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	1.0.2

Index

1. Introduce
2. Analysis of crash occurrence function
    1. Basic knowledge
        1. `__alignof__` value per architecture
    2. Code audit
    3. Vulnerability analysis
3. Crash inducement
    1. Host PC specification
    2. Vulnerability analysis

4. Proof of concept
    1. For Use-After-Free (1)
    2. For SMAP bypass
    3. For Use-After-Free (2)
    4. For KASLR bypass
    5. For local privilege escalation
5. Patch for the vulnerability
6. Conclusion
7. Reference

1. Introduce

CVE-2021-22555는 사용자 영역 내에서 분리 되어있는 어플리케이션이 heap Out-of-bounds를 통해 권한 상승이 가능한 취약점 이다.

Linux kernel의 network 관련 handler 형식의 framework인 `netfilter` 내에서 유발 된다. 2006년 경 수정 된 `x_tables` 모듈 내 `xt_compat_match()` 에서 type-safe 의 목적으로 분리 된 함수로부터 시작 된다.
`xt_compat_match_from_user()` 및 `xt_compat_target_from_user()` 내 integer 형식로 선언 된 불완전한 한 `pad` 변수에 의하여 유도된다.

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 메인 upstream 브랜치에 패치가 `Florian Westphal` 로부터 병합되며 각 벤더 업체들에게 release 되었다. 이후 동년 7월 7일 공개 release 되며 CVSS 3.0 기준 `7.8 High` 로 점수를 받았다.

또한 이는 Kubernetes environment의 SBX (sandbox escaping)에 사용이 가능한것을 고려한다면 상당한 위험성을 내제하고 있다.

2. Analysis of crash occurrence function

2-1. Basic knowledge

2-1-1. `__alignof__` value per architecture

`__alignof__` 의 반환 값은 매개 변수로 주어진 data의 정렬을 위해 필요한 byte이다. 이를 이용하여 하기와 같아 컴파일을 수행한 아키텍처의 레지스터의 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 선언 된 구조체의 `__alignof__` 를 수행 시 가장 큰 data의 형식을 기준으로 반환 한다.

2-2. Code audit

```
1  /net/netfilter/x_tables.c
2
3  oid xt_compat_match_from_user(struct xt_entry_match *m, void **dstptr,
   nt *size)
4  {
5    struct xt_match *match = m->u.kernel.match;
6    struct compat_xt_entry_match *cm = (struct compat_xt_entry_match *)m;
7    int pad, off = xt_compat_match_offset(match);
8    u_int16_t msize = cm->u.user.match_size;
9
10   m = *dstptr;
11   memcpy(m, cm, sizeof(*cm));
12   if (match->compat_from_user)
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;
17   if (pad > 0)
18     memset(m->data + match->matchsize, 0, pad);
19
20   msize += off;
21   m->u.user.match_size = msize;
22
23   *size += off;
24   *dstptr += msize;
25 }
```

우선 **Line. 5~8**에서는 매개변수로 넘어온 `xt_entry_match` 구조체를 User Kernel Match data의 메모리 주소를 `m` 에 가져온다. 다음 `compat_xt_entry_match` 형식의 구조체 포인터 `cm` 에 매개변수로 받은 `m` 을 `compat_xt_entry_match` 으로 형변환 하여 대입한다.

이후 integer 형식의 `pad` 를 선언하며 `off` 를 `xt_compat_match_offset()` 의 반환 data로 `compatsize` 의 값에 따라 (`matchsize` + `__alignof__` - 1) - `compatsize` 를 통해 할당 한다. 여기서 `__alignof__` 로 정의된 값은 컴파일을 진행한 시스템 아키텍처에 따라 결정된다. 따라서 `matchsize` + `n` - `compatsize` 로 둘의 차이값이 대입된다. 다음 unsigned 16bit integer 형식의 `msize` 에 매개변수의 user.match_size를 대입한다.

Line. 10, 11에서는 `m` 포인터의 주소를 매개변수의 `dstptr` 로 바꾼다. 이후 `memcpy()` 을 통해 `dstptr` 의 data를 다시 `cm` 의 주소로 `cm` 의 크기 만큼 복사한다.

Line 12~15의 if 문에서 kernel space 상에서 별도의 정렬이 필요할 경우 Kernel config 내 boolean 형식의 `CONFIG_COMPAT` 값에 따라 존재가 결정되는 `compat_from_user()` 가 조건이 된다. `compat_from_user()` 이 존재한다면 매개변수로 `m` 과 `cm` 의 data로 넘겨 `cm` 의 data를 최종적으로 `m` 의 data에 넘긴다. 만일 존재하지 않는다면 `memcpy()` 로 `cm` struct의 data를 `m` 구조체의 `data` 로 복사한다.

Line. 16~18에서 `pad` 변수에 위에서 1회 연산 되었던 `XT_ALIGN(matchsize)` 와 `matchsize` 를 차를 계산하여 대입한다. 이는 아키텍처에 따라 다른 값이 나온다. 이때 if 문의 조건에 따라 **Line. 16**의 `pad` 값이 0보다 큰 경우에는 `memset()` 을 통해 `m` 구조체의 `data` 주소와 `match` 의 `match` 주소의 합에 `0` 을 `pad` 값 만큼 메모리를 초기화한다.

