Mitigation 원정대

(2)

Stack Canary
(Stack Smashing Protector)

Write by Py0zz1

까나리란?

```
[19:13:44] root@ubuntu:~/tmp/canary$ ./canary
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[19:13:58] root@ubuntu:~/tmp/canary$ ./canary
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAB
*** stack smashing detected ***: ./canary terminated
중지됨
```

Memory View

Buf[64] “AAAAAA…”	(0xdeadbeef) Canary	SFP	RET

버퍼에 “A” *64 입력

까나리란?

```
[19:13:44] root@ubuntu:~/tmp/canary$ ./canary
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[19:13:58] root@ubuntu:~/tmp/canary$ ./canary
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAB
*** stack smashing detected ***: ./canary terminated
중지됨
```

Memory View

Buf[64] “AAAAAA…”	(0xdeadbe42) Canary	SFP	RET

버퍼에 “A” *64 + “B” 입력

까나리란?

```
[19:13:44] root@ubuntu:~/tmp/canary$ ./canary
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[19:13:58] root@ubuntu:~/tmp/canary$ ./canary
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAB
*** stack smashing detected ***: ./canary terminated
중지됨
```

Buffer Overflow를 막기위한 보호기법

AAAAAAAAAAAA Buf[64]	(0xdeadbeef) Canary	SFP	RET

!

AAAAAAAAAAAA Buf[64]	(0xdeadbe42) Canary	SFP	RET

```
*** stack smashing detected ***: ./canary terminated
중지됨
```

까나리(Canary)의 유래

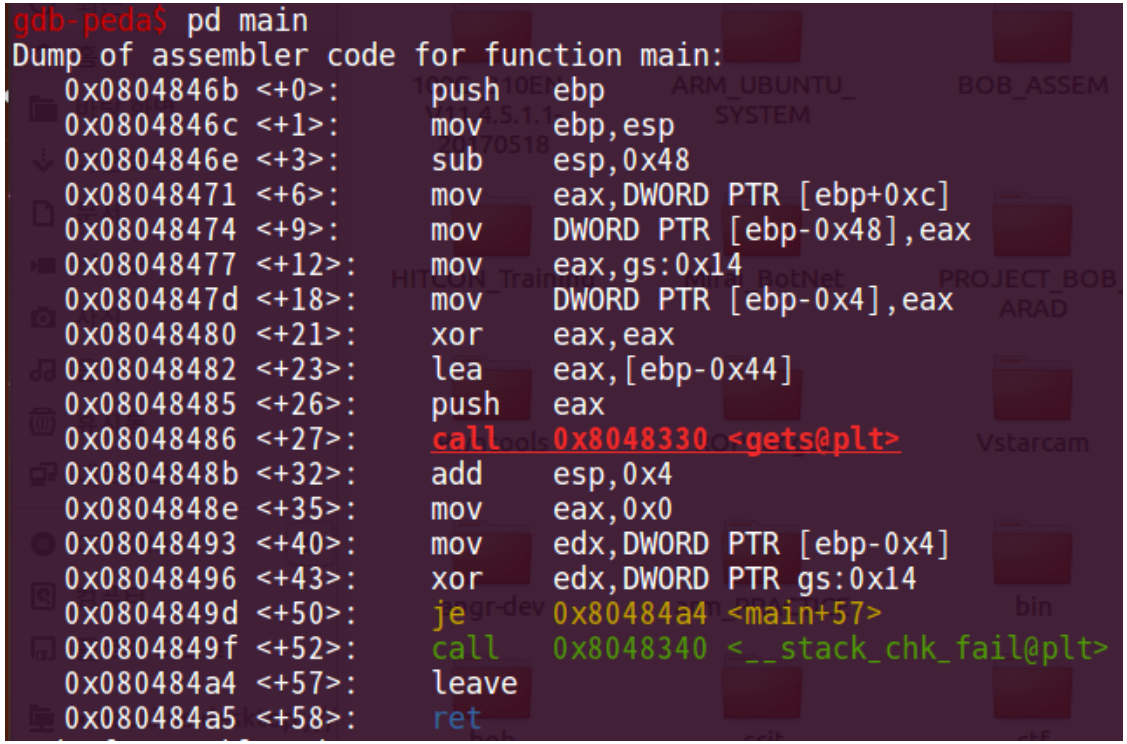
카나리아	
Atlantic canary	이명:
Serinus canaria Linnaeus, 1758	
분류	계 동물계
	문 척삭동물문(Chordata)
	강 조강(Aves)
	목 참새목(Passeriformes)
	과 되새과(Fringillidae)
	속 카나리아속(Serinus)
	종 **카나리아(S. canaria)**

Canary는 새의 이름으로, 산소 포화도의 민감한 것이 특징이다.

예전에 광부들이 갱도안에 Canary를 키우면서
Canary가 울기 시작하면 탄광작업을 멈추는 식으로
산소 포화도를 측정했다고 한다.

시스템에서 Data가 Canary공간을 덮으면
“Stack Smashing Detected”라는 Error가 발생하면서
바이너리가 종료된다.

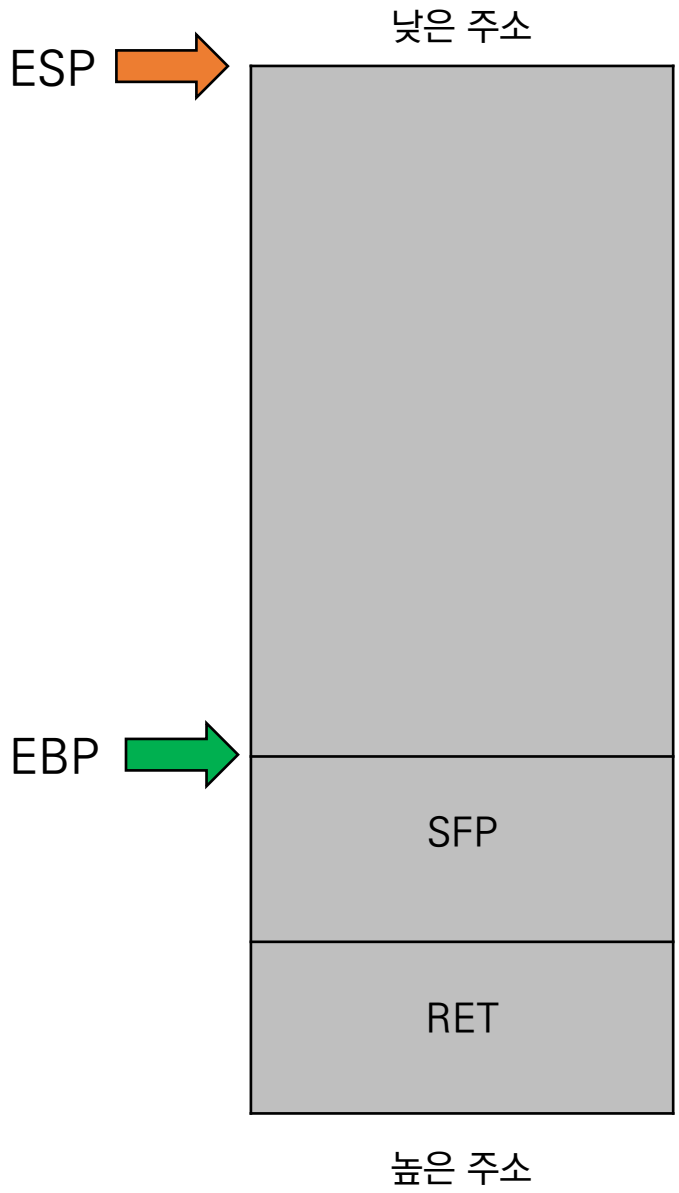
까나리 동작 흐름을 알아봅시다.



