CWL
CyberWarFare Labs

How Mitigations Work Against Stack-Based Overflows

About CyberWarFare Labs :

CW Labs is a renowned UK based Ed-tech company specializing in cybersecurity cyber range labs.

They provide on-demand educational services and recognize the need for continuous adaptation to evolving threats and client requirements.

The company has two primary divisions :

1. Cyber Range Labs

2. Up-Skilling Platform

aws
Cloud Security
RED TEAM
PURPLE TEAM
BLUE TEAM
AD
WEB
On-Premise Security
CI/CD
PLATFORM
NETWORK

INFINITE LEARNING EXPERIENCE

John Sherchan

Red Team Security Researcher at CW Labs

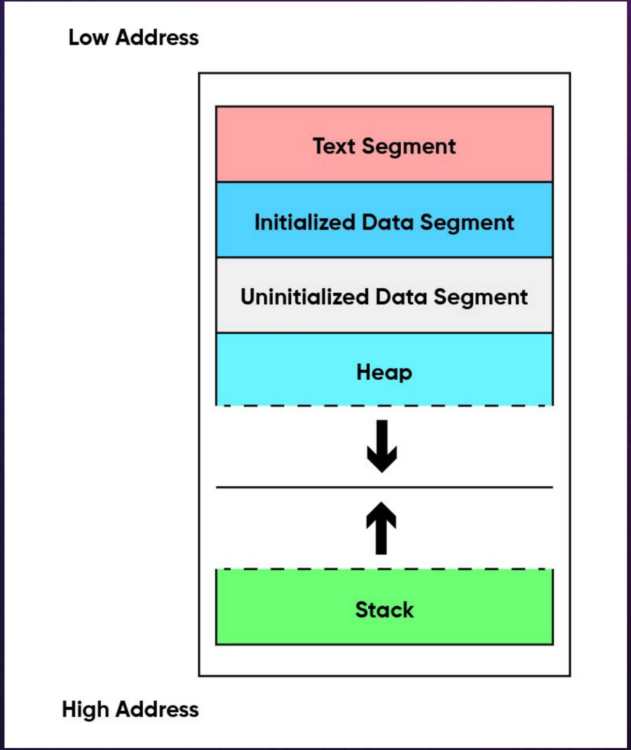
He is a Red Team Security researcher, bringing over 5+ years of experience in Reverse Engineering, Malware Analysis/Development, and Source Code Reviewing, with a specialization in Windows Internals (User and Kernel Modes). Demonstrating an advanced understanding, he has successfully reversed multiple Antivirus (AV) and Endpoint Detection and Response (EDR) systems to comprehend its architecture. Committed to advancing cybersecurity, his additional interests include PWNing Active Directory, conducting Adversary emulation/simulation, writing rootkits, crafting exploits, and strategically overcoming challenges.

Agenda

- Memory Layout (LINUX)
- Stack
- Stack Based Overflow
- Mitigations

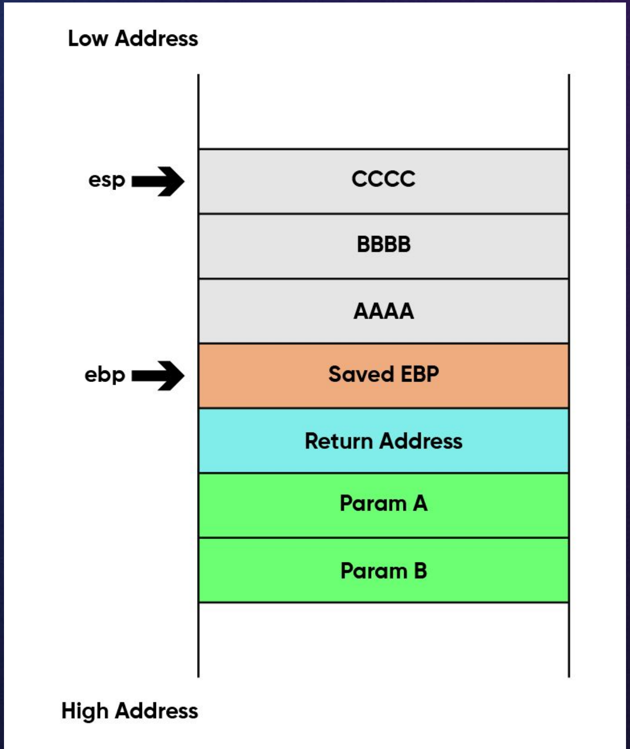
Memory Layout (Linux)

- Typical memory Layout consist of
  - Stack
  - Heap
  - Uninitialized Data Segment
  - Initialized Data Segment
  - Text/Code Segment



STACK

- Block of memory that holds temporary data
  - Operates in LIFO (Last In, First Out) principal
- Grows and shrinks dynamically during program execution
  - Grows towards the lower address (higher -> lower)
- Each function call creates the stack frame, containing parameters, local variables and return address



