Operating Systems Security

Linux Fundamentals & Practice

(Manual, Commands, Alignment, …)

Seong-je Cho

Fall 2018

Computer Security & Operating Systems Lab, DKU

Sources / References

- Linux User & Programmer's Manual – Man_pages
- Linux and UNIX overview, www.cs.sjsu.edu/~stamp/CS286/**ppt**/3_Linux_UNIX.**ppt**
- Several Blogs for Linux practice and fundamentals
- Alignment in C
- gcc Compiler Options
- Some information from Google Searches

Please do not duplicate and distribute

Contents

Linux Fundamentals & Practice that focuses on Buffer Overflows

- UNIX/Linux Overview
- Overview of Linux User & Programmer's Manual

- Linux commands for Hands-on Experience of Buffer Overflows
  - Enabling/Disabling Stack Protection
  - Enabling/Disabling Address Space Layout Randomization (ASLR)

- Alignment, Memory allocation, …

- Compiler Options

Linux and UNIX

- Linux and UNIX OSs are…
  - Often targets for attacks
  - Often used for launching attacks
- So we need to understand basics

- UNIX
  - Strange because so many UNIX OSs
  - Popular variants include
    - Solaris by Sun / HP-UX by HP / IRIX by sgi / AIX by IBM
    - MacOS by Apple
    - FreeBSD, free open source
    - OpenBSD, “the #1 most secure” OS
  - Differences between UNIX variants
    - File system organization
    - System calls, commands, command options, etc.

Linux and UNIX

- **Linux**
  - Developed by Linus Torvalds
  - Technically, not a variant of UNIX
  - Created without using any of the underlying UNIX code
  - A “UNIX-like environment”
  - Strictly speaking, “Linux” is just the kernel
  - Many Linux “distros”: Debian, Gentoo, Mandrake, Red Hat, Slackware, SuSE, etc.

- **Here, we focus on generic UNIX/Linux concepts**
  - Things that apply to most UNIX/Linux
  - Linux/UNIX is Multi-user system

Linux User & Programmer's Manual - Manpages

Section	Description
1	General commands
2	System calls
3	Library functions, covering in particular the C standard library
4	Special files (usually devices, those found in /dev) and drivers
5	File formats and conventions
6	Games and screensavers
7	Miscellanea
8	System administration commands and daemons

- Section l: math library functions
- Quiz
  - passwd (?),
  - system(), execve(), execlp(),

Keywords related to Each Section in Linux Manual

1. passwd, ls -l, ps, ipcs, mknod, fdisk, sh, bash, strings, size, file, gdb, chmod, chown, find, ln, su,
2. execve(), fork(), read(), write(), ioctl(),

3. system(), gets(), fread(), printf(), sprint(), scanf(), execl(), execlp(), strcpy(), strncpy(), strlcpy(), strcat(), strlen(), memcpy(),
4. /dev/{tty, hd*, mem, kmem, ram0, initrd}

5. /etc/{passwd, shadow, hosts, fstab, netmasks, profile}, /etc/networks/interfaces, /proc/[pid]/{attr, exe, limits, maps, stat, …}, crontab,
6. …
7. sched, capabilities, credentials, cpuset, cgroups, netlink, socket,
8. mount, filecap, ld.so, netcap, pscap,

Linux Commands

- **execstack (8)**
  - tool to set, clear, or query executable stack flag of ELF binaries and shared libraries
- **sysctl (8) : configure kernel parameters at runtime**
  - `-w` : use this option when you want to change a sysctl setting
  - The parameters available are those listed under `/proc/sys/`
- **sudo (su "do") (8)**
  - allows a system administrator to delegate authority to give certain users (or groups of users) the ability to run some (or all) commands as root or another user while providing an audit trail of the commands and their arguments
- **ldd (1) – list dynamic dependencies of executable files or shared objects**
- **tee (1) – read from standard input and write to standard output and files**
- **readelf (1) – display information about ELF files**
- **objdump (1) – display information from object files.**
- **nm (1) – list symbols from object files**

Stack Protection against Buffer Overflow

- Disable stack protection on Ubuntu for buffer overflow

```
gcc -fstack-protector -masm=intel -S test.c
gcc -fno-stack-protector -masm=intel -S vulpro.c
```

  - You can compile without stack canaries (`-fno-stack-protector`) and with making code executable on the stack (`-z execstack`)

```
gcc -fno-stack-protector -z execstack -o <my_pg> my_pg.c
```

  - Making code no-executable on the stack with linker options (`-z noexecstack`)

- Enable an executable stack (without needing a recompile)

```
execstack -s /path/to/myprog
```

  - `-s --set-execstack`
    - Mark binary or shared library as requiring executable stack.
  - `-c --clear-execstack`
    - Mark binary or shared library as not requiring executable stack.