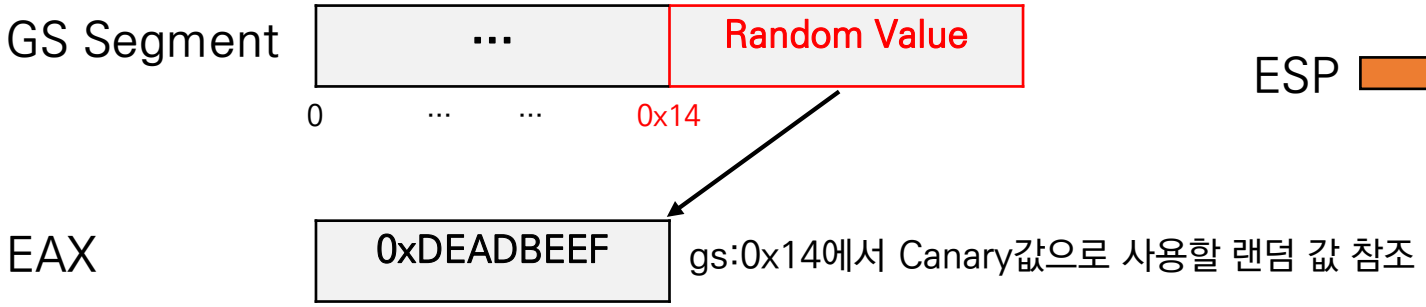
1. 함수 프롤로그

- Stack Frame구조를 설정한다.
- 필요한 Stack Size만큼 공간을 할당한다.

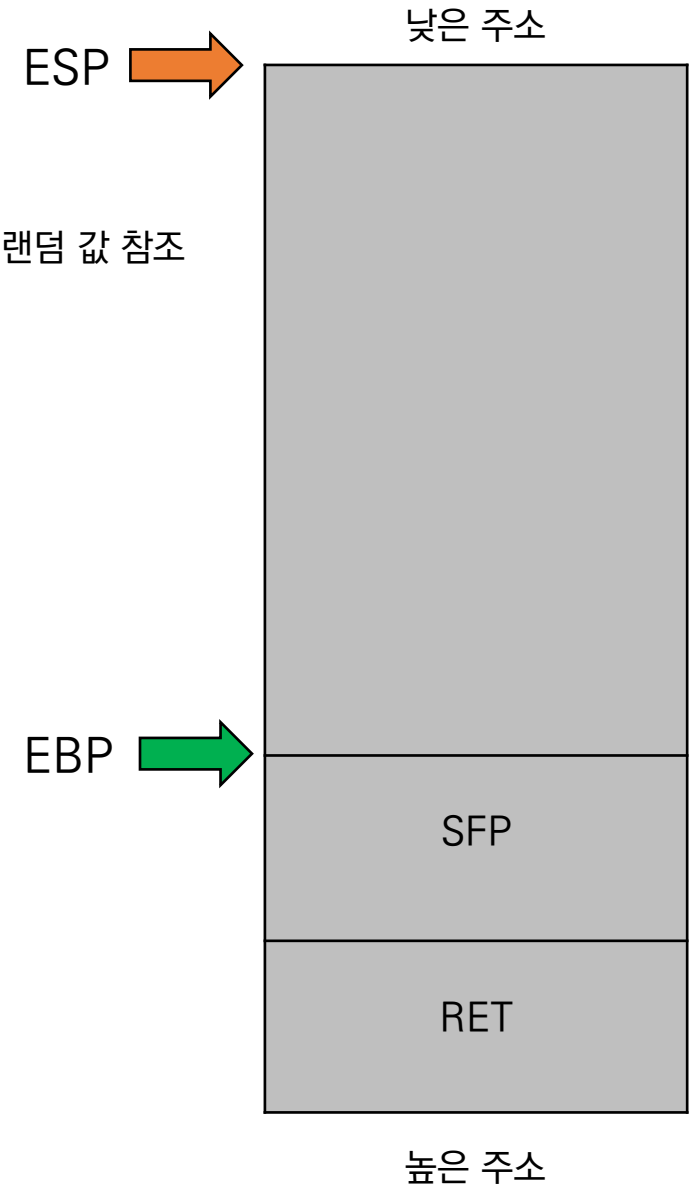
```
gdb-peda$ pd main
Dump of assembler code for function main:
   0x0804846b <+0>:     push   ebp
   0x0804846c <+1>:     mov    ebp,esp
   0x0804846e <+3>:     sub    esp,0x48
   0x08048471 <+6>:     mov    eax,DWORD PTR [ebp+0xc]
   0x08048474 <+9>:     mov    DWORD PTR [ebp-0x48],eax
   0x08048477 <+12>:    mov    eax,gs:0x14
   0x0804847d <+18>:    mov    DWORD PTR [ebp-0x4],eax
   0x08048480 <+21>:    xor    eax,eax
   0x08048482 <+23>:    lea    eax,[ebp-0x44]
   0x08048485 <+26>:    push   eax
   0x08048486 <+27>:    call   0x8048330 <gets@plt>
   0x0804848b <+32>:    add    esp,0x4
   0x0804848e <+35>:    mov    eax,0x0
   0x08048493 <+40>:    mov    edx,DWORD PTR [ebp-0x4]
   0x08048496 <+43>:    xor    edx,DWORD PTR gs:0x14
   0x0804849d <+50>:    je     0x80484a4 <main+57>
   0x0804849f <+52>:    call   0x8048340 <__stack_chk_fail@plt>
   0x080484a4 <+57>:    leave
   0x080484a5 <+58>:    ret
```



2. Canary 값 저장

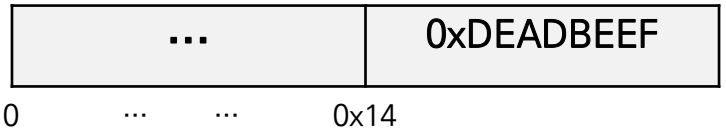



```
gdb-peda$ pd main
Dump of assembler code for function main:
   0x0804846b <+0>:     push   ebp
   0x0804846c <+1>:     mov    ebp,esp
   0x0804846e <+3>:     sub    esp,0x48
   0x08048471 <+6>:     mov    eax,DWORD PTR [ebp+0xc]
   0x08048474 <+9>:     mov    DWORD PTR [ebp-0x48],eax
   0x08048477 <+12>:    mov    eax,gs:0x14
   0x0804847d <+18>:    mov    DWORD PTR [ebp-0x4],eax
   0x08048480 <+21>:    xor    eax,eax
   0x08048482 <+23>:    lea    eax,[ebp-0x44]
   0x08048485 <+26>:    push   eax
   0x08048486 <+27>:    call   0x8048330 <gets@plt>
   0x0804848b <+32>:    add    esp,0x4
   0x0804848e <+35>:    mov    eax,0x0
   0x08048493 <+40>:    mov    edx,DWORD PTR [ebp-0x4]
   0x08048496 <+43>:    xor    edx,DWORD PTR gs:0x14
   0x0804849d <+50>:    je     0x80484a4 <main+57>
   0x0804849f <+52>:    call   0x8048340 <__stack_chk_fail@plt>
   0x080484a4 <+57>:    leave
   0x080484a5 <+58>:    ret
```