Low Address

esp → CCCC

BBBB

AAAA

ebp → Saved EBP

Return Address

Param A

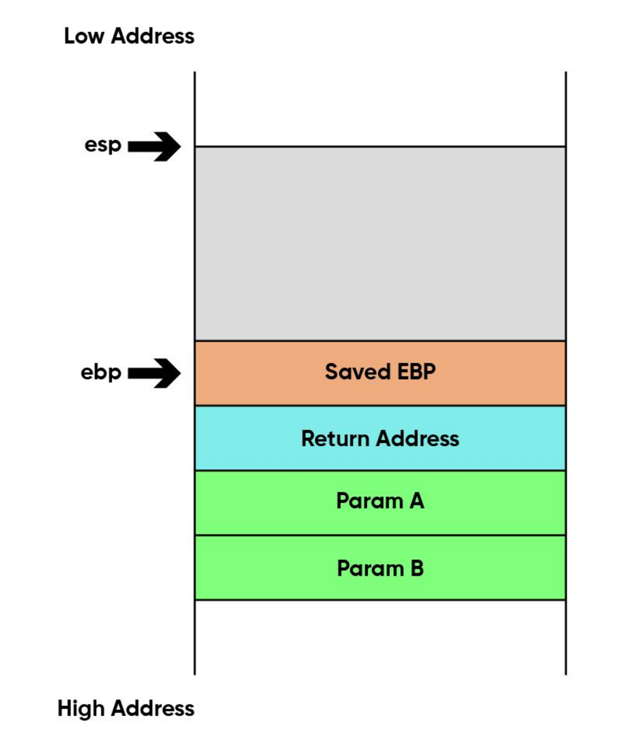
Param B

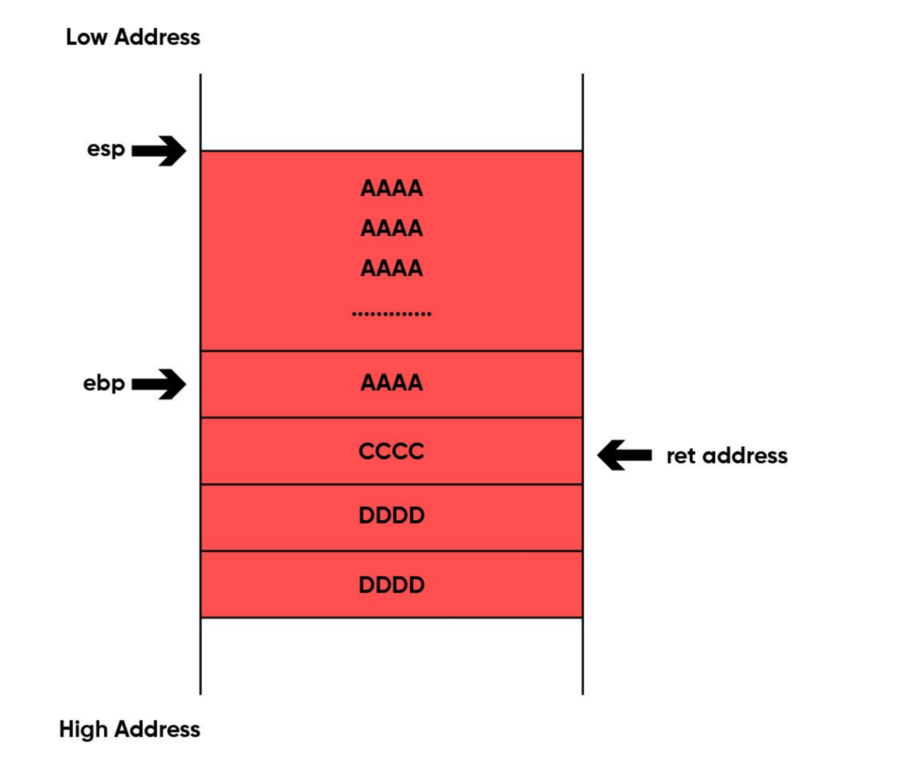
High Address

Stack Based Overflow

- A flaw in software that occurs when more data is written to a buffer on the stack than it can hold,
  - resulting in the overwriting of adjacent memory, including other variables and the return address.

- If exploited correctly and all required conditions are met
  - attacker can overwrite the EIP (Instruction Pointer) register
    - potentially redirecting program execution to malicious code.


Low Address
esp
ebp
Saved EBP
Return Address
Param A
Param B
High Address


Low Address
esp
AAAA
AAAA
AAAA
....
ebp
AAAA
CCCC
ret address
DDDD
DDDD
High Address

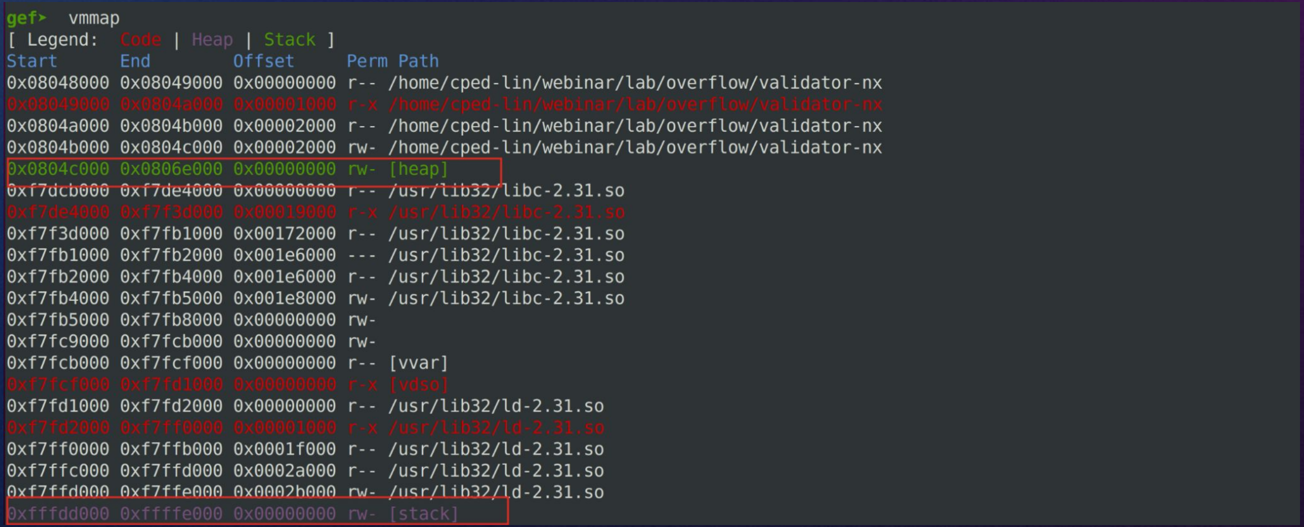
Mitigations

- NX bit
- Canary
- ASLR / PIE
- FORTIFY_SOURCE

NX Bit

- Makes memory region either writable or executable (W^X)
  - cpu won’t execute any code or instructions resides in non-executable region
- Prevents execution in some memory region
  - Stack
  - Heap
- This feature prevents buffer overflow attack to some extent

