03.Canaries

List

Canaries

- Canaries 또는 Canary word는 버퍼 오버 플로우를 모니터하기 위해 버퍼와 제어 데이터 사이에 설정 된 값입니다.
- 버퍼 오버플로가 발생하면 Canary 값이 손상되며, Canaries 데이터의 검증에 실패하여, 오버플로에 대한 경고가 출력되고, 손상된 데이터를 무효화 처리됩니다.

Types of canaries

Terminator canaries

- Terminator Canaries는 Canary의 값을 문자열의 끝을 나타내는 문자들을 이용해 생성합니다.
- Terminator Canaries의 값은 NULL (0x00), CR (0x0d), LF (0x0a) 및 EOF (0xff)로 구성되어 있습니다.
  - 공격자는 Canaries를 우회하기 위해 위해 Return address를 쓰기 전에 null 문자를 써야 합니다.
  - null문자로 인해 Overflow를 방지하게 됩니다.
    - strcpy()는 null문자의 위치까지 복사합니다.
  - 이 보호에도 불구하고 공격자는 잠재적으로 Canary를 알려진 값으로 겹쳐쓰고 정보를 틀린 값들로 제어해서 Canary 검사 코드를 통과할 수 있다.

Random canaries

- Random Canaries는 Canary의 값을 랜덤하게 값이 생성합니다.
  - 일반적으로 익스플로잇을 이용해 Canary를 읽는 것은 논리적으로 불가능하다.
- Random Canaries는 프로그램 초기 설정 시에 전역 변수에 Canary 값이 저장된다.
  - 이 값은 보통 매핑되지 않은 페이지에 저장됩니다.
    - 해당 메모리를 읽으려는 시도를 할 경우 segmentation fault가 발생하고 프로그램이 종료됩니다.
  - 공격자가 Canary 값이 저장된 stack address를 알거나 스택의 값을 읽어올수 있는 프로그램이 있다면 Canary의 값을 확인 할 수 있습니다.

Random XOR canaries

- Random XOR Canaries는 Canary의 값을 모든 제어 데이터 또는 일부를 사용해 XOR-scramble 하여 생성합니다.
  - 이 방식은 Canary의 값, 제어 데이터가 오염되면 Canary의 값이 틀려집니다.
- Random XOR Canaries는 Random Canaries와 동일한 취약점을 가지고 있습니다.
  - 단지 Canary 값을 Stack에서 읽어오는 방법이 조금더 복잡해집니다.
  - 공격자는 canary를 다시 인코딩 하기위해 Original Canary 값, 알고리즘, 제어 데티어가 필요합니다.

Example

Source code

Canary

```
#include <stdio.h>

void main(int argc, char **argv)
{
        char Overflow[32];

        printf("Hello world!\n");
        gets(Overflow);

}
```

Build command

Option

```
gcc -fstack-protector -param ssp-buffer-size=N xx.c   ==> byte 변경
gcc -fstack-protector-all xx.c                        ==> 모든 함수 보호
```

Build Command(Do not set Canary)

gcc -fstack-protector -o canary canary.c

Check to Canary

- **다음과 같이 Canary 값을 확인 할 수 있습니다.**
  - 사용자 값이 저장되는 영역은 0x7fffffffe180 입니다.
    - 해당 영역에 코드에서 할당한 길이의 문자를 저장합니다. ('A' * 32)
  - 0x400610 코드 영역에서 rax 레지스터에 rbp - 0x8 영역에 저장된 값을 저장합니다.
    - rbp(0x7fffffffe1b0) - 0x8 = 0x7fffffffe1a8
    - 0x7fffffffe1a8 영역에 저장된 값 : 0x3a3b864735c7b300
  - 0x400614 코드 영역에서 rax 레지스터에 저장된 값과 fs:0x28 레지스터에 저장된 값을 xor 연산합니다.
  - 0x40061d 코드 영역에서 rax 레지스터의 값이 0과 같으면 0x400624 영역으로 이동합니다.
  - 이로 인해 정상적으로 프로그램이 종료됩니다.