3. Canary 값 스택에 저장

GS Segment

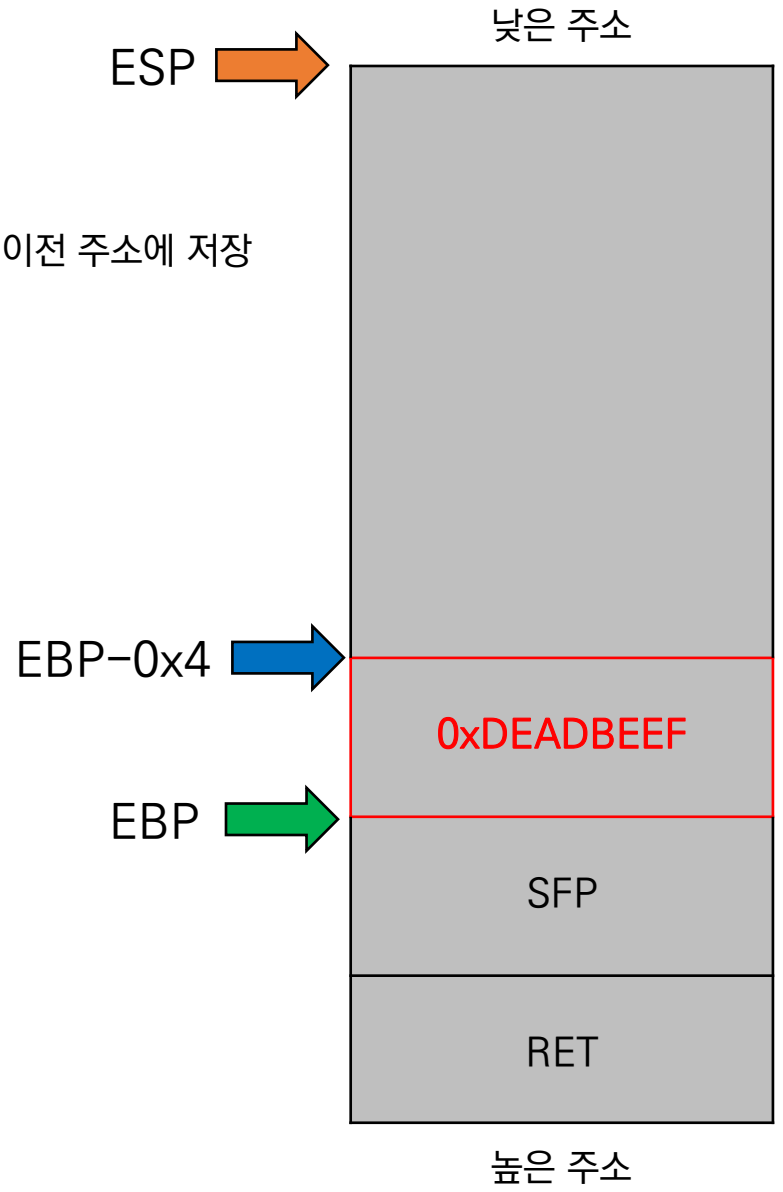


EAX

0xDEADBEEF

EAX에 저장된 Canary값 SFP 이전 주소에 저장

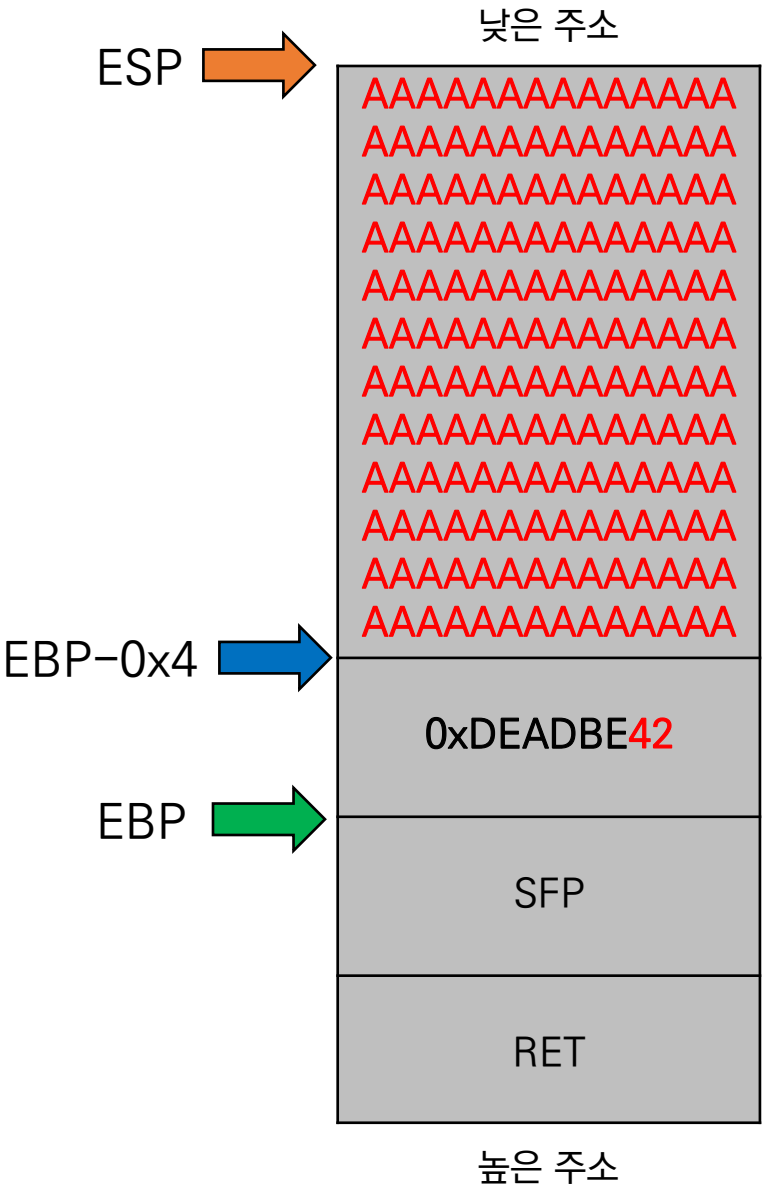
```
gdb-peda$ pd main
Dump of assembler code for function main:
   0x0804846b <+0>:     push   ebp
   0x0804846c <+1>:     mov    ebp,esp
   0x0804846e <+3>:     sub    esp,0x48
   0x08048471 <+6>:     mov    eax,DWORD PTR [ebp+0xc]
   0x08048474 <+9>:     mov    DWORD PTR [ebp-0x48],eax
   0x08048477 <+12>:    mov    eax,gs:0x14
   0x0804847d <+18>:    mov    DWORD PTR [ebp-0x4],eax
   0x08048480 <+21>:    xor    eax,eax
   0x08048482 <+23>:    lea    eax,[ebp-0x44]
   0x08048485 <+26>:    push   eax
   0x08048486 <+27>:    call   0x8048330 <gets@plt>
   0x0804848b <+32>:    add    esp,0x4
   0x0804848e <+35>:    mov    eax,0x0
   0x08048493 <+40>:    mov    edx,DWORD PTR [ebp-0x4]
   0x08048496 <+43>:    xor    edx,DWORD PTR gs:0x14
   0x0804849d <+50>:    je     0x80484a4 <main+57>
   0x0804849f <+52>:    call   0x8048340 <__stack_chk_fail@plt>
   0x080484a4 <+57>:    leave
   0x080484a5 <+58>:    ret
```