Address Space Layout Randomization (ASLR)

- View ASLR settings

```
$ cat /proc/sys/kernel/randomize_va_space
2
$ sysctl -a --pattern randomize
kernel.randomize_va_space = 2
```

- View Address Space

```
$ ldd /bin/bash
 linux-vdso.so.1 =>  (0x00007fff6f572000)
 libtinfo.so.5 => /lib/x86_64-linux-gnu/libtinfo.so.5 (0x00007fc1ecf16000
 libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007fc1ecd12000)
 libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fc1ec948000)
 /lib64/ld-linux-x86-64.so.2 (0x00007fc1ed13f000)
$ ldd /bin/bash
 linux-vdso.so.1 =>  (0x00007ffca6113000)
 libtinfo.so.5 => /lib/x86_64-linux-gnu/libtinfo.so.5 (0x00007f8e8dc3e000
 libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f8e8da3a000)
 libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f8e8d670000)
 /lib64/ld-linux-x86-64.so.2 (0x00007f8e8de67000)
```

```
$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

- Disable ASLR

```
$ ldd /bin/bash
 linux-vdso.so.1 =>  (0x00007ffff7ffa000)
 libtinfo.so.5 => /lib/x86_64-linux-gnu/libtinfo.so.5 (0x00007ffff7bae000)
 libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007ffff79aa000)
 libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ffff75e0000)
 /lib64/ld-linux-x86-64.so.2 (0x00007ffff7dd7000)
$ ldd /bin/bash
 linux-vdso.so.1 =>  (0x00007ffff7ffa000)
 libtinfo.so.5 => /lib/x86_64-linux-gnu/libtinfo.so.5 (0x00007ffff7bae000)
 libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007ffff79aa000)
 libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ffff75e0000)
 /lib64/ld-linux-x86-64.so.2 (0x00007ffff7dd7000)
```

Disable/Enable ASLR

- Configure ASLR using **/proc/sys/kernel/randomize_va_space**
  - 0 : No randomization. Everything is static.
  - 1 : Conservative randomization.
    - Shared libraries, stack, mmap(), VDSO and heap are randomized.
    - VDSO: virtual dynamically linked shared objects
  - 2 : Full randomization.
    - In addition to elements listed in the previous point, memory managed through brk() is also randomized.
- Disable ASLR

```
echo 0 | sudo tee /proc/sys/kernel/randomize_va_space    (or)
echo 0 > /proc/sys/kernel/randomize_va_space
```

- Enable ASLR

```
echo 2 | sudo tee /proc/sys/kernel/randomize_va_space
```

- To temporally disable it, use `sudo sysctl kernel.randomize_va_space=0`
- To permanently disable it, add a file **/etc/sysctl.d/01-disable-aslr.conf** containing: **kernel.randomize_va_space = 0**

Alignment in C

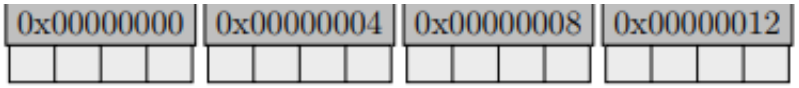


Figure 1: Four word-sized memory cells in a 32-bit computer

For instance, saving a 4 byte **int** in our memory will result in the integer being properly aligned without doing any special work because an int on this architecture is exactly 4 byte which will fit perfectly into the first slot.

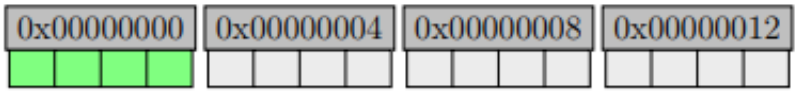


Figure 2: Our memory with an int in it

If we instead decided to put a **char** , a **short** and an **int** into our memory we would get a problem if we did so naïvely without worrying for alignment.

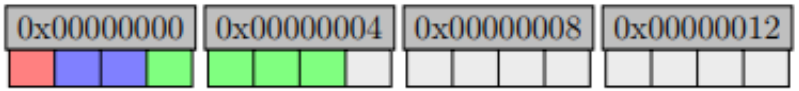


Figure 3: Misaligned memory

This would need two memory accesses and some bitshifting to fetch the **int**. Effectively that means it will take at least two times as long as it would if the data were properly aligned. For this reason, computer scientists came up with the idea of adding padding to data in memory so it would be properly aligned. In our example, adding padding after the first byte, the char, would ensure that the last part of the data would be properly aligned in memory:

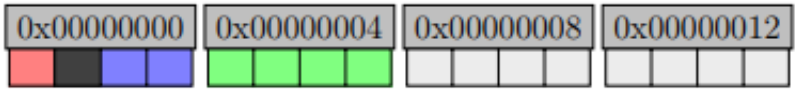


Figure 4: Properly aligned memory using padding

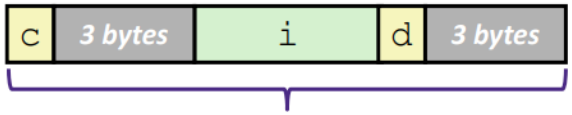
The figure 4 is considered naturally aligned. Compilers will automatically add correct padding for the target platform unless this feature is deliberately switched off.

Alignment

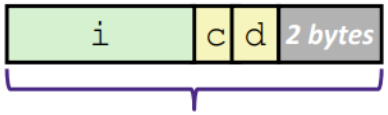
- Memory allocation

