**Reverse Engineering HW4:**

**Mori Levinzon 308328467**

**Omri Klein 318733565**

**Dry part:**

1. It is possible to get this gadget. If we would be scouting the code after the combination of the two commands opcodes (60 and C3) ,we may took part of another command in order to find the opcode 60 which is not so common.

For example : if we had the following commands at the end of some function in the code:

add    esp,0x60

ret

this is two commands that we can find in allot of function (as we need to return the stack pointer to it's original state before we reti=urn from the function).

the opcode of the two commands is translated to :

83 c4 60

C3

And the last two opcode is exactly what we need.

1. Here is a look at the stack :

|  |  |  |
| --- | --- | --- |
| 1 | 9ad9e706 | xor eax,eax |
| 2 | 9ad9e704 | pop ebx |
| 3 | 70707061 |  |
| 4 | 9ad9e700 | Add eax, ebx |
| 5 | Deadbeef |  |
| 6 | Deadbeef |  |
| 7 | Deadbeef |  |
| 8 | 9ad9e716 | Mov eax, dword ptr[eax+0fh] |
| 9 | 9ad9e711 | Inc eax |
| 10 | 9ad9e711 | Inc eax |
| 11 | 9ad9e709 | Mov ecx, eax |
| 12 | 9ad9e716 | Mov eax, dword ptr[eax+0fh] |
| 13 | 9ad9e703 | Pop ecx |
| 14 | 9ad9e71c | pusha |
| 15 | Virtual Protect address |  |
| 16 | 9ad9e71a | Pop edi |
| 17 | 9ad9e704 | Pop ebx (will go to edi) |
| 18 | 9ad9e714 | Pop ebp |
| 19 | 9ad9e702 | Pop ebx (will go to ebp) |
| 20 | 9ad9e71c | pusha |
| 21 | 00001000 | dwSize of page |
| 22 | 40000040 | R\W\X permission |
| 23 | 00025000 | lpfOldProtect |

First, we cleared the eax register, popped ebx and added it to eax, dereferenced it's value and increased eax in order in order to store the address of the start of the page in eax register (commands 1-12).

After that, we put in inside the ecx the address of the gadget that does pusha command and stores the virtual protect address in the ebp register..(commands 13-15).

After the "pusha" gadget is called the stack look like this:

|  |
| --- |
| Edi-pop ebx |
| Esi |
| Ebp- POP |
| Ebp- POP |
| Ebp- POP |
| Esp |
| Ebx=VP |
| Edx |
| Ecx-pusha |
| Eax-address |
| 00001000 |
| 40000040 |
| 00025000 |

So after ret will be excecuted to edi, esi value will pop to ebx. After that will do ret to ebp 3 times , ending in ebx storing edx (junk value) and ecx will have Vitual protect address (ebx old value). now we will do ret inside ecx – pusha and will push all the register to the stack the will look like this:

|  |
| --- |
| Edi-pop1 |
| Esi |
| Ebp-pop |
| Ebp-pop |
| Ebp-pop |
| Esp |
| Ebx |
| Edx |
| Ecx-VP |
| Eax-address |
| Eax-address |
| 00001000 |
| 40000040 |
| 00025000 |

We can noticed that both edi and ebp has not changed and ecx updated to store the address of virtual protect. Now as we did before will do ret to the ecx register (after the pops) and move to the Virtual Pprotect address. We can see the return address of VP is now the start of the page and it's parameters are:

lpadress=start of page,

dwSize=00001000,

flNewProtect=40000040,

lpfOldProtect=00025000

SO virtual protect changes the page permissions to r/w/x. saves the old permissions in 25000 and after the return to the start of the page which is exactly what we wanted.

The value of the new Permissions should be 40000040 since inserting only 40000000 will be problematic since it containing null values (00).

1. We saw in the tutorial that there are some things that we can’t write when implementing stack overflow such as null (00), ‘\n’ (0A) and more.  
   We can see that we need to use the value of ‘\n’ – 0A.
2. Because we can’t use this value, we will enter the value F6 to the neg function. This way, we won’t enter the value 0A in the buffer, but the value F6. Now, before we will execute anything, we can use the neg function to get the wanted value – neg(F6) = 0A.

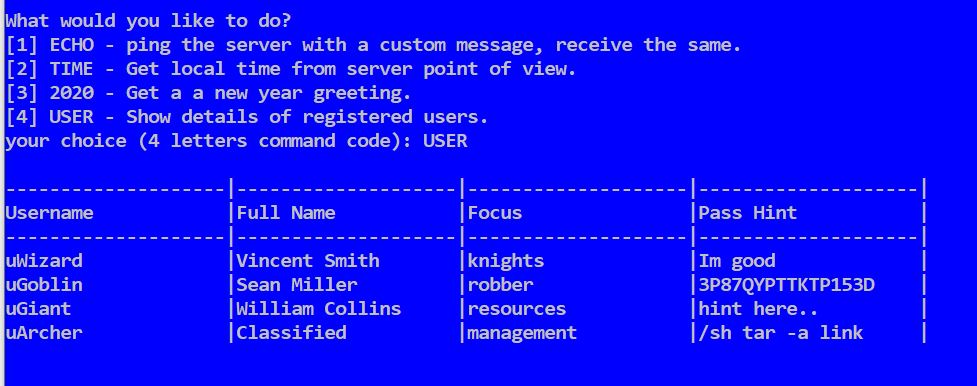
This will change the rop chain be inserting FFFFF000 instead of 00001000 than negating it using the rop, thus preventing us from inserting NULL char, and give us the correct value we wanted to be as the dwSize.