4. 사용자 Input입력

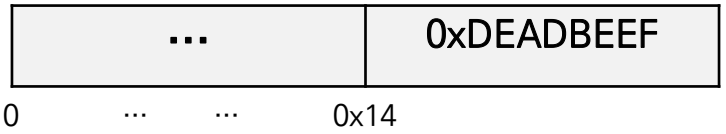
gets( )는 입력 길이 값을 검사하지 않기 때문에
Buffer Overflow취약점이 발생한다.

```
gdb-peda$ pd main
Dump of assembler code for function main:
   0x0804846b <+0>:     push   ebp
   0x0804846c <+1>:     mov    ebp,esp
   0x0804846e <+3>:     sub    esp,0x48
   0x08048471 <+6>:     mov    eax,DWORD PTR [ebp+0xc]
   0x08048474 <+9>:     mov    DWORD PTR [ebp-0x48],eax
   0x08048477 <+12>:    mov    eax,gs:0x14
   0x0804847d <+18>:    mov    DWORD PTR [ebp-0x4],eax
   0x08048480 <+21>:    xor    eax,eax
   0x08048482 <+23>:    lea    eax,[ebp-0x44]
   0x08048485 <+26>:    push   eax
   0x08048486 <+27>:    call   0x8048330 <gets@plt>
   0x0804848b <+32>:    add    esp,0x4
   0x0804848e <+35>:    mov    eax,0x0
   0x08048493 <+40>:    mov    edx,DWORD PTR [ebp-0x4]
   0x08048496 <+43>:    xor    edx,DWORD PTR gs:0x14
   0x0804849d <+50>:    je     0x80484a4 <main+57>
   0x0804849f <+52>:    call   0x8048340 <__stack_chk_fail@plt>
   0x080484a4 <+57>:    leave
   0x080484a5 <+58>:    ret
```



5. EDX에 스택 Canary값 저장

GS Segment

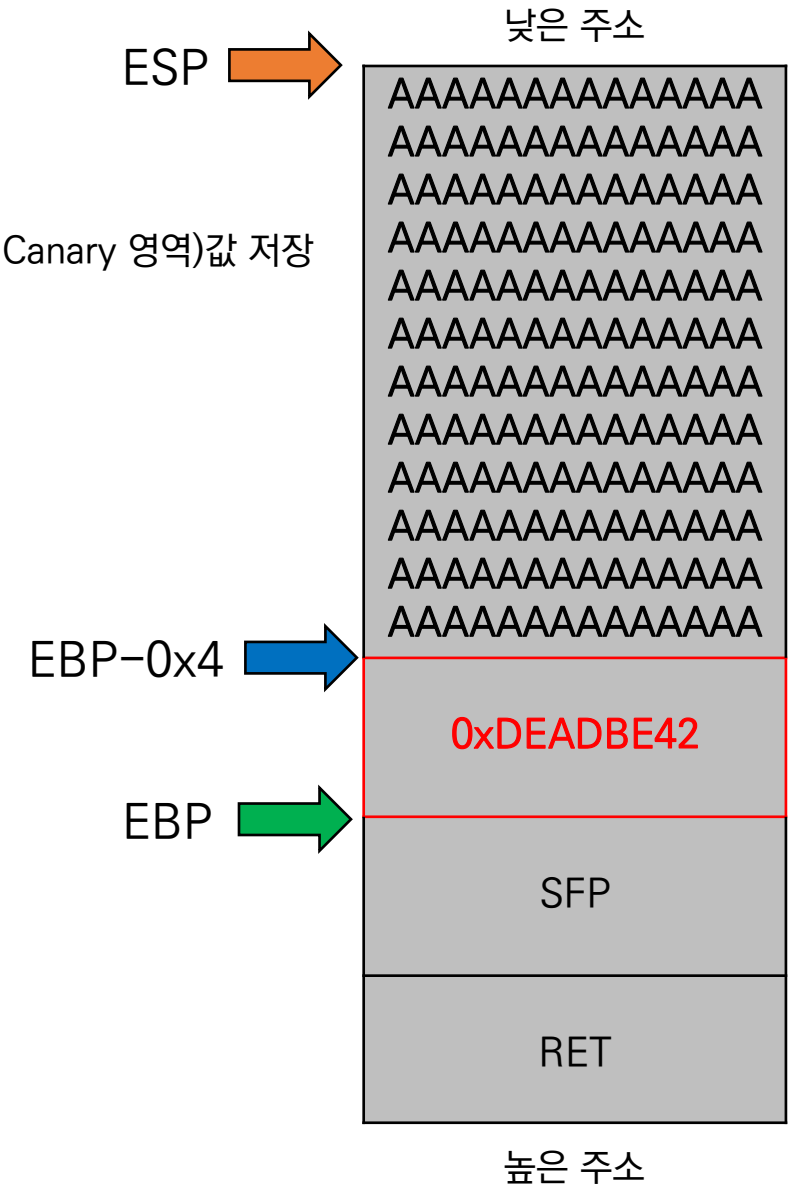


EDX

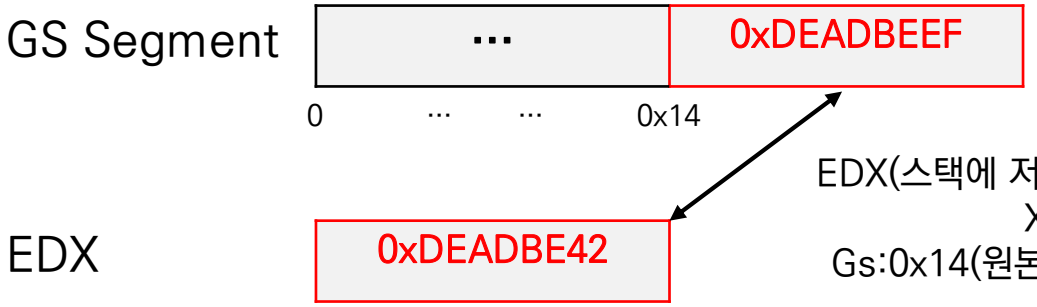
0xDEADBE42

EDX에 [EBP-0x4](저장했던 Canary 영역)값 저장

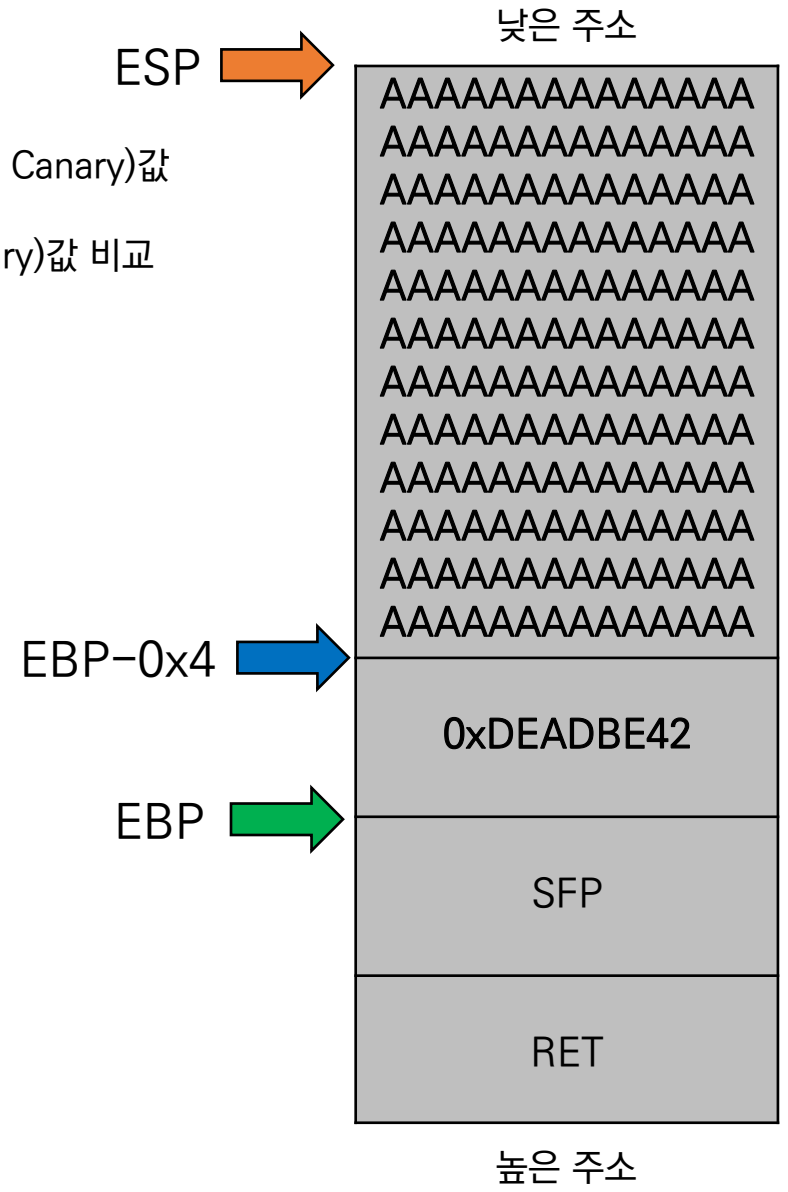
```
gdb-peda$ pd main
Dump of assembler code for function main:
   0x0804846b <+0>:     push   ebp
   0x0804846c <+1>:     mov    ebp,esp
   0x0804846e <+3>:     sub    esp,0x48
   0x08048471 <+6>:     mov    eax,DWORD PTR [ebp+0xc]
   0x08048474 <+9>:     mov    DWORD PTR [ebp-0x48],eax
   0x08048477 <+12>:    mov    eax,gs:0x14
   0x0804847d <+18>:    mov    DWORD PTR [ebp-0x4],eax
   0x08048480 <+21>:    xor    eax,eax
   0x08048482 <+23>:    lea    eax,[ebp-0x44]
   0x08048485 <+26>:    push   eax
   0x08048486 <+27>:    call   0x8048330 <gets@plt>
   0x0804848b <+32>:    add    esp,0x4
   0x0804848e <+35>:    mov    eax,0x0
   0x08048493 <+40>:    mov    edx,DWORD PTR [ebp-0x4]
   0x08048496 <+43>:    xor    edx,DWORD PTR gs:0x14
   0x0804849d <+50>:    je     0x80484a4 <main+57>
   0x0804849f <+52>:    call   0x8048340 <__stack_chk_fail@plt>
   0x080484a4 <+57>:    leave
   0x080484a5 <+58>:    ret
```



