Lecture # 8.2
Exploiting Buffer Overflow Vulnerability
Part-II

Course: **Advance Operating System**

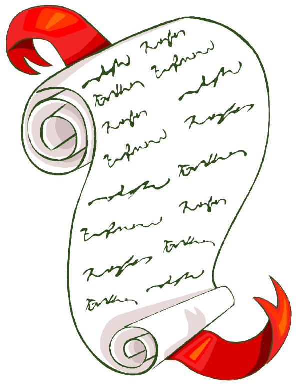
Instructor: Arif Butt

Punjab University College of Information Technology (PUCIT)
University of the Punjab

Source Code files available at: **`https://bitbucket.org/arifpucit/spvl-repo/src`**
Lecture Slides available at: **`http://arifbutt.me`**

Today's Agenda

- What is Shellcode?
- From where to get shellcodes?
  - Writing your own using Assembly Language
  - Downloading from some Internet source
  - Creating using PEDA and pwn tools
  - Creating using Metasploit Framework (msfvenom)
- How to use shellcodes?
  - Stand alone
  - Injecting to a vulnerable process address space

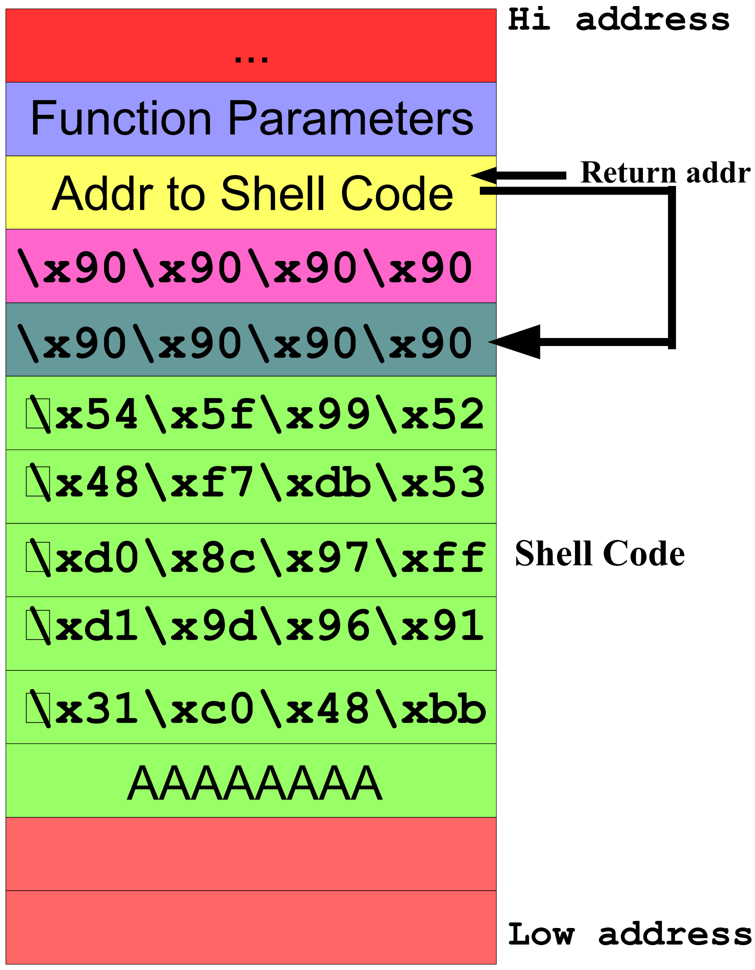


What is a Shellcode?

