# LiveFire writeup

HolyShield 2025

# 목차

0. 개요
1. Buffer OverFlow
2. Double Free
3. Command Injection

# 0. 개요

LiveFire 에서 제공된 file_manager 바이너리에는 총 3개의 취약점이 존재합니다.

- Buffer OverFlow
- Use After Free
- Command Injection

본 문서에서는 위의 3가지 취약점에 대한 설명 및 바이너리 패치를 통한 취약점 패치 방법을 설명합니다.

바이너리 패치 방법들은 모두 바이너리의 기존 파일 크기를 유지하며, healthCheck.py 의 결과가 기존 바이너리와 동일하도록 진행되었습니다.

# 1. Buffer OverFlow

```
<+692>:   mov    rax,QWORD PTR [rbp-0x38]
<+696>:   sub    rax,QWORD PTR [rbp-0x28]
<+700>:   mov    QWORD PTR [rbp-0x40],rax
<+704>:   mov    QWORD PTR [rbp-0x2002e0],0x0
<+715>:   mov    QWORD PTR [rbp-0x2002d8],0x0
<+726>:   lea    rax,[rbp-0x2002d0]
<+733>:   mov    edx,0x1ffff1
<+738>:   mov    esi,0x0
<+743>:   mov    rdi,rax
<+746>:   call   0x402350 <memset@plt>
<+751>:   lea    rdx,[rbp-0x2002e0]
<+758>:   mov    rcx,QWORD PTR [rbp-0x40]
<+762>:   mov    rax,QWORD PTR [rbp-0x28]
<+766>:   mov    rsi,rcx
<+769>:   mov    rdi,rax
<+772>:   call   0x4030aa <base64_encode>
```

```
// 파일 크기 계산
size_t file_size = content_end - content_start;

// base64_contents 변수에 Base64로 인코딩하여 저장
char base64_contents[1024 * 1024 * 2] = {0,};
base64_encode(content_start, file_size, base64_contents);
```

upload_post 함수에서 multipart/form-data 를 읽을 때 파일 크기에 대한 검증이 없습니다. 이에 따라 main 함수에서 요청을 받는 버퍼의 크기 2M에 근접하는 크기의 파일을 업로드할 수 있습니다.

```
<+106>:   lea    rax,[rbp-0x300120]
<+113>:   lea    rdx,[rax+0x200001]
<+120>:   lea    rax,[rbp-0x300120]
<+127>:   mov    rsi,rdx
<+130>:   mov    rdi,rax
<+133>:   call   0x4032db <base64_decode>
<+138>:   mov    QWORD PTR [rbp-0x8],rax
```

```
struct {
    char contents[1024 * 1024 * 2 + 1];
    unsigned char decoded_contents[1024 * 1024];
} data;

...

// Base64 디코딩
size_t decoded_size = base64_decode(data.contents, data.decoded_contents);
```

download_get 함수에서 업로드된 데이터를 불러올 때 파일 크기에 대한 검증이 없습니다. 따라서 1M 크기 이상의 파일을 다운받는 요청을 보내면 download_get 함수에서 BOF가 발생합니다.

```
<+692>:   mov    rax,QWORD PTR [rbp-0x38]
<+696>:   sub    rax,QWORD PTR [rbp-0x28]
<+700>:   mov    QWORD PTR [rbp-0x40],rax
<+704>:   mov    QWORD PTR [rbp-0x2002e0],0x0
<+715>:   mov    QWORD PTR [rbp-0x2002d8],0x0
<+726>:   lea    rax,[rbp-0x2002d0]
<+733>:   mov    edx,0x1ffff1
<+738>:   mov    esi,0x0
<+743>:   mov    rdi,rax
<+746>:   call   0x402350 <memset@plt>
<+751>:   lea    rdx,[rbp-0x2002e0]
<+758>:   mov    rcx,QWORD PTR [rbp-0x40]
<+762>:   mov    rax,QWORD PTR [rbp-0x28]
<+766>:   mov    rsi,rcx
<+769>:   mov    rdi,rax
<+772>:   call   0x4030aa <base64_encode>
```

```
<+692>:   mov    rax,QWORD PTR [rbp-0x38]
<+696>:   sub    rax,QWORD PTR [rbp-0x28]
<+700>:   mov    QWORD PTR [rbp-0x40],rax
<+704>:   cmp    QWORD PTR [rbp-0x40],0x100000
<+712>:   ja     0x4044ba <upload_post+659>
<+714>:   nop
 ...
<+750>:   nop
<+751>:   lea    rdx,[rbp-0x2002e0]
<+758>:   mov    rcx,QWORD PTR [rbp-0x40]
<+762>:   mov    rax,QWORD PTR [rbp-0x28]
<+766>:   mov    rsi,rcx
<+769>:   mov    rdi,rax
<+772>:   call   0x4030aa <base64_encode>
```