6. 스택 Canary 무결성 검사

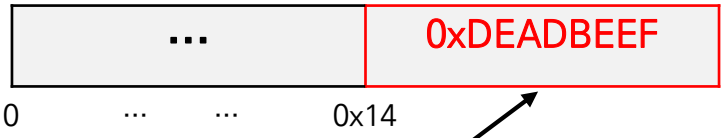



```
gdb-peda$ pd main
Dump of assembler code for function main:
   0x0804846b <+0>:     push   ebp
   0x0804846c <+1>:     mov    ebp,esp
   0x0804846e <+3>:     sub    esp,0x48
   0x08048471 <+6>:     mov    eax,DWORD PTR [ebp+0xc]
   0x08048474 <+9>:     mov    DWORD PTR [ebp-0x48],eax
   0x08048477 <+12>:    mov    eax,gs:0x14
   0x0804847d <+18>:    mov    DWORD PTR [ebp-0x4],eax
   0x08048480 <+21>:    xor    eax,eax
   0x08048482 <+23>:    lea    eax,[ebp-0x44]
   0x08048485 <+26>:    push   eax
   0x08048486 <+27>:    call   0x8048330 <gets@plt>
   0x0804848b <+32>:    add    esp,0x4
   0x0804848e <+35>:    mov    eax,0x0
   0x08048493 <+40>:    mov    edx,DWORD PTR [ebp-0x4]
   0x08048496 <+43>:    xor    edx,DWORD PTR gs:0x14
   0x0804849d <+50>:    je     0x80484a4 <main+57>
   0x0804849f <+52>:    call   0x8048340 <__stack_chk_fail@plt>
   0x080484a4 <+57>:    leave
   0x080484a5 <+58>:    ret
```