NX Bit

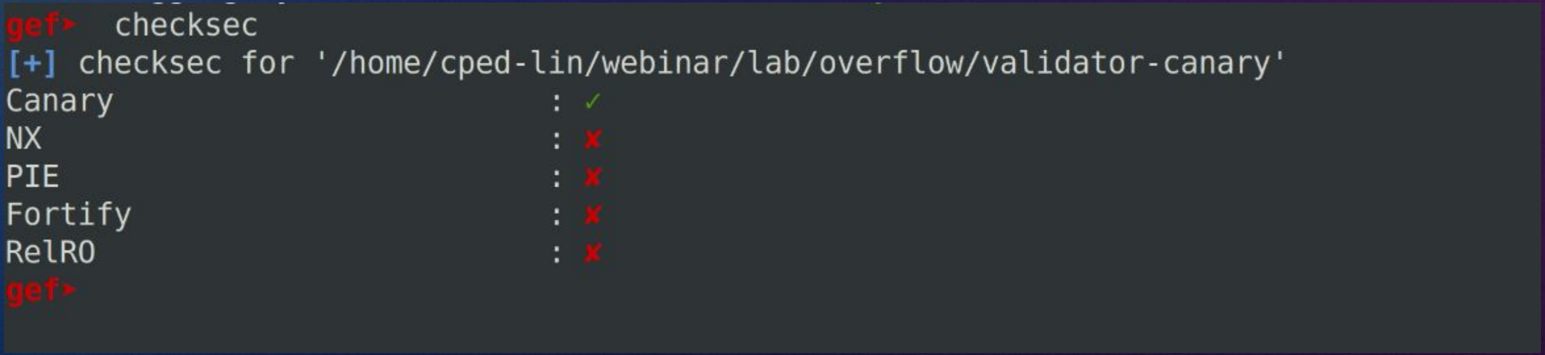


```
gef➤  vmmap
[ Legend:  Code | Heap | Stack ]
Start      End        Offset     Perm Path
0x08048000 0x08049000 0x00000000 r-- /home/cped-lin/webinar/lab/overflow/validator-nx
0x08049000 0x0804a000 0x00001000 r-x /home/cped-lin/webinar/lab/overflow/validator-nx
0x0804a000 0x0804b000 0x00002000 r-- /home/cped-lin/webinar/lab/overflow/validator-nx
0x0804b000 0x0804c000 0x00002000 rw- /home/cped-lin/webinar/lab/overflow/validator-nx
0x0804c000 0x0806e000 0x00000000 rw- [heap]
0xf7dcb000 0xf7de4000 0x00000000 r-- /usr/lib32/libc-2.31.so
0xf7de4000 0xf7f3d000 0x00019000 r-x /usr/lib32/libc-2.31.so
0xf7f3d000 0xf7fb1000 0x00172000 r-- /usr/lib32/libc-2.31.so
0xf7fb1000 0xf7fb2000 0x001e6000 --- /usr/lib32/libc-2.31.so
0xf7fb2000 0xf7fb4000 0x001e6000 r-- /usr/lib32/libc-2.31.so
0xf7fb4000 0xf7fb5000 0x001e8000 rw- /usr/lib32/libc-2.31.so
0xf7fb5000 0xf7fb8000 0x00000000 rw-
0xf7fc9000 0xf7fcb000 0x00000000 rw-
0xf7fcb000 0xf7fcf000 0x00000000 r-- [vvar]
0xf7fcf000 0xf7fd1000 0x00000000 r-x [vdso]
0xf7fd1000 0xf7fd2000 0x00000000 r-- /usr/lib32/ld-2.31.so
0xf7fd2000 0xf7ff0000 0x00001000 r-x /usr/lib32/ld-2.31.so
0xf7ff0000 0xf7ffb000 0x0001f000 r-- /usr/lib32/ld-2.31.so
0xf7ffc000 0xf7ffd000 0x0002a000 r-- /usr/lib32/ld-2.31.so
0xf7ffd000 0xf7ffe000 0x0002b000 rw- /usr/lib32/ld-2.31.so
0xfffdd000 0xffffe000 0x00000000 rw- [stack]
```

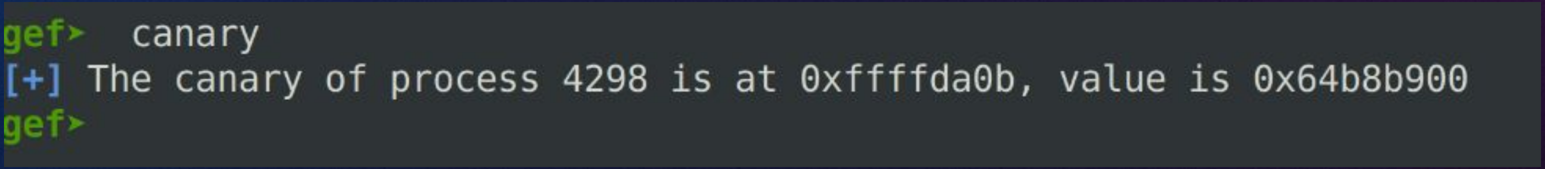
Canary

- Random value that stores before return address in stack
- Random value gets pushed into the stack at function prologue
- Detects the stack smashing
    - While returning from the function
        - Canary value gets checked if overwritten
            - Program terminates and throw message:
                - “Stack smashing detected”