A *shellcode* is a machine dependent code that can be executed by the CPU directly w/o the need of any compiling, assembling or linking. The typical goal of a *shellcode* is to create a shell preferably with root privileges, that is why it is named as *shellcode*. The *shellcode* is stored in a process address space at some convenient place, which can be:

- Code Section
- Process Stack
  - As part of input buffer
  - In some environment variable
- Process Heap

To execute a *shellcode,* all you need to do is simply transfer control of execution to that address



Writing your own Shellcode

Writing your own Shellcodes

- Can be written in C, but better to write in assembly
- Must know the underlying system calls (`execve()`, `open()`, `read()`, `write()`, `dup2()`, `setresuid()` and others). For example, place the string `/bin/bash` in memory and pass it as argument to `syscall()` system call
- Must have a clear understanding of architecture's function calling conventions
- **Steps to follow:**
  - Write assembly code and create object file using following command

    ```
    $ nasm -f elf64 <filename>.nasm
    ```
  - Check out the opcodes in the object file

    ```
    $ objdump -M intel -D <filename>.o
    ```
  - If opcode of an instruction contain NULL bytes `'\x00'`, then that assembly instruction must be rewritten. For example
  - `MOV rax, 0     (b8 00 00 00 00)`
  - `XOR rax, rax      (48 31 c0)`
  - Finally extract the opcode from the object file, which is you *shellcode*

Using Shellcode in a Standalone C Program

- Once you have written your *shellcode*. You can use it. Mostly this *shellcode* is made a part of the string which is given as input to a vulnerable program for a buffer overflow attack. But today we will run our *shellcodes* as a stand alone C program
- The general format of a C program using a *shellcode* is shown below:

```
#include <stdio.h>
#include <string.h>

char *code = "shellcode";

int main(){
   printf("len:%d bytes\n", strlen(code));
   int (*foo)() = (int(*)())code;
   foo();
   return 0;
}
```

Writing your own Shellcodes Proof of Concept

`writing_shellcodes/ex1/`

`writing_shellcodes/ex2/`

Downloading Shellcodes From Internet Archives

Shellcode Archives on Internet

- A good hacker always write his/her own *shellcodes*, and the task is not much tricky for assembly guys. The obvious drawback is that you can write *shellcode* for the architecture that you are working on. What if you want the *shellcode* for other architectures like arm64, powerpc or android. Moreover, sometimes it becomes a bit tricky if you are not good at assembly language programming. So being newbies the simplest option is downloading *shellcode* for your specific hardware and operating system from some Internet archives like:

```
http://www.shell-storm.org/shellcode

http://www.exploit-db.com

http://www.projectshellcode.com
```

Using Shell Codes Downloaded from Internet
Proof of Concept

downloaded_shellcodes/myshell.c
downloaded_shellcodes/shell_bind_tcp.c
downloaded_shellcodes/shell_reverse_tcp.c

Generating Shellcodes using PWN Tools

PEDA and PWN Tools

- We have seen the installation and usage of PEDA in the previous session which is just a wrapper around gdb. We have used the feature of its customized view (register, code, and stack) in the previous session. Today, we are going to use `pwntools` which is a CTF framework and exploit development library written in python. It is designed for rapid prototyping and development and intended to make exploit writing as simple as possible
- We will focus on one of its sub-module called `shellcraft`, which allows us to write assembly code similar to what we can do with NASM, but using python. So you don't have to know much about assembly to make it work
- You can install `pwntools` using the following command

```
$ sudo apt-get install python-dev python-pip
$ sudo pip install pwntools
```