7. 비교값에 따른 분기

GS Segment

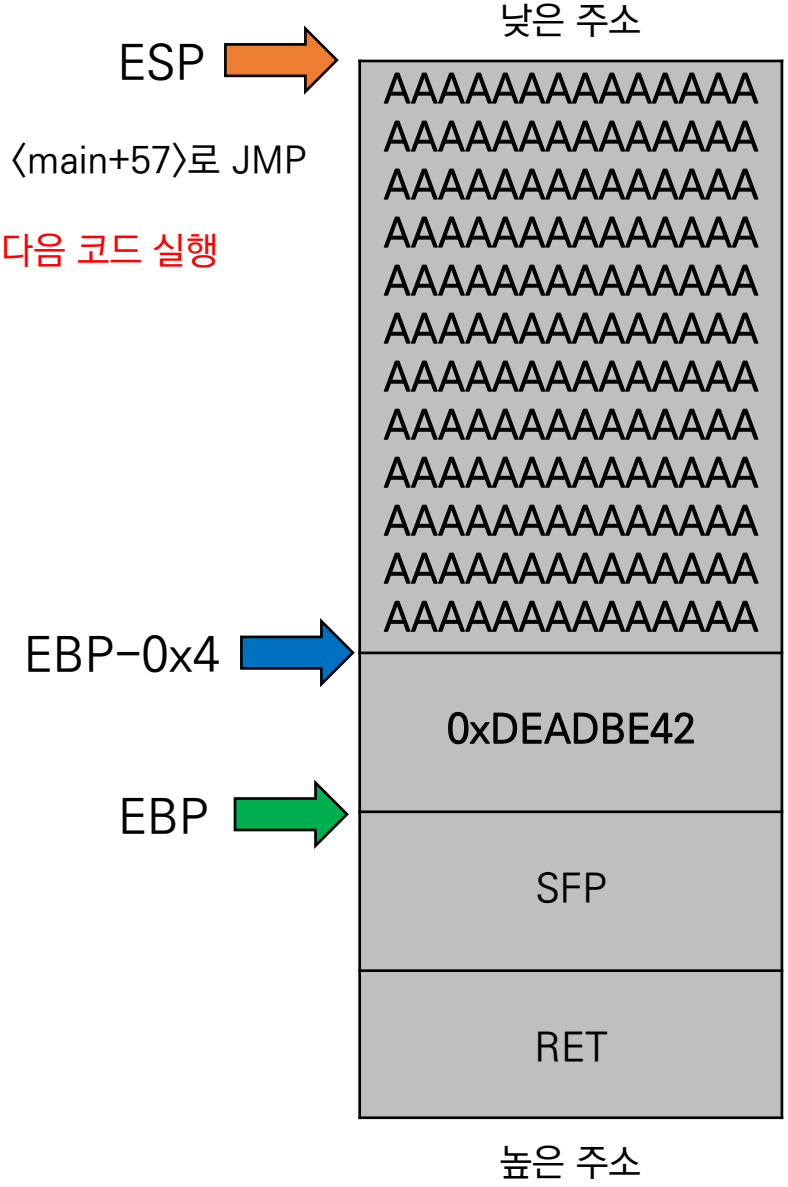


EDX

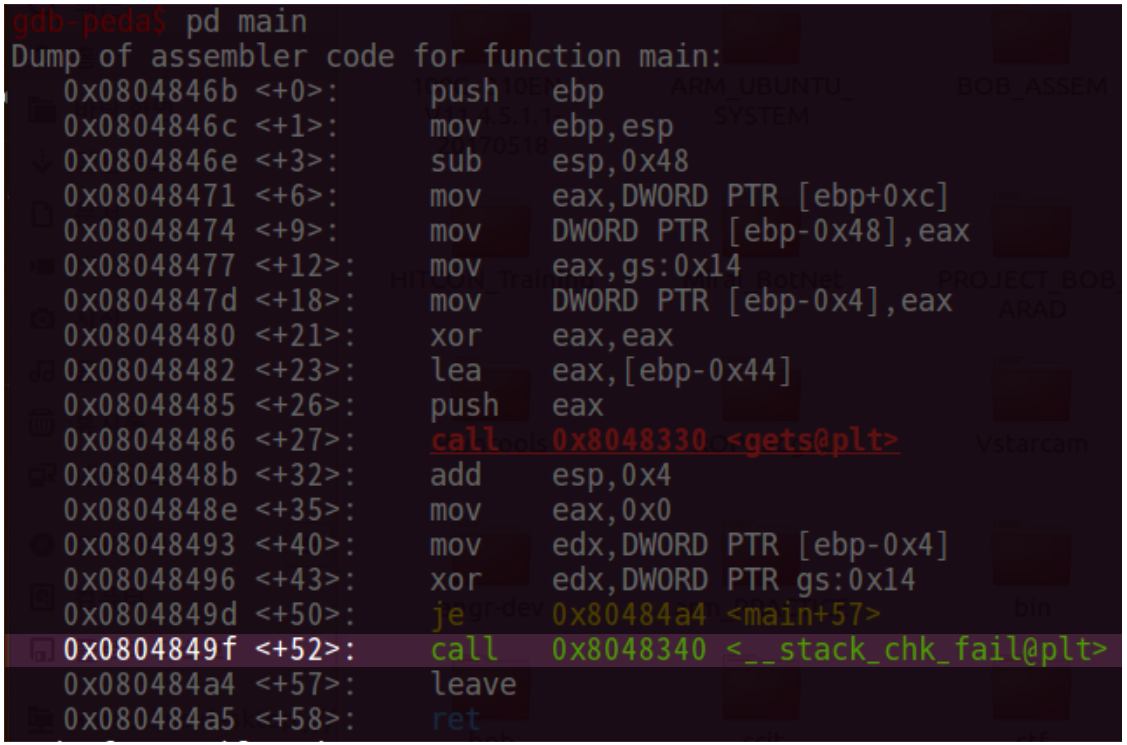
0xDEADBE42

- XOR 결과값 == 0이면, <main+57>로 JMP
- XOR 결과값 != 0이면, 다음 코드 실행

```
gdb-peda$ pd main
Dump of assembler code for function main:
   0x0804846b <+0>:     push   ebp
   0x0804846c <+1>:     mov    ebp,esp
   0x0804846e <+3>:     sub    esp,0x48
   0x08048471 <+6>:     mov    eax,DWORD PTR [ebp+0xc]
   0x08048474 <+9>:     mov    DWORD PTR [ebp-0x48],eax
   0x08048477 <+12>:    mov    eax,gs:0x14
   0x0804847d <+18>:    mov    DWORD PTR [ebp-0x4],eax
   0x08048480 <+21>:    xor    eax,eax
   0x08048482 <+23>:    lea    eax,[ebp-0x44]
   0x08048485 <+26>:    push   eax
   0x08048486 <+27>:    call   0x8048330 <gets@plt>
   0x0804848b <+32>:    add    esp,0x4
   0x0804848e <+35>:    mov    eax,0x0
   0x08048493 <+40>:    mov    edx,DWORD PTR [ebp-0x4]
   0x08048496 <+43>:    xor    edx,DWORD PTR gs:0x14
   0x0804849d <+50>:    je     0x80484a4 <main+57>
   0x0804849f <+52>:    call   0x8048340 <__stack_chk_fail@plt>
   0x080484a4 <+57>:    leave
   0x080484a5 <+58>:    ret
```



8. __stack_chk_fail( ) 실행




```
*** stack smashing detected ***: ./canary terminated
중지됨
```

Stack Canary값이 원본 값 이랑 다를 경우,
“Stack smashing detected” 오류 메시지를 출력하고 바이너리 종료

그럼 Canary는 어떻게 우회할 수 있을까?

Canary의 한계점이 존재!

한계점을 이용해서 Canary를 우회할 수 있다.

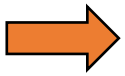
Canary 한계점

1. 고정적인 Canary 값

2. Terminator Canary의 존재

3. StackGuard v2.0.1에서는 정적 Canary를 사용한다.

Canary 한계점



1. 고정적인 Canary 값

2. Terminator Canary의 존재

3. StackGuard v2.0.1에서는 정적 Canary를 사용한다.

1. 고정적인 Canary 값

```
v6 = sub_80488CB;
sigemptyset((sigset_t *)&v7);
v8 = 0;
v2 = sigaction(17, (const struct sigaction *)&v6, 0);
if ( v2 )
  sub_804889D("sigaction error");
v3 = socket(2, 1, 0);
memset(&s, 0, 0x10u);
*(_WORD *)&s = 2;
v11 = htonl(0);
v10 = htons(0x22B8u);
setsockopt(v3, 1, 2, &optval, 4u);
if ( bind(v3, (const struct sockaddr *)&s, 0x10u) == -1 )
  sub_804889D("bind() error");
if ( listen(v3, 10) == -1 )
  sub_804889D("listen() error");
while ( 1 )
{
  do
  {
    addr_len = 16;
    v4 = accept(v3, &addr, &addr_len);
  }
  while ( v4 == -1 );
  v5 = fork();
  if ( v5 == -1 )
  {
    close(v4);
  }
  else
  {
    if ( v5 <= 0 )
    {
      close(v3);
      print menu(v4);
      close(v4);
      exit(0);
    }
    close(v4);
```

부모 프로세스는 Accept( )로 Client연결만 관리하고

연결되는 Client들은 fork( )로 복제한 자식 프로세스로 묶어준다.

1. 고정적인 Canary 값

```
v6 = sub_80488CB;
sigemptyset((sigset_t *)&v7);
v8 = 0;
v2 = sigaction(17, (const struct sigaction *)&v6, 0);
if ( v2 )
  sub_804889D("sigaction error");
v3 = socket(2, 1, 0);
memset(&s, 0, 0x10u);
*(_WORD *)&s = 2;
v11 = htonl(0);
v10 = htons(0x22B8u);
setsockopt(v3, 1, 2, &optval, 4u);
if ( bind(v3, (const struct sockaddr *)&s, 0x10u) == -1 )
  sub_804889D("bind() error");
if ( listen(v3, 10) == -1 )
  sub_804889D("listen() error");
while ( 1 )
{
  do
  {
    addr_len = 16;
    v4 = accept(v3, &addr, &addr_len);
  }
  while ( v4 == -1 );
  v5 = fork();
  if ( v5 == -1 )
  {
    close(v4);
  }
  else
  {
    if ( v5 <= 0 )
    {
      close(v3);
      print menu(v4);
      close(v4);
      exit(0);
    }
    close(v4);
```

부모 프로세스는 Accept( )로 Client연결만 관리하고

연결되는 Client들은 fork( )로 복제한 자식 프로세스로 묶어준다.

fork( )는 부모 프로세스를 그대로 복제해서
자식 프로세스를 만들어주는 함수다.

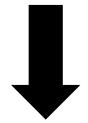
1. 고정적인 Canary 값

```
v6 = sub_80488CB;
sigemptyset((sigset_t *)&v7);
v8 = 0;
v2 = sigaction(17, (const struct sigaction *)&v6, 0);
if ( v2 )
  sub_804889D("sigaction error");
v3 = socket(2, 1, 0);
memset(&s, 0, 0x10u);
*(_WORD *)&s = 2;
v11 = htonl(0);
v10 = htons(0x22B8u);
setsockopt(v3, 1, 2, &optval, 4u);
if ( bind(v3, (const struct sockaddr *)&s, 0x10u) == -1 )
  sub_804889D("bind() error");
if ( listen(v3, 10) == -1 )
  sub_804889D("listen() error");
while ( 1 )
{
  do
  {
    addr_len = 16;
    v4 = accept(v3, &addr, &addr_len);
  }
  while ( v4 == -1 );
  v5 = fork();
  if ( v5 == -1 )
  {
    close(v4);
  }
  else
  {
    if ( v5 <= 0 )
    {
      close(v3);
      print_menu(v4);
      close(v4);
      exit(0);
    }
    close(v4);
```

부모 프로세스는 Accept( )로 Client연결만 관리하고

연결되는 Client들은 fork( )로 복제한 자식 프로세스로 묶어준다.

fork( )는 부모 프로세스를 그대로 복제해서
자식 프로세스를 만들어주는 함수다.



