ADDRESS SPACE LAYOUT RANDOMIZATION (ASLR) OVERVIEW

Agenda

- Previous cycles:
  - Discussed stack-based buffer overflow
  - Discussed DEP mitigation
  - Discussed how ROP can thwart DEP
- This cycle:
  - Discuss ASLR mitigation
  - Discuss how to brute force ASLR
  - Tool Demonstration

Stack-based Buffer Overflow

- GOAL:
  - Take control of EIP to execute code of your choosing
- Fixed Memory Locations:
  - Simply overwrite the return address with the address where your code resides

ASLR Mitigations

- ASLR randomizes the memory locations
  - You won't know what address to use to call your shellcode!

STACK on FIRST EXECUTION

0xBFABABA0	shellcode
0xBFABABA4	shellcode
0xBFABABA8	SAVED_EBP
0xBFABABAC	**0xBFABABA0**
0xBFABABB0	… stuff …
0xBFABABB4	… stuff …

STACK on NEXT EXECUTION

0xBFCDCDC0	shellcode
0xBFCDCDC4	shellcode
0xBFCDCDC8	SAVED_EBP
0xBFCDCDCC	**0xBFABABA0**
0xBFCDCDD0	… stuff …
0xBFCDCDD4	… stuff …

ASLR

- Each time the program runs, it is loaded into a different location in memory
  - Thus, the stack address changes every time the program is restarted
- Enable ASLR on Linux:
  - echo 2 > /proc/sys/kernel/randomize_va_space
- Disable ASLR on Linux:
  - echo 0 > /proc/sys/kernel/randomize_va_space

Tool Demonstration

Tool Demonstration - Compile

```
#gcc -g -fno-stack-protector -z
execstack mybigecho.c -o mybigecho

#./mybigecho < payload
mybigecho() $esp = 0xbffff310
Address buffer  =  0xbffff310
   HOWDY TEXAS!!!!
```

Demo – ASLR Disabled

Recall our basic buffer overflow from Cycle02:

```
root@kali:~# echo 0 > /proc/sys/kernel/
randomize_va_space

root@kali:~# ./mybigecho < payload
mybigecho() $esp = 0xbffff310
Address buffer  =  0xbffff310
  HOWDY TEXAS!!!!

root@kali:~# ./mybigecho < payload
mybigecho() $esp = 0xbffff310
Address buffer  =  0xbffff310
  HOWDY TEXAS!!!!
```

Demo – ASLR Enabled

```
root@kali:~# echo 1 > /proc/sys/kernel/
randomize_va_space

root@kali:~# ./mybigecho < payload
mybigecho() $esp = 0xbfd99580
Address buffer  =  0xbfd99580
Segmentation fault

root@kali:~# ./mybigecho < payload
mybigecho() $esp = 0xbfe9fa60
Address buffer  =  0xbfe9fa60
Segmentation fault

root@kali:~# ./mybigecho < payload
mybigecho() $esp = 0xbfcc4520
Address buffer  =  0xbfcc4520
Segmentation fault
```

Brute Force ASLR

- On 32-bit architectures with ASLR, there are only 20 bits of randomness in the stack address because in practice certain bits remain the same
- From our previous example:
  - `0xBFD99580`
  - `0xBFE9FA60`
  - `0xBFCC4520`
- Only the middle 5 bytes are changing:
  - `0xBFxxxxx0`

Tool Demonstration

Keep the same address guess (0xBFFFF310), but try it repeatedly, with a NOP-sled to help:

```
for i in {1..250000}; do
  echo "Try: $i";
  out=$( ./mybigecho < payload 2>&1 );
  if [[ $out == *TEXAS* ]]; then
    echo "$out";
    break;
  fi;
done
```

Summary

3 Main Ideas

- ASLR mitigation randomizes memory addresses, making it more difficult for the attacker to know where the code he wants to execute is located
- ASLR can be brute forced by attacking repetitively until the guessed address lands in the NOP-sled, assuming the program can handle multiple tries
- 64 bit operating systems make brute forcing more difficult

Future Work

- Discuss memory leakage attacks to gain knowledge about the memory locations without brute forcing
- Discuss how Stack Canaries affect the scenario
- Discuss additional mitigations such as Enhanced Mitigation Experience Toolkit (EMET) for Windows and grsecurity for Linux

References

- Sims, S. (2010). Brute forcing ASLR on Linux, part 1 [video]. *Youtube.* Retrieved from https://youtu.be/DcaVyy4yu88
- Sims, S. (2010). Brute forcing ASLR on Linux, part 2 [video]. *Youtube.* Retrieved from https://youtu.be/LRjsv5zAHjQ

Post Questions and Comments to the Discussion Board