Line. 20, 21에서 `msize` 의 값을 **Line. 7**에서 계산된 `off` 값 만큼 증가 시킨 뒤, user.match_size 를 `msize` 에 대입한다. 이후 **Line 23, 24**에서 매개변수의 `size` 포인터가 가르키는 값 또한 `off` 값 만큼 증가 시킨 다음 `dstptr` 포인터 역시 `msize` 만큼 증가 시킨다.

```
1  xt_compat_target_from_user(struct xt_entry_target *t, void
  tptr,
2      unsigned int *size)
3
4  ıst struct xt_target *target = t->u.kernel.target;
5  ruct compat_xt_entry_target *ct = (struct compat_xt_entry_target

6  t pad, off = xt_compat_target_offset(target);
7  int16_t tsize = ct->u.user.target_size;
8  ar name[sizeof(t->u.user.name)];                    // [1]
9
```

```
10  = *dstptr;
11  ncpy(t, ct, sizeof(*ct));
12   (target->compat_from_user)
13  target->compat_from_user(t->data, ct->data);
14  se
15  memcpy(t->data, ct->data, tsize - sizeof(*ct));
16  d = XT_ALIGN(target->targetsize) - target->targetsize;
17   (pad > 0)
18  memset(t->data + target->targetsize, 0, pad);
19
20  ize += off;
21  >u.user.target_size = tsize;
22
23  rlcpy(name, target->name, sizeof(name));                // [2]
24  dule_put(target->me);                                   // [3]
25  rncpy(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 메커니즘으로 동작하지만 차이점은 다음과 같다.

Line. 8에서 char 형식으로 `user.name` 의 크기 만큼 배열을 구성한다. 이후 `memset()` 으로 메모리 초기화까지 동일하게 진행된다.

Line. 23 `strlcpy()` 으로 `\0` 을 포함하여 `target` 구조체의 `name` 을 **Line. 8**에서 선언된 `name` 에 복사한다. 이후 **Line. 24**에서 User가 요청한 netfilter table의 재할당을 위해 target 인덱싱을 목적으로 호출되는 netfilter의 `conntrack_bridge` 모듈의 참조 계수기가 1회 증가 되었던 것을 다시 감소 시킨다.

Line. 25에서 다시금 `strncpy()` 을 통해 user.name에 `name` 의 주소에 u.user.name의 크기만큼 복사한다.

2-3. Vulnerability analysis

실질적으로 취약한 code는 하기와 같다.

```
1  # xt_compat_match_from_user
2    int pad, off = xt_compat_match_offset(match);
3
4    pad = XT_ALIGN(match->matchsize) - match->matchsize;
5    if (pad > 0)
6      memset(m->data + match->matchsize, 0, pad);
7
8  # xt_compat_target_from_user
9    int pad, off = xt_compat_target_offset(target);
10
11   pad = XT_ALIGN(target->targetsize) - target->targetsize;
12   if (pad > 0)
13     memset(t->data + target->targetsize, 0, pad);
```

Line. 2, 9 및 **Line. 4, 11**은 integer 형식으로 선언된 `pad` 변수에 컴파일을 진행한 아키텍처 별 차이 값을 연산한다.

```
1 /* /linux/net/netfilter/x_tables.h */
2
3 #define XT_ALIGN(s) __ALIGN_KERNEL((s), __alignof__(struct _xt_align))
4
5 /* /linux/include/uapi/linux/const.h */
6 #define __ALIGN_KERNEL(x, a)    __ALIGN_KERNEL_MASK(x, (typeof(x))(a) -
  1)
7 #define __ALIGN_KERNEL_MASK(x, mask)  (((x) + (mask)) & ~(mask))
```

이후 `memset()` 및 null padding을 이용한 아키텍처 간 `m` 구조체 형변환 과정에서 `matchsize` 및 `targetsize`에 대한 검증 과정이 없다. 이로인하여 `memset()`의 3번째 매개변수로 `size_t num`에 비정상적인 `pad` 값가 들어갈 경우 oob가 유발된다.

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

4. Proof of concept

4-1. For Use-After-Free (1)

4-2. For SMAP/SMEP bypass

4-3. For Use-After-Free (2)

4-4. For KASLR bypass

4-5. For Local Privilege Escaltion

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 초기화를 목적으로 하는 메모리 접근 자체를 행하지 않게 하여 직접적으로 의도치 않은 접근을 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()` 을 호출하는 메서드에서 이용되는 `get_info` 에서 **[3]**과 같이 최종적으로 결정된 target의 size를 `sizeof()` 을 통해 `memset()` 하여 주는 것으로 patch가 가능하다.

이후 실질적으로 `do_ipt_get_ctl()` 등과 같이 `xt_compat_*_from_user()` 를 호출하는 메서드에서 이용되는 `get_info()` 에서 **[3]**과 같이 이미 최종적으로 결정된 target의 크기를 `sizeof()` 를 통해 `memset()` 하여 주는 것으로 패치가 가능하다.

6. Conclusion

위 취약한 code의 경우 로컬 상에서 복잡도가 높지않게, 높은 권한을 요구하지 않은 상태에서 별도의 사용자 상호작용 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.

본 report는 version `1.0.2` 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
- About heap grooming

7. Reference

- 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

- 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

- 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