Canary

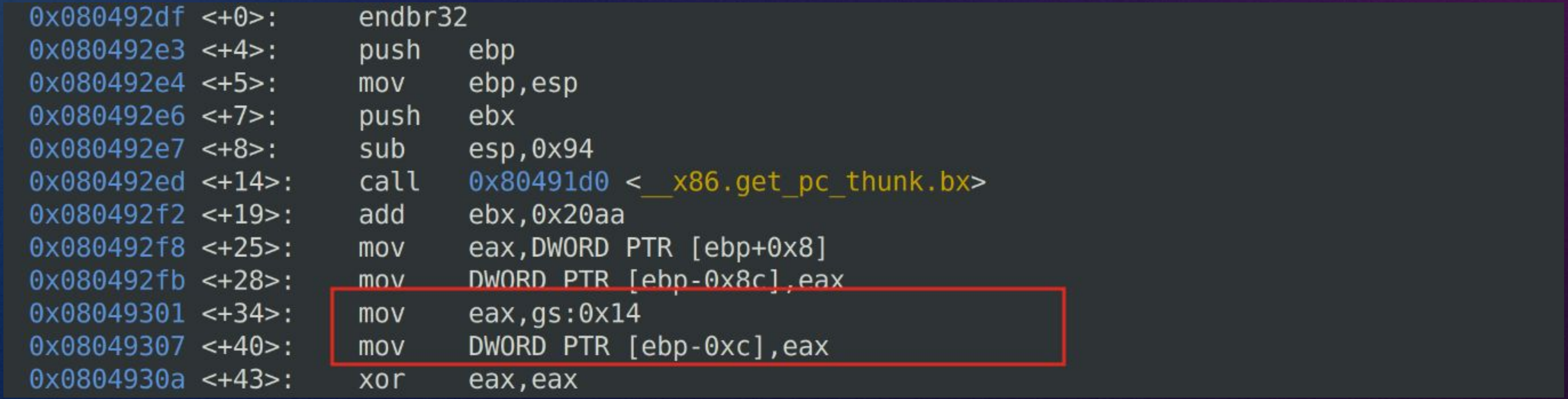


```
gef➤  checksec
[+] checksec for '/home/cped-lin/webinar/lab/overflow/validator-canary'
Canary                        : ✓
NX                            : ✘
PIE                           : ✘
Fortify                       : ✘
RelRO                         : ✘
gef➤
```



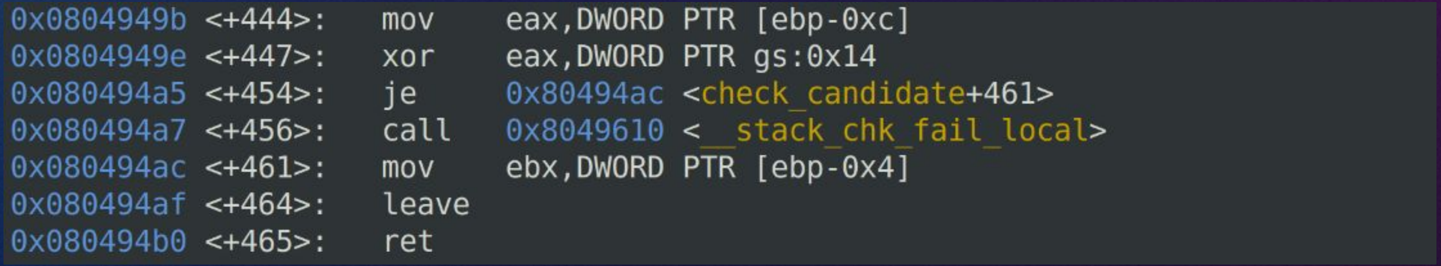
```
gef➤  canary
[+] The canary of process 4298 is at 0xffffda0b, value is 0x64b8b900
gef➤
```

Canary



```
0x080492df <+0>:     endbr32
0x080492e3 <+4>:     push   ebp
0x080492e4 <+5>:     mov    ebp,esp
0x080492e6 <+7>:     push   ebx
0x080492e7 <+8>:     sub    esp,0x94
0x080492ed <+14>:    call   0x80491d0 <__x86.get_pc_thunk.bx>
0x080492f2 <+19>:    add    ebx,0x20aa
0x080492f8 <+25>:    mov    eax,DWORD PTR [ebp+0x8]
0x080492fb <+28>:    mov    DWORD PTR [ebp-0x8c],eax
0x08049301 <+34>:    mov    eax,gs:0x14
0x08049307 <+40>:    mov    DWORD PTR [ebp-0xc],eax
0x0804930a <+43>:    xor    eax,eax
```

Canary



```
0x0804949b <+444>:   mov    eax,DWORD PTR [ebp-0xc]
0x0804949e <+447>:   xor    eax,DWORD PTR gs:0x14
0x080494a5 <+454>:   je     0x80494ac <check_candidate+461>
0x080494a7 <+456>:   call   0x8049610 <__stack_chk_fail_local>
0x080494ac <+461>:   mov    ebx,DWORD PTR [ebp-0x4]
0x080494af <+464>:   leave
0x080494b0 <+465>:   ret
```

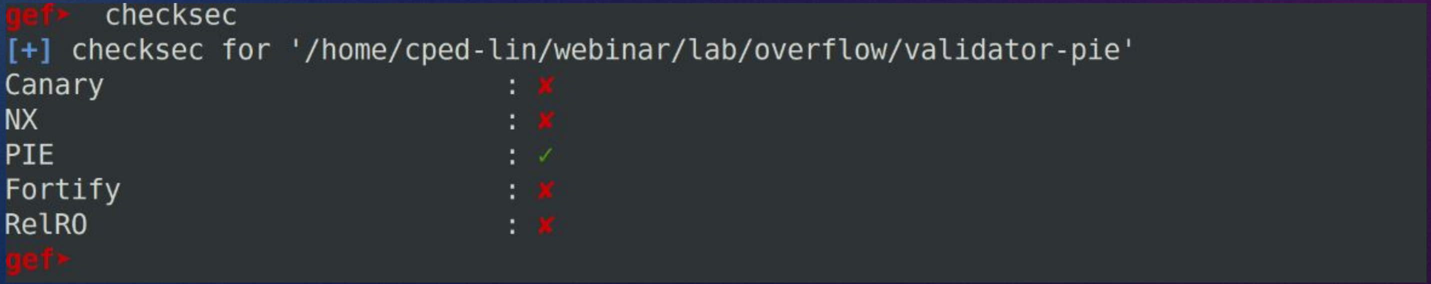
ASLR (Address Space Layout Randomization)