Generating Shellcodes using PEDA & PWN Tools

Proof of Concept

Generating Shellcodes using MSF's `msfvenom`

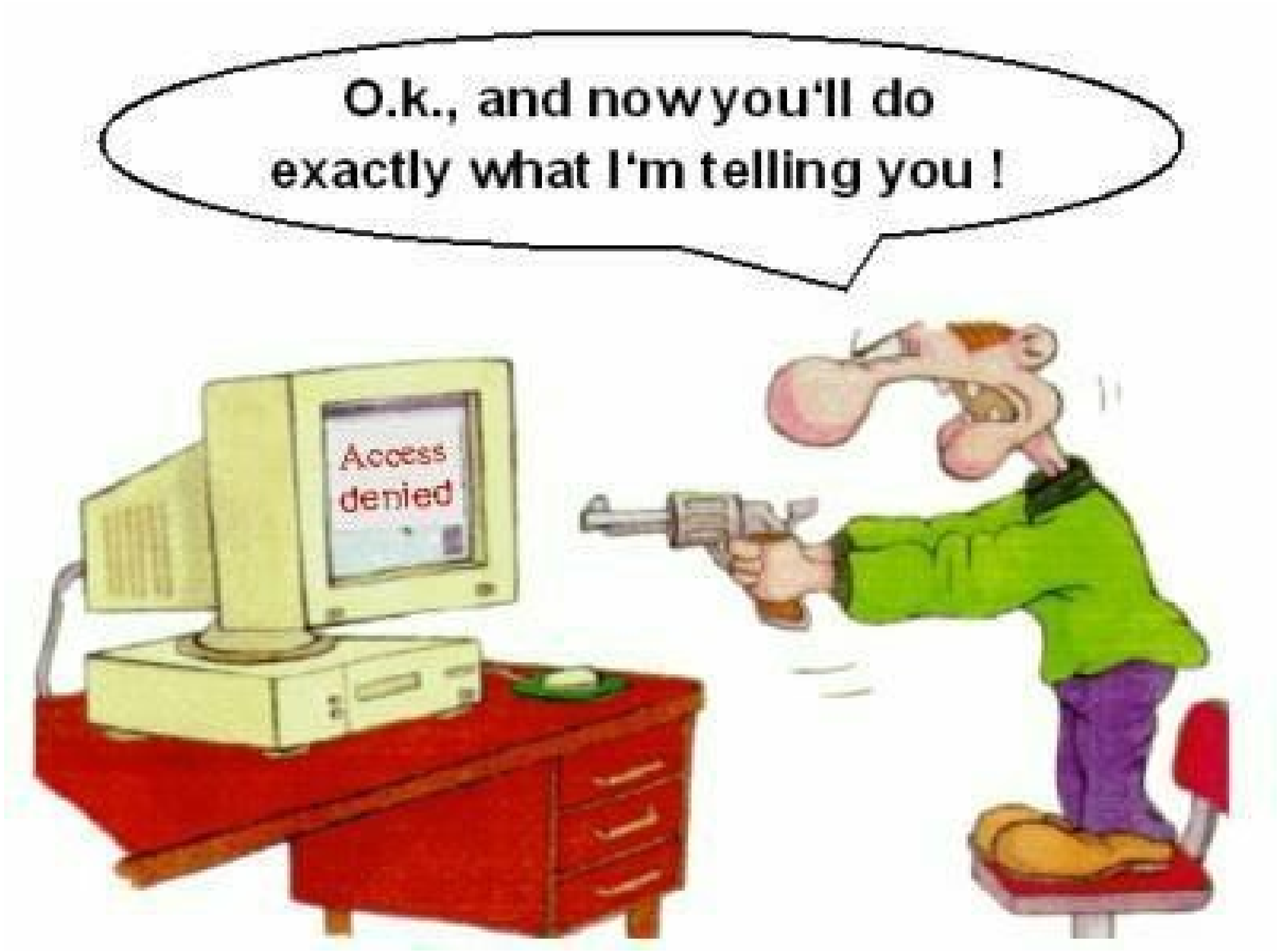
MSF and msfvenom

- Metasploit Framework is an open source pentesting framework used for:
  - Vulnerability research
  - Writing, testing and using exploit code
  - Shellcode development
- This comes pre-installed with Kali Linux. There are many interfaces to it like, `msfconsole`, `msfcli`, `msfgui`, `msfweb`, and `armitage`
- Today our main concern is about `shellcodes`, so we are going to use its submodule `msfvenom`, to generate predefined `shellcodes` suitable for many platforms and architectures. Some self contained payloads that do a specific task are available in `/usr/share/metasploit-framework/modules/singles/` directory
- msfvenom is a combination of Msfpayload and Msfencode, putting both of these tools into a single Framework instance
- Additional benefits of using `msfvenom` are like avoiding bad characters like null bytes and generating the `shellcode` in various file formats to be used in C, Python, or C# programs
- Do dig out another module of MSF, i.e., `meterpreter`, which is a post exploitation tool and is used for in-memory dll/so-injection in the memory space of the exploited process without having to create a new process

Generating Shellcodes using `msfvenom`

Proof of Concept

Things To Do



If you have problems visit me in counseling hours. . . .