Do not overwrite the value in the canary area.

```
lazenca0x0@ubuntu:~/Documents/Definition/protection/Canary$ gdb -q ./Canary
Reading symbols from ./Canary...(no debugging symbols found)...done.
gdb-peda$ disassemble main
Dump of assembler code for function main:
   0x00000000004005d6 <+0>:        push   rbp
   0x00000000004005d7 <+1>:        mov    rbp,rsp
   0x00000000004005da <+4>:        sub    rsp,0x40
   0x00000000004005de <+8>:        mov    DWORD PTR [rbp-0x34],edi
   0x00000000004005e1 <+11>:        mov    QWORD PTR [rbp-0x40],rsi
   0x00000000004005e5 <+15>:        mov    rax,QWORD PTR fs:0x28
   0x00000000004005ee <+24>:        mov    QWORD PTR [rbp-0x8],rax
   0x00000000004005f2 <+28>:        xor    eax,eax
   0x00000000004005f4 <+30>:        mov    edi,0x4006b4
   0x00000000004005f9 <+35>:        call   0x400490 <puts@plt>
   0x00000000004005fe <+40>:        lea    rax,[rbp-0x30]
   0x0000000000400602 <+44>:        mov    rdi,rax
   0x0000000000400605 <+47>:        mov    eax,0x0
   0x000000000040060a <+52>:        call   0x4004c0 <gets@plt>
   0x000000000040060f <+57>:        nop
   0x0000000000400610 <+58>:        mov    rax,QWORD PTR [rbp-0x8]
   0x0000000000400614 <+62>:        xor    rax,QWORD PTR fs:0x28
   0x000000000040061d <+71>:        je     0x400624 <main+78>
   0x000000000040061f <+73>:        call   0x4004a0 <__stack_chk_fail@plt>
```

```
   0x0000000000400624 <+78>:        leave
   0x0000000000400625 <+79>:        ret
End of assembler dump.
gdb-peda$ b *0x000000000040060a
Breakpoint 1 at 0x40060a
gdb-peda$ b *0x0000000000400610
Breakpoint 2 at 0x400610
gdb-peda$ r
Starting program: /home/lazenca0x0/Documents/Definition/protection/Canary/Canary
Hello world!


Breakpoint 1, 0x000000000040060a in main ()
gdb-peda$ i r rdi
rdi            0x7fffffffe180        0x7fffffffe180
gdb-peda$ ni
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
0x000000000040060f in main ()
gdb-peda$ x/10gx 0x7fffffffe180
0x7fffffffe180:        0x4141414141414141        0x4141414141414141
0x7fffffffe190:        0x4141414141414141        0x4141414141414141
0x7fffffffe1a0:        0x00007fffffffe200        0x3a3b864735c7b300
0x7fffffffe1b0:        0x0000000000400630        0x00007ffff7a2d830
0x7fffffffe1c0:        0x0000000000000000        0x00007fffffffe298
gdb-peda$ c
Continuing.


Breakpoint 2, 0x0000000000400610 in main ()
gdb-peda$ i r rbp
rbp            0x7fffffffe1b0        0x7fffffffe1b0
gdb-peda$ x/gx 0x7fffffffe1b0 - 0x8
0x7fffffffe1a8:        0x3a3b864735c7b300
gdb-peda$ ni


0x0000000000400614 in main ()
gdb-peda$ i r rax
rax            0x3a3b864735c7b300        0x3a3b864735c7b300
gdb-peda$ ni
0x000000000040061d in main ()
gdb-peda$ i r rax
rax            0x0        0x0
gdb-peda$ ni

0x0000000000400624 in main ()
gdb-peda$ x/2i $rip
=> 0x400624 <main+78>:        leave
   0x400625 <main+79>:        ret
gdb-peda$
```