- ASLR randomizes the memory address layout
  - stack, heap, shared libraries

- Makes difficult to find the accurate memory address
  - Prevents from controlling the flow of the execution

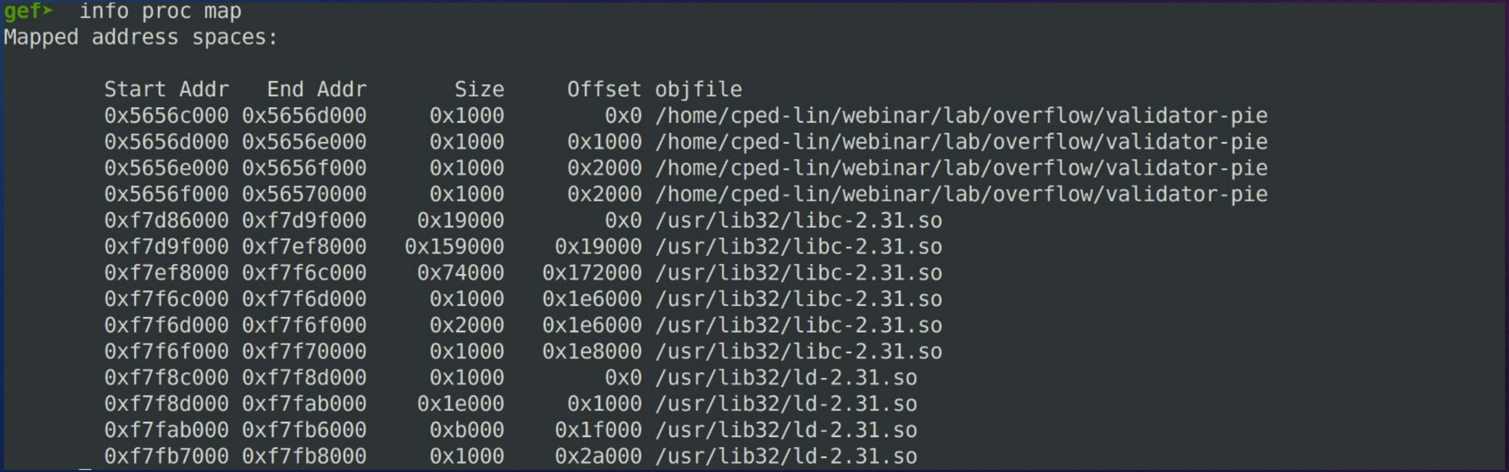
- Position Independent Executable (PIE) randomizes the binary memory base address

ASLR (Address Space Layout Randomization)



```
gef➤  checksec
[+] checksec for '/home/cped-lin/webinar/lab/overflow/validator-pie'
Canary                        : ✘
NX                            : ✘
PIE                           : ✓
Fortify                       : ✘
RelRO                         : ✘
gef➤
```

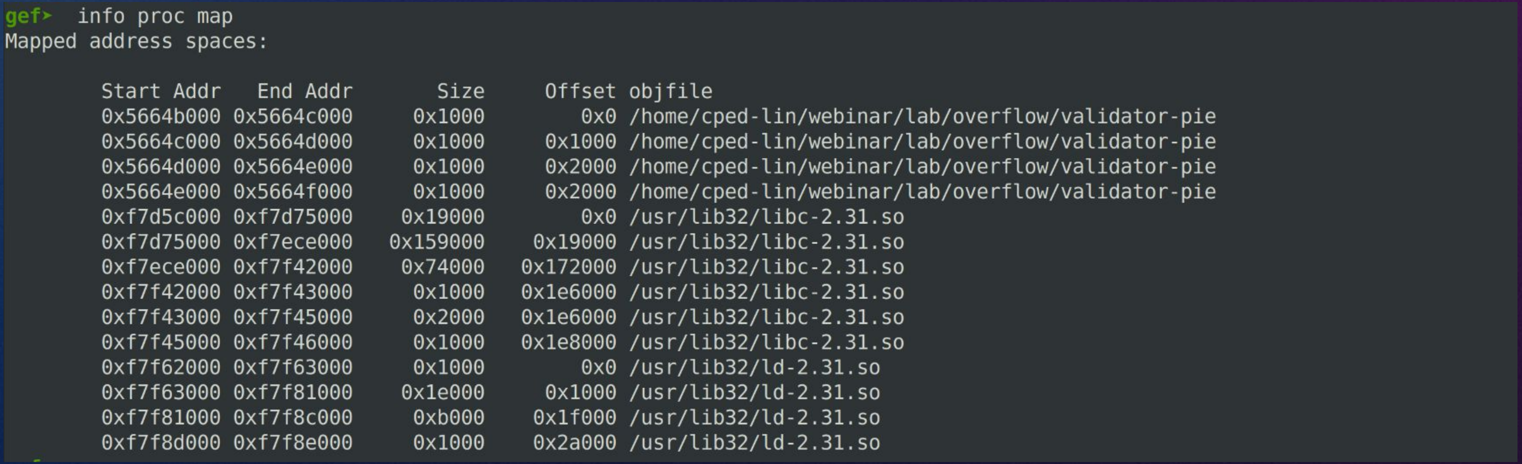
ASLR (Address Space Layout Randomization)



```
gef➤  info proc map
Mapped address spaces:

        Start Addr   End Addr       Size     Offset objfile
        0x5656c000 0x5656d000     0x1000        0x0 /home/cped-lin/webinar/lab/overflow/validator-pie
        0x5656d000 0x5656e000     0x1000     0x1000 /home/cped-lin/webinar/lab/overflow/validator-pie
        0x5656e000 0x5656f000     0x1000     0x2000 /home/cped-lin/webinar/lab/overflow/validator-pie
        0x5656f000 0x56570000     0x1000     0x2000 /home/cped-lin/webinar/lab/overflow/validator-pie
        0xf7d86000 0xf7d9f000    0x19000        0x0 /usr/lib32/libc-2.31.so
        0xf7d9f000 0xf7ef8000   0x159000    0x19000 /usr/lib32/libc-2.31.so
        0xf7ef8000 0xf7f6c000    0x74000   0x172000 /usr/lib32/libc-2.31.so
        0xf7f6c000 0xf7f6d000     0x1000   0x1e6000 /usr/lib32/libc-2.31.so
        0xf7f6d000 0xf7f6f000     0x2000   0x1e6000 /usr/lib32/libc-2.31.so
        0xf7f6f000 0xf7f70000     0x1000   0x1e8000 /usr/lib32/libc-2.31.so
        0xf7f8c000 0xf7f8d000     0x1000        0x0 /usr/lib32/ld-2.31.so
        0xf7f8d000 0xf7fab000    0x1e000     0x1000 /usr/lib32/ld-2.31.so
        0xf7fab000 0xf7fb6000     0xb000    0x1f000 /usr/lib32/ld-2.31.so
        0xf7fb7000 0xf7fb8000     0x1000    0x2a000 /usr/lib32/ld-2.31.so
```