부모 프로세스의 Canary값도 그대로 복제된다.

1. 고정적인 Canary 값

```
v6 = sub_80488CB;
sigemptyset((sigset_t *)&v7);
v8 = 0;
v2 = sigaction(17, (const struct sigaction *)&v6, 0);
if ( v2 )
  sub_804889D("sigaction error");
v3 = socket(2, 1, 0);
memset(&s, 0, 0x10u);
*(_WORD *)&s = 2;
v11 = htonl(0);
v10 = htons(0x22B8u);
setsockopt(v3, 1, 2, &optval, 4u);
if ( bind(v3, (const struct sockaddr *)&s, 0x10u) == -1 )
  sub_804889D("bind() error");
if ( listen(v3, 10) == -1 )
  sub_804889D("listen() error");
while ( 1 )
{
  do
  {
    addr_len = 16;
    v4 = accept(v3, &addr, &addr_len);
  }
  while ( v4 == -1 );
  v5 = fork();
  if ( v5 == -1 )
  {
    close(v4);
  }
  else
  {
    if ( v5 <= 0 )
    {
      close(v3);
      print menu(v4);
      close(v4);
      exit(0);
    }
    close(v4);
```

```
   0x804935e:   sub    esp,0xf0
   0x8049364:   mov    eax,gs:0x14
   0x804936a:   mov    DWORD PTR [esp+0xec],eax
=> 0x8049371:   xor    eax,eax
   0x8049373:   mov    DWORD PTR [esp+0x28],0x1
   0x804937b:   mov    DWORD PTR [esp+0x40],0x80
   0x8049383:   lea    eax,[esp+0x40]
   0x8049387:   add    eax,0x4
[----------------------------------stack-----
0000| 0xffffcd00 --> 0x0
0004| 0xffffcd04 --> 0x0
0008| 0xffffcd08 --> 0xf7ffd000 --> 0x23f3c
0012| 0xffffcd0c --> 0xf7ffdc08 --> 0xf7fd8000 (
0016| 0xffffcd10 --> 0x0
0020| 0xffffcd14 --> 0x0
0024| 0xffffcd18 --> 0x0
0028| 0xffffcd1c --> 0xffffcdac --> 0xf7fd41b0 -
[----------------------------------------------
Legend: code, data, rodata, value

Breakpoint 1, 0x08049371 in ?? ()
gdb-peda$ x/x $esp+0xec
0xffffcdec:     0x21026800
```

fork( ) 실행 전, 부모 프로세스의 Canary값: **0x21026800**

1. 고정적인 Canary 값

```
v6 = sub_80488CB;
sigemptyset((sigset_t *)&v7);
v8 = 0;
v2 = sigaction(17, (const struct sigaction *)&v6, 0);
if ( v2 )
  sub_804889D("sigaction error");
v3 = socket(2, 1, 0);
memset(&s, 0, 0x10u);
*(_WORD *)&s = 2;
v11 = htonl(0);
v10 = htons(0x22B8u);
setsockopt(v3, 1, 2, &optval, 4u);
if ( bind(v3, (const struct sockaddr *)&s, 0x10u) == -1 )
  sub_804889D("bind() error");
if ( listen(v3, 10) == -1 )
  sub_804889D("listen() error");
while ( 1 )
{
  do
  {
    addr_len = 16;
    v4 = accept(v3, &addr, &addr_len);
  }
  while ( v4 == -1 );
  v5 = fork();
  if ( v5 == -1 )
  {
    close(v4);
  }
  else
  {
    if ( v5 <= 0 )
    {
      close(v3);
      print menu(v4);
      close(v4);
      exit(0);
    }
    close(v4);
```

```
   0x8049204:   sub    esp,0x28
   0x8049207:   mov    eax,gs:0x14
   0x804920d:   mov    DWORD PTR [ebp-0xc],eax
=> 0x8049210:   xor    eax,eax
   0x8049212:   mov    eax,DWORD PTR [ebp+0x8]
   0x8049215:   mov    DWORD PTR [esp],eax
   0x8049218:   call   0x8048909
   0x804921d:   mov    DWORD PTR [esp+0x8],0x15
[----------------------------------stack-----
0000| 0xffffccd0 --> 0xf7fe77eb (<_dl_fixup+11>
0004| 0xffffccd4 --> 0x0
0008| 0xffffccd8 --> 0xf7fb1000 --> 0x1b1db0
0012| 0xffffccdc --> 0xf7fb1000 --> 0x1b1db0
0016| 0xffffcce0 --> 0xffffcdf8 --> 0x0
0020| 0xffffcce4 --> 0xf7fee010 (<_dl_runtime_r
0024| 0xffffcce8 --> 0xfa82
0028| 0xffffccec --> 0x21026800
[----------------------------------------------
Legend: code, data, rodata, value

Thread 2.1 "angrydoraemon" hit Breakpoint 2, 0x
gdb-peda$ x/x $ebp-0xc
0xffffccec:     0x21026800
```

fork( ) 실행 전, 부모 프로세스의 Canary값: **0x21026800**

fork( ) 실행 후, 자식 프로세스의 Canary값: **0x21026800**

1. 고정적인 Canary 값

```
v6 = sub_80488CB;
sigemptyset((sigset_t *)&v7);
v8 = 0;
v2 = sigaction(17, (const struct sigaction *)&v6, 0);
if ( v2 )
  sub_804889D("sigaction error");
v3 = socket(2, 1, 0);
memset(&s, 0, 0x10u);
*(_WORD *)&s = 2;
v11 = htonl(0);
v10 = htons(0x22B8u);
setsockopt(v3, 1, 2, &optval, 4u);
if ( bind(v3, (const struct sockaddr *)&s, 0x10u) == -1 )
  sub_804889D("bind() error");
if ( listen(v3, 10) == -1 )
  sub_804889D("listen() error");
while ( 1 )
{
  do
  {
    addr_len = 16;
    v4 = accept(v3, &addr, &addr_len);
  }
  while ( v4 == -1 );
  v5 = fork();
  if ( v5 == -1 )
  {
    close(v4);
  }
  else
  {
    if ( v5 <= 0 )
    {
      close(v3);
      print_menu(v4);
      close(v4);
      exit(0);
    }
    close(v4);
```

```
   0x8049204:   sub    esp,0x28
   0x8049207:   mov    eax,gs:0x14
   0x804920d:   mov    DWORD PTR [ebp-0xc],eax
=> 0x8049210:   xor    eax,eax
   0x8049212:   mov    eax,DWORD PTR [ebp+0x8]
   0x8049215:   mov    DWORD PTR [esp],eax
   0x8049218:   call   0x8048909
   0x804921d:   mov    DWORD PTR [esp+0x8],0x15
[------------------------------------stack-----
0000| 0xffffccd0 --> 0xf7fe77eb (<_dl_fixup+11>
0004| 0xffffccd4 --> 0x0
0008| 0xffffccd8 --> 0xf7fb1000 --> 0x1b1db0
0012| 0xffffccdc --> 0xf7fb1000 --> 0x1b1db0
0016| 0xffffcce0 --> 0xffffcdf8 --> 0x0
0020| 0xffffcce4 --> 0xf7fee010 (<_dl_runtime_r
0024| 0xffffcce8 --> 0xfa82
0028| 0xffffccec --> 0x21026800
[------------------------------------------------
Legend: code, data, rodata, value

Thread 2.1 "angrydoraemon" hit Breakpoint 2, 0x
gdb-peda$ x/x $ebp-0xc
0xffffccec:     0x21026800
```

실습!