- **다음과 같이 Canary 값을 덮어썻을 경우의 프로그램 동작을 확인 할 수 있습니다.**
  - 사용자 입력 값이 저장되는 위치와 Canary의 위치는 앞에서 설명한 것과 동일합니다.
  - 사용자 입력 값으로 'A' * 40 + 'B' * 8 을 입력합니다.
    - 해당 값으로 인해 canary의 값이 0x4242424242424242(BBBBBBBB) 으로 변경되었습니다.
  - 0x400610 코드 영역에서 rax 레지스터에 rbp - 0x8 영역에 저장된 값을 저장합니다.
    - rbp(0x7fffffffe1b0) - 0x8 = 0x7fffffffe1a8
    - 0x7fffffffe1a8 영역에 저장된 값 : 0x4242424242424242
  - 0x400614 코드 영역에서 rax 레지스터에 저장된 값과 fs:0x28 레지스터에 저장된 값을 xor 연산합니다.
  - 0x40061d 코드 영역에서 rax 레지스터의 값이 0x61061c8ecf993242 이기 때문에 다음 코드 영역(0x40061f)으로 이동합니다.
  - 이로 인해 프로그램에서 아래와 같은 Error 메시지를 출력합니다.
    - "stack smashing detected"

The value is overwritten in the canary area.

```
gdb-peda$ r
Starting program: /home/lazenca0x0/Documents/Definition/protection/Canary/Canary
Hello world!
Breakpoint 1, 0x000000000040060a in main ()
gdb-peda$ i r rdi
rdi            0x7fffffffe180        0x7fffffffe180
gdb-peda$ ni
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBB
0x000000000040060f in main ()
gdb-peda$ x/10gx 0x7fffffffe180
0x7fffffffe180:        0x4141414141414141        0x4141414141414141
0x7fffffffe190:        0x4141414141414141        0x4141414141414141
0x7fffffffe1a0:        0x4141414141414141        0x4242424242424242
0x7fffffffe1b0:        0x0000000000400600        0x00007ffff7a2d830
0x7fffffffe1c0:        0x0000000000000000        0x00007fffffffe298
gdb-peda$ c
Continuing.


Breakpoint 2, 0x0000000000400610 in main ()
gdb-peda$ i r rbp
rbp            0x7fffffffe1b0        0x7fffffffe1b0
gdb-peda$ x/gx 0x7fffffffe1b0 - 0x8
0x7fffffffe1a8:        0x4242424242424242
gdb-peda$ ni


0x0000000000400614 in main ()
gdb-peda$ i r rax
rax            0x4242424242424242        0x4242424242424242
gdb-peda$ ni
0x000000000040061d in main ()
gdb-peda$ i r rax
rax            0x61061c8ecf993242        0x61061c8ecf993242
gdb-peda$ ni

0x000000000040061f in main ()
gdb-peda$ x/3i $rip
=> 0x40061f <main+73>:        call   0x4004a0 <__stack_chk_fail@plt>
   0x400624 <main+78>:        leave
   0x400625 <main+79>:        ret
gdb-peda$ c
Continuing.
*** stack smashing detected ***: /home/lazenca0x0/Documents/Definition/protection/Canary/Canary terminated

Program received signal SIGABRT, Aborted.
```

Check the protection techniques of binary files.

checksec.sh

- Checksec.sh에서 다음과 같은 결과를 출력합니다.
    - Canary_Do-not-set: No canary found
    - Canary: Canary found

Not set Canary	gcc -fstack-protector -o Canary_Do-not-set Canary.c lazenca0x0@ubuntu:~/Documents/Definition/protection/Canary$ checksec.sh --file ./Canary_Do-not-set RELRO STACK CANARY NX PIE RPATH RUNPATH FILE Partial RELRO No canary found NX enabled No PIE No RPATH No RUNPATH ./Canary_Do-not-set lazenca0x0@ubuntu:~/Documents/Definition/protection/Canary$
Set Canary	gcc -o Canary Canary.c lazenca0x0@ubuntu:~/Documents/Definition/protection/Canary$ checksec.sh --file ./Canary RELRO STACK CANARY NX PIE RPATH RUNPATH FILE Partial RELRO Canary found NX enabled No PIE No RPATH No RUNPATH ./Canary lazenca0x0@ubuntu:~/Documents/Definition/protection/Canary$