ASLR (Address Space Layout Randomization)



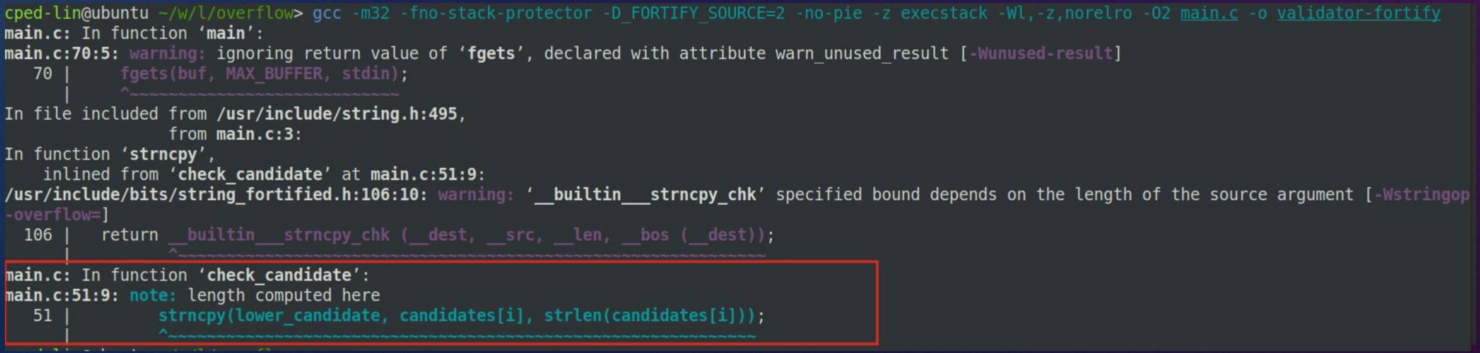
```
gef➤  info proc map
Mapped address spaces:

        Start Addr   End Addr       Size     Offset objfile
        0x5664b000 0x5664c000     0x1000        0x0 /home/cped-lin/webinar/lab/overflow/validator-pie
        0x5664c000 0x5664d000     0x1000     0x1000 /home/cped-lin/webinar/lab/overflow/validator-pie
        0x5664d000 0x5664e000     0x1000     0x2000 /home/cped-lin/webinar/lab/overflow/validator-pie
        0x5664e000 0x5664f000     0x1000     0x2000 /home/cped-lin/webinar/lab/overflow/validator-pie
        0xf7d5c000 0xf7d75000    0x19000        0x0 /usr/lib32/libc-2.31.so
        0xf7d75000 0xf7ece000   0x159000    0x19000 /usr/lib32/libc-2.31.so
        0xf7ece000 0xf7f42000    0x74000   0x172000 /usr/lib32/libc-2.31.so
        0xf7f42000 0xf7f43000     0x1000   0x1e6000 /usr/lib32/libc-2.31.so
        0xf7f43000 0xf7f45000     0x2000   0x1e6000 /usr/lib32/libc-2.31.so
        0xf7f45000 0xf7f46000     0x1000   0x1e8000 /usr/lib32/libc-2.31.so
        0xf7f62000 0xf7f63000     0x1000        0x0 /usr/lib32/ld-2.31.so
        0xf7f63000 0xf7f81000    0x1e000     0x1000 /usr/lib32/ld-2.31.so
        0xf7f81000 0xf7f8c000     0xb000    0x1f000 /usr/lib32/ld-2.31.so
        0xf7f8d000 0xf7f8e000     0x1000    0x2a000 /usr/lib32/ld-2.31.so
```

FORTIFY_SOURCE

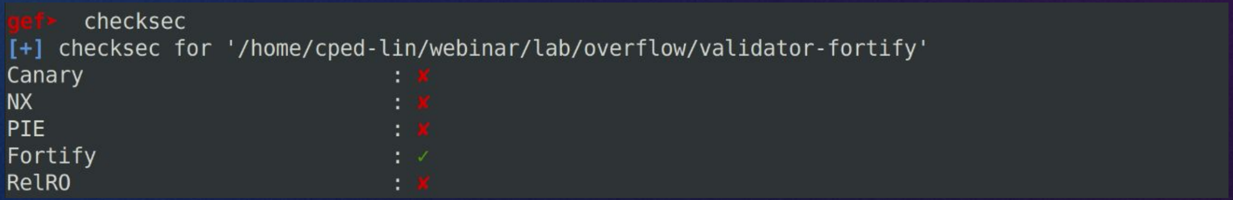
- Compile-time security feature in the GNU C Library (glibc)
- Provides runtime protection for detecting buffer overflow
- Certain buffer manipulation related functions are protected with additional wrapper function:
    - strcpy, gets, memcpy, memmove, etc. [2]
- Wrapper function ends with _chk.