```
struct S4 {
  char c;
  int i;
  char d;
} *p;
```

```
struct S5 {
  int i;
  char c;
  char d;
} *p;
```

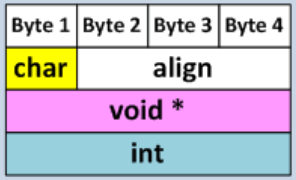
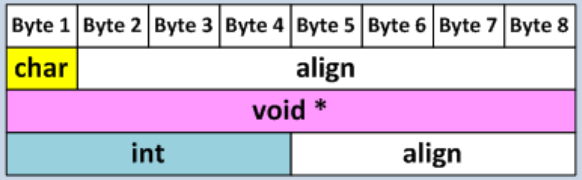


12 bytes

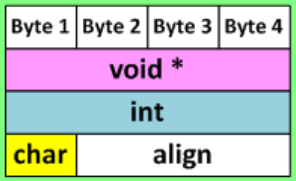
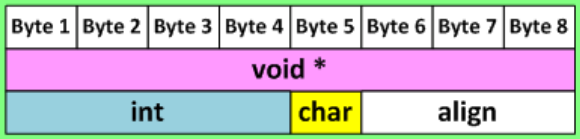


8 bytes

	Windows 32-bit (ILP32LL)	Windows 64-bit (LLP64)
sizeof(MyStruct1)	12	24

struct MyStruct1
{
char m_c;
void *m_p;
int m_i;
};
		

sizeof(MyStruct2)	12	16

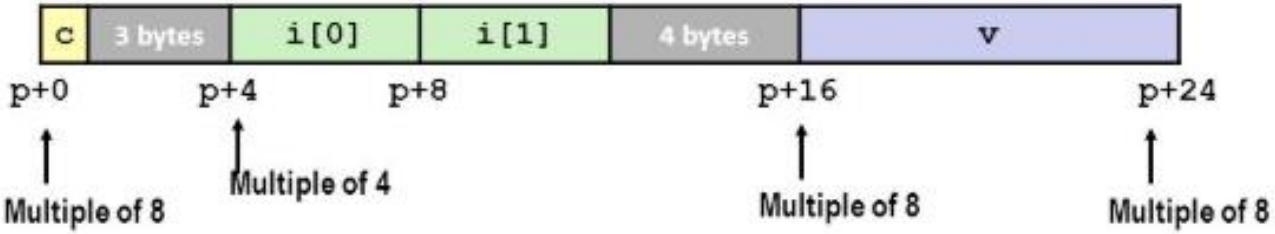
struct MyStruct2
{
void *m_p;
int m_i;
char m_c;
};
		

Different alignment conventions

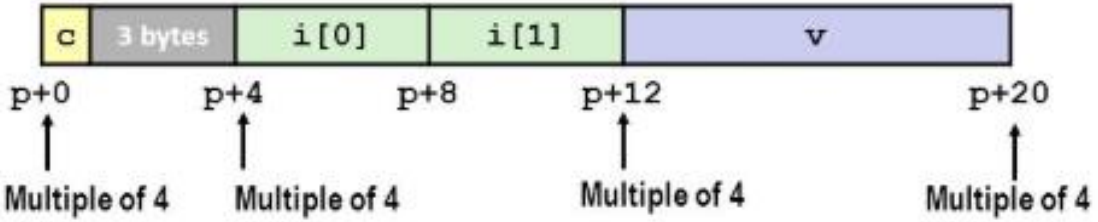
- **Windows vs. Linux**

- **x86-64 or IA32 Windows:**
  - K = 8, due to `double` element

```
struct S1 {
  char c;
  int i[2];
  double v;
} *p;
```



- **IA32 Linux**
  - K = 4; double treated like a 4-byte data type



How to compile 32-bit program on 64-bit gcc

- Confirm which bit-version of gcc is currently installed in our system
  - gcc –v
  - `Target: x86_64-linux-gnu` (64-bit gcc)

```
Command: gcc -v
Output
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/5/lto-wrapper
Target: x86_64-linux-gnu
.....................
.....................
```

- Install `gcc-multlib`
  - For C language: `sudo apt-get install gcc-multilib`
  - For C++ language: `sudo apt-get install g++-multilib`

- To compile with 32-bit gcc, just add a flag –m32
  - `gcc -m32 geek.c -o out`
  - `gcc -m64 geek.c -o out` ← default 64-bit compilation

Appendix

ASLR

- `/proc/sys/kernel/randomize_va_space` interface controls ASLR system-wide.
  - If you don't want a system-wide change, use ADDR_NO_RANDOMIZE personality to temporarily disable ASLR. Controlling this personality flag can be done with `setarch` and its `-R` option, prepending a command.
  - I find it really convenient to open a completely new shell using:

```
setarch `uname -m` -R /bin/bash
```

  - This will open a new Bash shell for you with ASLR disabled, including all child processes (programs run from this shell).
  - Just **`exit`** the shell once you're done.