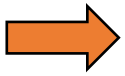
Codegate 2014
Angrydoraemon
/
picoCTF 2019
Canary

fork( ) 실행 전, 부모 프로세스의 Canary값: 0x21026800

fork( ) 실행 후, 자식 프로세스의 Canary값: 0x21026800

Canary 한계점

1. 고정적인 Canary 값

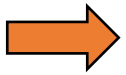


2. Terminator Canary의 존재

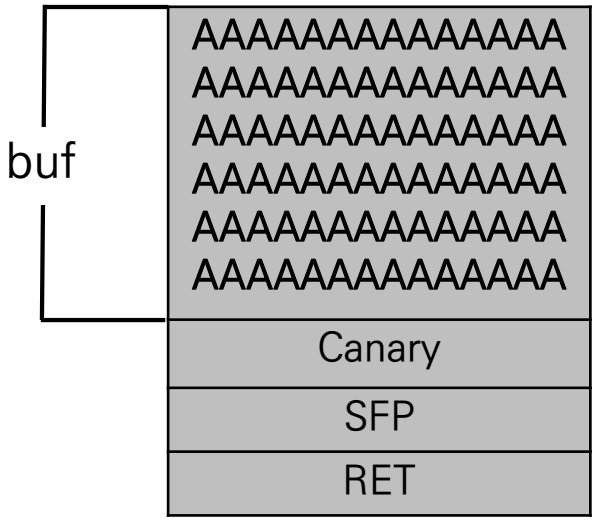
3. StackGuard v2.0.1에서는 정적 Canary를 사용한다.

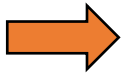
2. Terminator Canary의 존재

```
printf("%s",buf);
```



%s는 Null값을 만날 때 까지 출력한다.

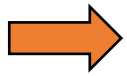




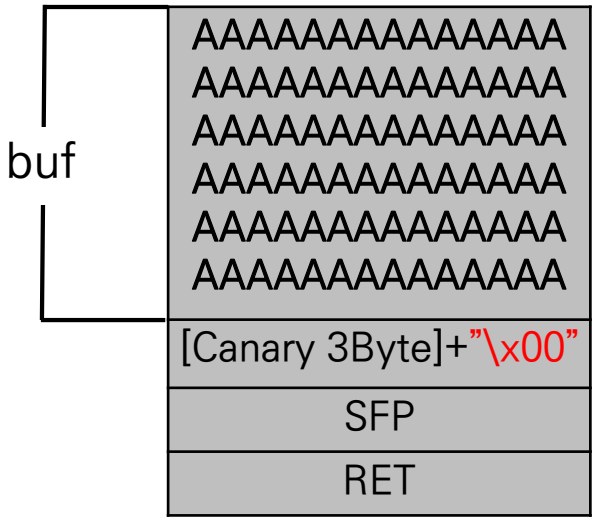
Canary까지 출력된다.

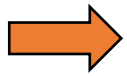
2. Terminator Canary의 존재

```
printf("%s",buf);
```



%s는 Null값을 만날 때 까지 출력한다.

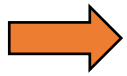




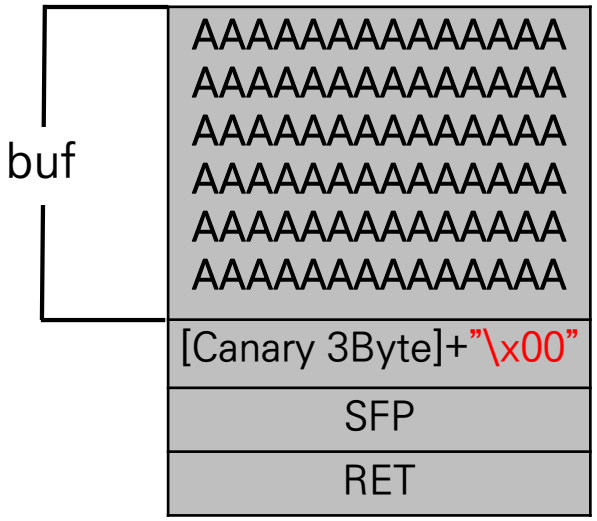
Canary의 하위1Byte를 “\x00”로 고정해서
앞과 같은 상황이 생겨도 Canary가 출력되지 않게 한다.

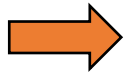
2. Terminator Canary의 존재

```
printf("%s",buf);
```



%s는 Null값을 만날 때 까지 출력한다.



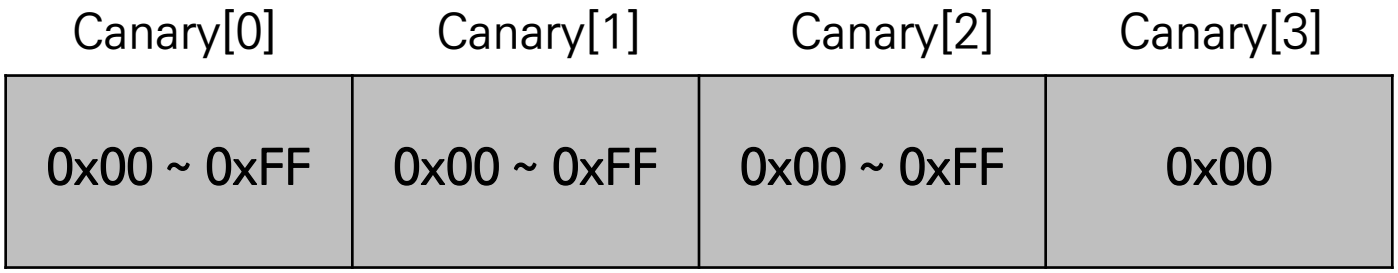


Canary의 하위1Byte를 "\x00"로 고정해서
앞과 같은 상황이 생겨도 Canary가 출력되지 않게 한다.

2. Terminator Canary의 존재

Canary의 하위1Byte를 “\x00”로 고정된다.

1번 상황과 같이 fork( )로 Canary값이 변하지 않을 때



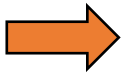
경우의 수 = 256 x 3

32bit환경

Canary 한계점

1. 네트워크 데몬들은 대부분 fork( )로 Client연결을 받는다.

2. Terminator Canary의 존재



3. StackGuard v2.0.1에서는 정적 Canary를 사용한다.

3. StackGuard v2.0.1에서는 정적 Canary를 사용한다.

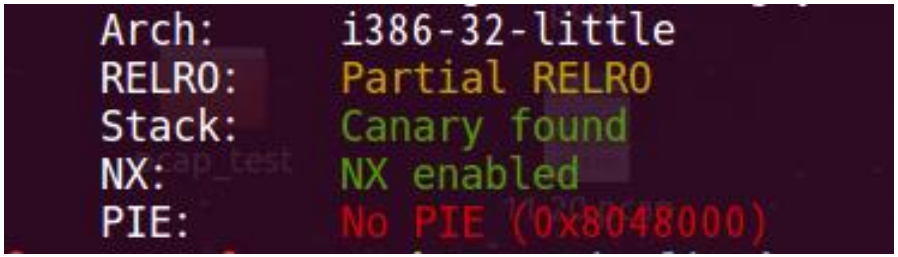
[0x000AFF0D] 이라는 고정된 Canary 값을 사용했었다.

0x00 – 문자열 인자의 복사를 막기 위함
0x0A – 띄어쓰기로 gets( )와 같은 함수들의 읽기를 막기 위함
0xFF / 0x0D – 이것들도 때때로 문자열 복사를 막는다.

Memory leak과 Canary 값 복사를 방지하기 위함 이였다.

QnA?

우리는 이제…



이런 문제들을 풀 수 있는 수준이 되었습니다.