FORTIFY_SOURCE



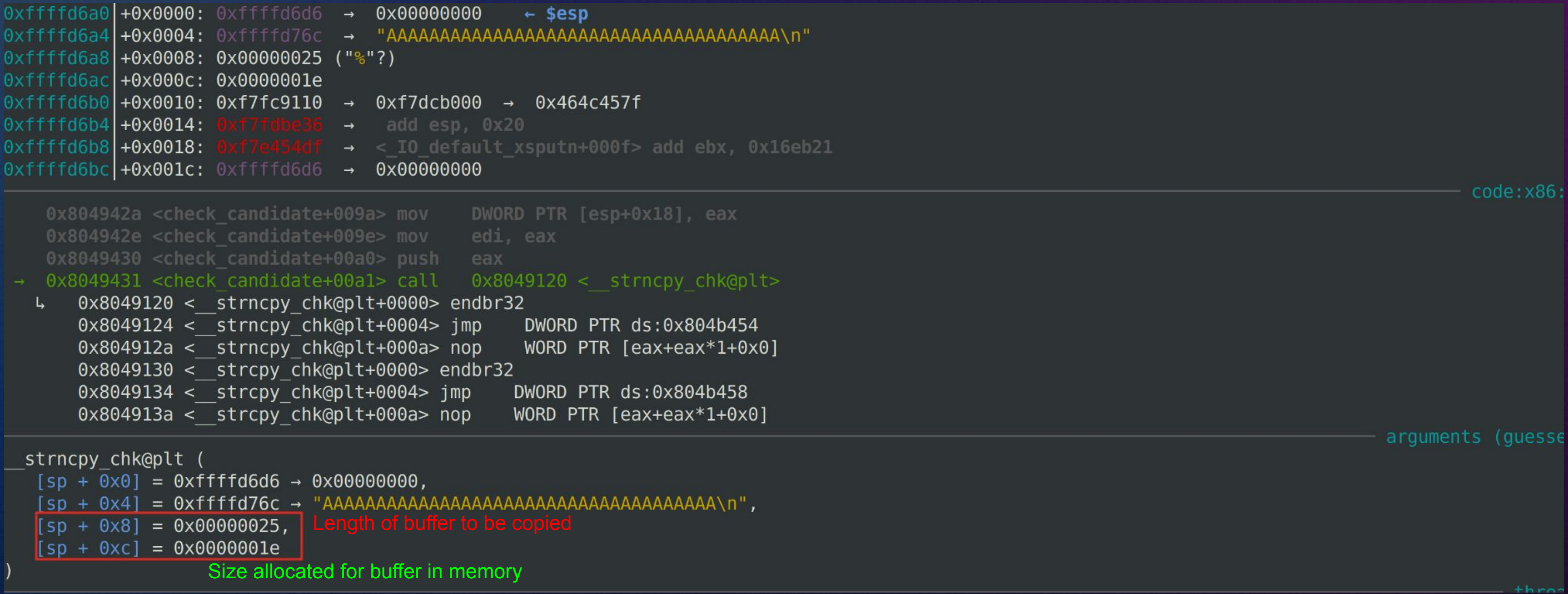
```
cped-lin@ubuntu ~/w/l/overflow> gcc -m32 -fno-stack-protector -D_FORTIFY_SOURCE=2 -no-pie -z execstack -Wl,-z,norelro -O2 main.c -o validator-fortify
main.c: In function ‘main’:
main.c:70:5: warning: ignoring return value of ‘fgets’, declared with attribute warn_unused_result [-Wunused-result]
   70 |     fgets(buf, MAX_BUFFER, stdin);
      |     ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~
In file included from /usr/include/string.h:495,
                 from main.c:3:
In function ‘strncpy’,
    inlined from ‘check_candidate’ at main.c:51:9:
/usr/include/bits/string_fortified.h:106:10: warning: ‘__builtin___strncpy_chk’ specified bound depends on the length of the source argument [-Wstringop
-overflow=]
  106 |   return __builtin___strncpy_chk (__dest, __src, __len, __bos (__dest));
      |          ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
main.c: In function ‘check_candidate’:
main.c:51:9: note: length computed here
   51 |         strncpy(lower_candidate, candidates[i], strlen(candidates[i]));
      |         ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

FORTIFY_SOURCE



```
gef➤  checksec
[+] checksec for '/home/cped-lin/webinar/lab/overflow/validator-fortify'
Canary                        : ✘
NX                            : ✘
PIE                           : ✘
Fortify                       : ✓
RelRO                         : ✘
```

FORTIFY_SOURCE

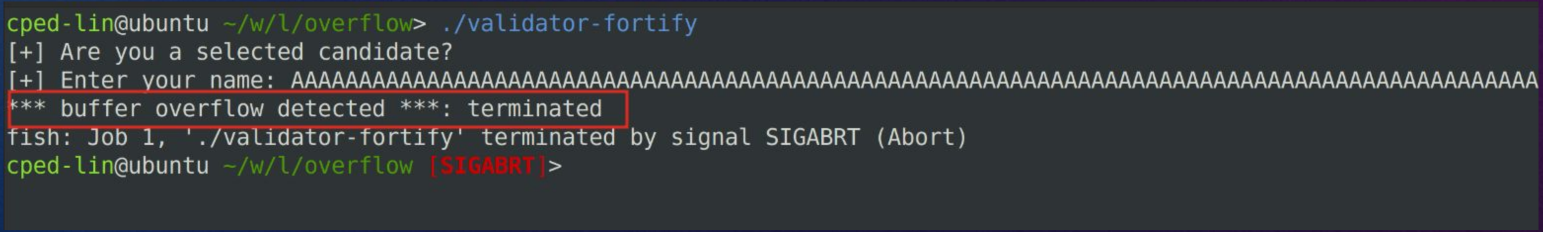