upload_post 함수에서 로직 상 의미없는 memset 코드를 삭제하고, 파일 크기를 검사하는 분기문을 추가해 크기 1M 이상의 파일 업로드 요청이 오면 400 응답을 하는 코드로 건너뜁니다.

# 2. Double Free

```
<+79>:    mov    eax,DWORD PTR [rbp-0x4]
<+82>:    cdqe
<+84>:    mov    rax,QWORD PTR [rax*8+0x40b200]
<+92>:    mov    DWORD PTR [rax+0x8],0x0
<+99>:    mov    eax,DWORD PTR [rbp-0x4]
<+102>:   cdqe
<+104>:   mov    rax,QWORD PTR [rax*8+0x40b200]
<+112>:   mov    rdi,rax
<+115>:   call   0x402420 <free@plt>
<+120>:   mov    eax,0x0
<+125>:   jmp    0x4030a8 <session_delete+145>
...
<+145>:   leave
<+146>:   ret
```

```
sessions[i]->is_alive = 0;
free(sessions[i]);
return 0;
```

session_delete 함수에서 세션이 담긴 포인터를 free한 후 sessions 배열에서 세션 포인터를 삭제하지 않습니다. main -> handle_request -> logout_get -> session_delete 로 진행되는 과정에서 삭제될 세션 구조체의 포인터에 대한 검증이 이루어지지 않으므로, 같은 session_id에 대해 2번 이상의 삭제 요청을 보낼 경우 Double Free 가 발생합니다.

```
<+79>:    mov    eax,DWORD PTR [rbp-0x4]
<+82>:    cdqe
<+84>:    mov    rax,QWORD PTR [rax*8+0x40b200]
<+92>:    mov    DWORD PTR [rax+0x8],0x0
<+99>:    mov    eax,DWORD PTR [rbp-0x4]
<+102>:   cdqe
<+104>:   mov    rax,QWORD PTR [rax*8+0x40b200]
<+112>:   mov    rdi,rax
<+115>:   call   0x402420 <free@plt>
<+120>:   mov    eax,0x0
<+125>:   jmp    0x4030a8 <session_delete+145>
...
<+145>:   leave
<+146>:   ret
```

```
<+79>:    mov    eax,DWORD PTR [rbp-0x4]
<+82>:    cdqe
<+84>:    mov    rax,QWORD PTR [rax*8+0x40b200]
<+92>:    mov    rdi,rax
<+95>:    call   0x402420 <free@plt>
<+100>:   mov    eax,DWORD PTR [rbp-0x4]
<+103>:   cdqe
<+105>:   xor    rdi,rdi
<+108>:   mov    QWORD PTR [rax*8+0x40b200],rdi
<+116>:   nop
<+117>:   nop
<+118>:   nop
<+119>:   nop
<+120>:   mov    eax,0x0
<+125>:   jmp    0x4030a8 <session_delete+145>
```

