**에서 `memset` function을 통해 architecture 간 `m` struct convertion 과정에서 `matchsize` 및 `targetsize`에 대한 value verification이 없어 `memset` function의 3번째 argument로 `size_t num`에 abnormal한 `pad` value가 들어갈 경우 필요 이상의 address에 `0`이 access-able 한 status가 된다.

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)

4. Proof of concept

5. Patch for the vulnerability

```
1  /* /linux/net/netfilter/x_tables.c */
2
3    void xt_compat_match_from_user(struct xt_entry_match *m, void
   **dstptr,
```

```
4                unsigned int *size)
5   {
6     const struct xt_match *match = m->u.kernel.match;
7     struct compat_xt_entry_match *cm = (struct compat_xt_entry_match
  *)m;
8 -   int pad, off = xt_compat_match_offset(match);
9 +   int off = xt_compat_match_offset(match);            // [1]
10    u_int16_t msize = cm->u.user.match_size;
11    char name[sizeof(m->u.user.name)];
12
13    m = *dstptr;
14    memcpy(m, cm, sizeof(*cm));
15    if (match->compat_from_user)
16      match->compat_from_user(m->data, cm->data);
17    else
18      memcpy(m->data, cm->data, msize - sizeof(*cm));
19 -   pad = XT_ALIGN(match->matchsize) - match->matchsize;
20 -   if (pad > 0)
21 -     memset(m->data + match->matchsize, 0, pad);       // [2]
22
23    msize += off;
24    m->u.user.match_size = msize;
```

```
1  /* /linux/net/netfilter/x_tables.c */
2
3    void xt_compat_target_from_user(struct xt_entry_target *t, void
  **dstptr,
4            unsigned int *size)
5   {
6     const struct xt_target *target = t->u.kernel.target;
7     struct compat_xt_entry_target *ct = (struct compat_xt_entry_target
  *)t;
8 -   int pad, off = xt_compat_target_offset(target);
9 +   int off = xt_compat_target_offset(target);          // [1]
10    u_int16_t tsize = ct->u.user.target_size;
11    char name[sizeof(t->u.user.name)];
```

```
12
13      t = *dstptr;
14      memcpy(t, ct, sizeof(*ct));
15      if (target->compat_from_user)
16        target->compat_from_user(t->data, ct->data);
17      else
18        memcpy(t->data, ct->data, tsize - sizeof(*ct));
19  -   pad = XT_ALIGN(target->targetsize) - target->targetsize;
20  -   if (pad > 0)
21  -     memset(t->data + target->targetsize, 0, pad);     // [2]
22
23      tsize += off;
24      t->u.user.target_size = tsize;
```

우선 **[1]** 및 **[2]**와 같이 initialization을 목적으로 하는 memory access 자체를 행하지 않게 하여 직접적으로 unwanted한 memory access를 막을 수 있다.

```
1  /* /linux/net/ipv4/netfilter/arp_tables.c */
2
3    static int translate_compat_table(struct net *net,
4              struct xt_table_info **pinfo,
5              void **pentry0,
6              const struct compat_arpt_replace *compatr)
7    {
8      ret = -ENOMEM;
9      newinfo = xt_alloc_table_info(size);
10     if (!newinfo)
11       goto out_unlock;
12
13 +   memset(newinfo->entries, 0, size);        // [3]
14
15     newinfo->number = compatr->num_entries;
```

```
/* /linux/net/ipv4/netfilter/ip_tables.c */

  static int get_info(struct net *net, void __user *user, const int
*len)
  {
+   memset(&info, 0, sizeof(info));           // [3]
    info.valid_hooks = t->valid_hooks;
    memcpy(info.hook_entry, private->hook_entry,
           sizeof(info.hook_entry));
```

```
/* /linux/net/ipv6/netfilter/ip6_tables.c */

  static int get_info(struct net *net, void __user *user, const int
*len)
  {
+   memset(&info, 0, sizeof(info));           // [3]
    info.valid_hooks = t->valid_hooks;
    memcpy(info.hook_entry, private->hook_entry,
           sizeof(info.hook_entry));
```

이후 실질적으로 `do_ipt_get_ctl` 과 같은 `xt_compat_*_from_user` function을 call 하는 parent에서 이용되는 `get_info` 에서 최종적으로 결정된 target의 size를 `sizeof` function을 통해 `memset` 하여 주는 것으로 patch가 가능하다.

6. Conclusion

본 report는 version `1.0.1` 까지 author의 skill 부족으로 인해 Proof of Concept을 포함하지 못하였다. 이후 해당 issue가 해결된다면 지울 part 이다.

하기와 같은 component가 앞으로 learning이 필요하다 teach 받았다.

- Checkout each mitgation mechanism

- Learning about mitgation bypass without `pwntools`
- Basic level binary exploit via C PoC
- Auditing and write write-up without online searching at auditing stage

6. Conclusion

위 vulnerability code의 경우 local 상에서 complexity가 high 하지 않게 low-preivilieges status에서 별도의 user interaction 없이 system confidentiality / integritiy / availability에 상당한 영향력을 끼치므로 높게 scoring 되었다. USD 기준 추정 $5,000 ~ $25,000의 bug bounty가 offer 된 것으로 추정 된다.

7. Reference

Follow template is temporary for building step of report

- About CVE-2021-22555
  - https://www.cvedetails.com/cve/CVE-2021-22555/
  - https://ubuntu.com/security/CVE-2021-22555
  - 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://vulners.com/cve/CVE-2021-22555
  - https://github.com/torvalds/linux/commit/b29c457a6511435960115c0f548c4360d5f4801d
  - 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://lore.kernel.org
- About `netfilter` & `iptables`

    - https://www.netfilter.org
    - https://en.wikipedia.org/wiki/Iptables
    - https://www.digitalocean.com/community/tutorials/a-deep-dive-into-iptables-and-netfilter-architecture
    - https://programmer.group/netfilter-framework-of-linux-kernel.html
- For code auditing
    - https://elixir.bootlin.com
    - https://jybaek.tistory.com/160
    - https://gcc.gnu.org/onlinedocs/gcc-6.2.0/gcc/Alignment.html
    - https://www.ibm.com/docs/en/i/7.4?topic=expressions-alignof-operator
    - https://blog.katastros.com/a?ID=01550-a475cf58-24f4-403a-9f6e-8a800ba9ae14