```
0xffffd6a0│+0x0000: 0xffffd6d6  →  0x00000000    ← $esp
0xffffd6a4│+0x0004: 0xffffd76c  →  "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\n"
0xffffd6a8│+0x0008: 0x00000025 ("%"?)
0xffffd6ac│+0x000c: 0x0000001e
0xffffd6b0│+0x0010: 0xf7fc9110  →  0xf7dcb000  →  0x464c457f
0xffffd6b4│+0x0014: 0xf7fdbe36  →   add esp, 0x20
0xffffd6b8│+0x0018: 0xf7e454df  →  <_IO_default_xsputn+000f> add ebx, 0x16eb21
0xffffd6bc│+0x001c: 0xffffd6d6  →  0x00000000
──────────────────────────────────────────────────────────────────────── code:x86:
    0x804942a <check_candidate+009a> mov    DWORD PTR [esp+0x18], eax
    0x804942e <check_candidate+009e> mov    edi, eax
    0x8049430 <check_candidate+00a0> push   eax
 →  0x8049431 <check_candidate+00a1> call   0x8049120 <__strncpy_chk@plt>
   ↳   0x8049120 <__strncpy_chk@plt+0000> endbr32
       0x8049124 <__strncpy_chk@plt+0004> jmp    DWORD PTR ds:0x804b454
       0x804912a <__strncpy_chk@plt+000a> nop    WORD PTR [eax+eax*1+0x0]
       0x8049130 <__strcpy_chk@plt+0000> endbr32
       0x8049134 <__strcpy_chk@plt+0004> jmp    DWORD PTR ds:0x804b458
       0x804913a <__strcpy_chk@plt+000a> nop    WORD PTR [eax+eax*1+0x0]
──────────────────────────────────────────────────────────────── arguments (guesse
__strncpy_chk@plt (
   [sp + 0x0] = 0xffffd6d6 → 0x00000000,
   [sp + 0x4] = 0xffffd76c → "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\n",
   [sp + 0x8] = 0x00000025,
   [sp + 0xc] = 0x0000001e
)
──────────────────────────────────────────────────────────────────────────── threa
```

Length of buffer to be copied

Size allocated for buffer in memory

FORTIFY_SOURCE



```
cped-lin@ubuntu ~/w/l/overflow> ./validator-fortify
[+] Are you a selected candidate?
[+] Enter your name: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
*** buffer overflow detected ***: terminated
fish: Job 1, './validator-fortify' terminated by signal SIGABRT (Abort)
cped-lin@ubuntu ~/w/l/overflow [SIGABRT]>
```

Length of buffer to be copied

Size allocated for buffer in memory

Certified Exploit Development Professional (CEDP)

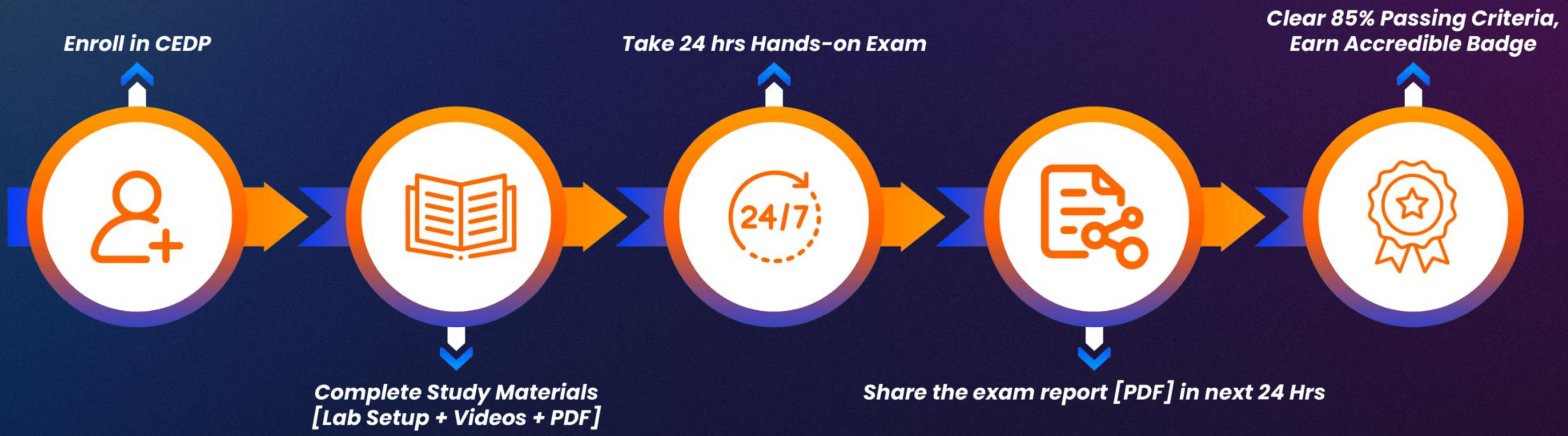
Course Content - Linux

- Vanilla Stack Overflow
- Stack Overflow + NX bypass (ret2libc)
- Stack Overflow + NX bypass (rop chain)
  - Roping mprotect
- Format String BUG
  - NX
  - Canary,
  - ASLR/PIE

Course Content - Win32

- Introduction to Win32 SEH (Structured Exception Handling)
- SEH Overflow + NX bypass
    - Eliminating Bad characters
    - ASLR bypass
        - Non-aslr module
    - ROPing VirtualProtect

Certification Procedure of CEDP Course:



Enroll in CEDP

Complete Study Materials [Lab Setup + Videos + PDF]

Take 24 hrs Hands-on Exam

Share the exam report [PDF] in next 24 Hrs

Clear 85% Passing Criteria, Earn Accredible Badge

References

1. https://cwe.mitre.org/data/definitions/121.html
2. https://www.gnu.org/software/libc/manual/html_node/Source-Fortification.html

Stack-Based-Overflow-&-Mitigations-Webinar.zip:

1. https://drive.google.com/file/d/1mMCHbfBaLGiNVfmUy4yS3lr4v5XwKP-1

CWL
CyberWarFare Labs

Thank You

For Professional Red Team / Blue Team / Purple Team,
Cloud Cyber Range labs / Courses / Trainings, please contact

info@cyberwarfare.live

To know more about our offerings, please visit:

https://cyberwarfare.live