# Defenses in depth

## Summary

In this exercise, we will explore pertinent security features in Ubuntu Linux.

## Prerequisites

* Setup an Ubuntu VM as outlined in the VM setup instructions on Blackboard
* Install hardening-check perl script (part of devscripts package) using

$ sudo apt-get install devscripts

## Background

* Address space layout randomization
  + Programs and all shared objects must be compiled with "-fPIC"
  + ASLR is implemented by the kernel and the ELF loader by randomizing the location of memory allocations (stack, heap, shared libraries, etc)
  + Makes memory addresses harder to predict when an attacker is attempting a memory-corruption exploit
  + Controlled system-wide by the value of /proc/sys/kernel/randomize\_va\_space
    - 0 = Disabled
    - 1 = On
    - 2 = On, with brk ASLR (Default setting)
  + Types of ASLR
    - STACK ASLR - Each execution of a program results in a different stack memory space layout
      * Makes it harder to locate in memory where to attack or deliver an executable attack payload
    - LIBS/MMAP ASLR - Each execution of a program results in a different mmap memory space layout (which causes the dynamically loaded libraries to get loaded into different locations each time)
      * Makes it harder to locate in memory where to jump to for "return to libc" to similar attacks
    - EXEC ASLR - Each execution of a program (that has been linked with "-fPIE -pie") will get loaded into a different memory location
      * Makes it harder to locate in memory where to attack or jump to when performing memory-corruption-based attacks (e.g., "return-to-text" attacks)
    - brk ASLR - adjusts the memory locations relative between the exec memory area and the brk memory area (for small mallocs)
    - VDSO ASLR - Each execution of a program results in a random virtual dynamic shared object location
      * Protects against jump-into-syscall attacks
* Stack protection (gcc)
  + "-fstack-protector" option provides a randomized stack canary that protects against stack overflows
    - Reduces the chances of arbitrary code execution via controlling return address destinations. Enabled at compile-time
    - "-fstack-protector" option only modifies functions that call alloca, and functions with buffers larger than 8 bytes
    - "-fstack-protector-all" option provides randomized stack canary checks that protects all functions
  + Routines used for stack checking are actually part of glibc, but gcc is patched to enable linking against those routines by default
* Non-executable data sections in memory
  + Most modern CPUs protect against executing non-executable memory regions (heap, stack, etc)
    - Known either as Non-eXecute (NX) or eXecute-Disable (XD)
  + Programs linked with "-z noexecstack" will enable this security feature (we will explore this in the ROP class exercise)
* Glibc
  + Format string security checks
    - "-Wformat=1 -Wformat-security=1" enables checks (set to 0 disables)
  + Fortify source protections
    - -D\_FORTIFY\_SOURCE=1, with compiler optimization level 1 (gcc -O1) and above, checks that shouldn't change the behavior of conforming programs are performed
    - -D\_FORTIFY\_SOURCE=2, more checking is added, but conforming programs might fail these checks. Some of the checks can be performed at compile time, and result in compiler warnings; other checks take place at run time, and result in a run-time error if the check fails
* GOT protections (RELRO + BIND\_NOW)
  + RELRO - Hardens ELF programs against loader memory area overwrites by having the loader mark any areas of the relocation table as read-only for any symbols resolved at load-time ("read-only relocations")
  + Protects against GOT-overwrite-style memory corruption attacks
  + Linker option is "-z relro"
  + BIND\_NOW - Marks ELF programs to resolve all dynamic symbols at start-up (instead of on-demand, also known as "immediate binding") so that GOT can be made entirely read-only
  + Linker option is "-z now"

## Details

### ASLR

* Download the project into a local sandbox

$ git clone <https://gitlab.com/underpantsgnomes/softwaresecurity/aslrdemo.git>

* Turn off ASLR and build with option "-fPIE" to gcc and "-no-pie" to ld

$ echo 0 | sudo tee /proc/sys/kernel/randomize\_va\_space

* Inspect aslrdemo with hardening-check

$ hardening-check aslrdemo

* Run aslrdemo several times
* Turn on ASLR, mode 1

$ echo 1 | sudo tee /proc/sys/kernel/randomize\_va\_space

* Run aslrdemo several times
* Turn on ASLR, mode 2

$ echo 2 | sudo tee /proc/sys/kernel/randomize\_va\_space

* Run aslrdemo several times
* Build with linker option "-fPIE" to gcc and "-pie" to ld
* Inspect aslrdemo with hardening-check

$ hardening-check aslrdemo

* Run aslrdemo several times

### Stack protections

* Compile stack smashing exploit with "-fno-stack-protector" and specify SLEDGEHAMMER mode
* Run and inspect prologue and epilogue of vulnerable function
* Compile the same with "-fstack-protector" and specify SLEDGEHAMMER mode
* Run and inspect prologue and epilogue of vulnerable function

### Fortify source protections

* Instructor will provide details for this to do in HW4

### GOT protections (RELRO + BIND\_NOW)

* Instructor will provide details for a GOT corruption program

## References

* <https://wiki.ubuntu.com/Security/Features>
* <https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html>
* <https://gcc.gnu.org/onlinedocs/gcc/Warning-Options.html>
* <http://man7.org/linux/man-pages/man7/feature_test_macros.7.html>
* <https://sourceware.org/glibc/wiki/PointerEncryption>