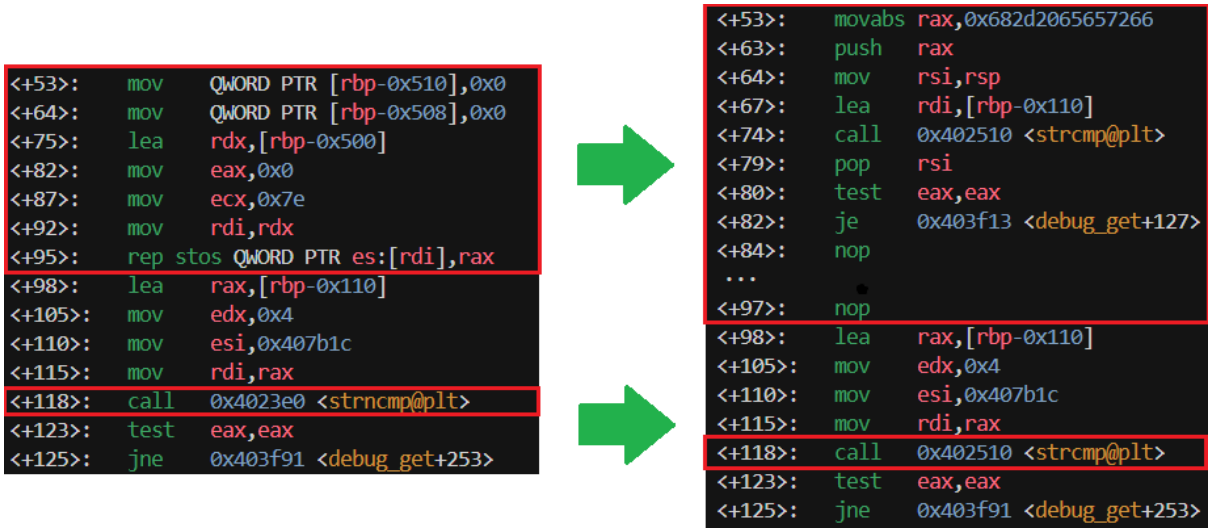
session 구조체의 is_alive 멤버는 session_create 함수에서 설정되고 session_delete 함수에서 삭제되지만, 그 외에 쓰이는 곳이 없어 로직 상 의미 없는 코드입니다. session_delete 함수에서 해당 코드를 지우고, 구조체 포인터 free 후 해당 포인터를 sessions 배열에서 삭제하는 코드를 추가합니다.

# 3. Command Injection

```
<+53>:    mov    QWORD PTR [rbp-0x510],0x0
<+64>:    mov    QWORD PTR [rbp-0x508],0x0
<+75>:    lea    rdx,[rbp-0x500]
<+82>:    mov    eax,0x0
<+87>:    mov    ecx,0x7e
<+92>:    mov    rdi,rdx
<+95>:    rep stos QWORD PTR es:[rdi],rax
<+98>:    lea    rax,[rbp-0x110]
<+105>:   mov    edx,0x4
<+110>:   mov    esi,0x407b1c
<+115>:   mov    rdi,rax
<+118>:   call   0x4023e0 <strncmp@plt>
<+123>:   test   eax,eax
<+125>:   jne    0x403f91 <debug_get+253>
<+127>:   lea    rax,[rbp-0x110]
<+134>:   mov    esi,0x407b21
<+139>:   mov    rdi,rax
<+142>:   call   0x4025e0 <popen@plt>
<+147>:   mov    QWORD PTR [rbp-0x8],rax
 ...
<+191>:   mov    rdx,QWORD PTR [rbp-0x8]
<+195>:   lea    rax,[rbp-0x510]
<+202>:   mov    rcx,rdx
<+205>:   mov    edx,0x3ff
<+210>:   mov    esi,0x1
<+215>:   mov    rdi,rax
<+218>:   call   0x402500 <fread@plt>
```

```
if (strncmp(decoded_cmd, "free", 4) == 0)
    FILE* p = popen(decoded_cmd, "r");
    ...
    int len = fread(output, 1, 1023, p);
```

/debug?cmd=... 경로로 요청을 보내면 debug_get 함수로 cmd 인자가 전달되어 popen 함수에서 해당 명령어가 실행됩니다. 그러나 cmd에 대한 검증이 strncmp(cmd, "free", 4) 밖에 없기 때문에 "free ; cat /etc/passwd" 등으로 우회 가능합니다.




healthCheck.py 에서 사용하는 "free" 와 "free -h" 만 통과할 수 있으면 됩니다.

strncmp 호출을 strcmp 호출로 바꿔 정확하게 "free" 네글자만 통과할 수 있게 합니다.

output 버퍼 초기화 코드를 지우고, strcmp(decoded_cmd, "free -h") 를 추가해 통과시 popen 쪽으로 점프하는 코드로 수정합니다.