How to detect Canary in the "Checksec.sh" file

Binary

- **다음과 같은 방법으로 바이너리의 Canary 설정여부를 확인합니다.**
  - 'readelf' 명령어를 이용해 해당 파일의 심볼 테이블 정보를 가져와 Canary 설정여부를 확인합니다.
  - 파일의 심볼 테이블에 "__stack_chk_fail"가 있으면 Canary가 적용되었다고 판단합니다.

Checksec.sh - line 156

```
  # check for stack canary support
  if readelf -s $1 2>/dev/null | grep -q '__stack_chk_fail'; then
    echo -n -e '\033[32mCanary found   \033[m   '
  else
    echo -n -e '\033[31mNo canary found\033[m   '
  fi
```

readelf -s ./Canary_Do-not-set |grep __stack_chk_fail

```
lazenca0x0@ubuntu:~/Documents/Definition/protection/Canary$ readelf -s ./Canary_Do-not-set |grep
__stack_chk_fail
lazenca0x0@ubuntu:~/Documents/Definition/protection/Canary$
```

readelf -s ./Canary |grep __stack_chk_fail

```
lazenca0x0@ubuntu:~/Documents/Definition/protection/Canary$ readelf -s ./Canary |grep __stack_chk_fail
     2: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __stack_chk_fail@GLIBC_2.4 (3)
    54: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __stack_chk_fail@@GLIBC_2
lazenca0x0@ubuntu:~/Documents/Definition/protection/Canary$
```

Process

- **다음과 같은 방법으로 프로세서의 Canary 설정여부를 확인합니다.**
  - Binary의 확인 방식과 비슷하며, 전달되는 파일의 경로가 다음과 같이 다릅니다.
    - Ex) /proc/<PID>/exe
  - 추가된 동작은 '/proc/<PID>/exe' 파일에 'Symbol table' 정보가 있는지 확인합니다.

Checksec.sh - line 215

```
# check for stack canary support
if readelf -s $1/exe 2>/dev/null | grep -q 'Symbol table'; then
  if readelf -s $1/exe 2>/dev/null | grep -q '__stack_chk_fail'; then
    echo -n -e '\033[32mCanary found         \033[m  '
  else
    echo -n -e '\033[31mNo canary found      \033[m  '
  fi
else
  if [ "$1" != "1" ] ; then
    echo -n -e '\033[33mPermission denied    \033[m  '
  else
    echo -n -e '\033[33mNo symbol table found\033[m  '
  fi
fi
```

readelf -s /proc/12602/exe |grep '__stack_chk_fail'

```
lazenca0x0@ubuntu:~/Documents/Definition/protection/Canary$ ./Canary
Hello world!
^Z
[1]+  Stopped                 ./Canary
lazenca0x0@ubuntu:~/Documents/Definition/protection/Canary$ ps -ef|grep Canary
lazenca+  12602  11197  0 01:21 pts/4    00:00:00 ./Canary
lazenca+  12604  11197  0 01:21 pts/4    00:00:00 grep --color=auto Canary
lazenca0x0@ubuntu:~/Documents/Definition/protection/Canary$ readelf -s /proc/12602/exe |grep 'Symbol table'
Symbol table '.dynsym' contains 6 entries:
Symbol table '.symtab' contains 70 entries:
lazenca0x0@ubuntu:~/Documents/Definition/protection/Canary$ readelf -s /proc/12602/exe |grep '__stack_chk_fail'
     2: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __stack_chk_fail@GLIBC_2.4 (3)
    54: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __stack_chk_fail@@GLIBC_2
lazenca0x0@ubuntu:~/Documents/Definition/protection/Canary$
```

Related information

- https://en.wikipedia.org/wiki/Buffer_overflow_protection