**Wet part:**

Part 1:

At first glance of hw4\_client.exe trough IDA everything seems in order. There is a request for username and password that is checked through the server using the methods we already saw (send and recv) . after going trough couple of check up to find to any vulnerability, we found out that whilst the username length is limited to up to 31 letters there is no such limitation on the password length, and therefore the exe is exposed to buffer overflow. We needed a way to pass trough the username and password checks that are been done after we are returning from the function. So we decided to use the buffer overflow to overwrite the return address from the function and instead of making the function return to the rest of the code (and the authentication check ) in address 401C3B we decided to changed it to loc\_401CAE, after the authentication part and on to printing the menu and the rest of the desired user table. We wrote python script the opens a process and insert to it "MENU" as a user name in order to make the program print the menu in the branch later on located next to loc\_401CAE. Afte that we needed to insert a password. The problem is that the password address is located higher than the username address, so in order to reach the return address and changed it we needed to assemble the password carefully, avoiding rewriting the username in the processs. We padded the start of the menu with couple of "A" till we reached the username address where we kept it as is ("MENU" with null), after that we kept on padding with "A" till we reach the return address and replaced it with "401CAE".

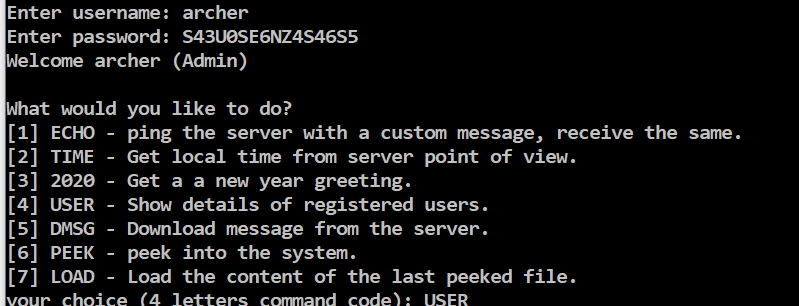
Running the users.py and requesting the users table file gave us the following result:



Part 2:

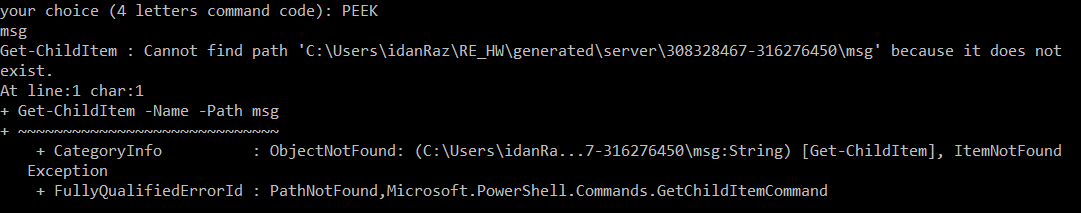
As we we saw from the table printed to us the only given password is the one that belongs to the goblin. We tried to log in with the goblin as a username and his password and got the same table, but this time it had an additional option: DMSG. The same option that prints the message that we decypherd in the previous assignment.

We noticed that the goblin password is the same one that we found in the 1st assignment, so we tried to log in with all of the accounts. The Wizrds and the giant showed us the same menu and messages as the goblin did. But when we logged using the archer account we found out that he is an admin and there fore has more permissions and his menu contained more options:



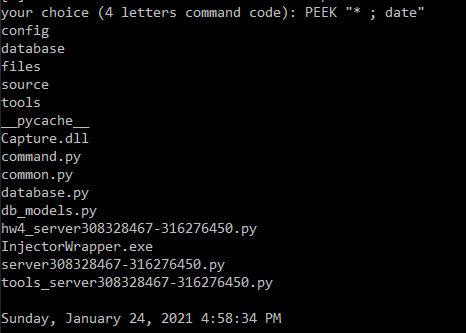
Part 3:

Part 4:

To find out what PEEK is doing, first of all we ran the command, we can see that it’s waiting for some input. We try the input “msg” and got an error:  


From the error, we can see that the command tries to find a path. So, the next thing we tried is ‘\*’ (which means all). The command PEEK with the value ‘\*’ showed us all the files\folders in the server (C:\Users\idanRaz\RE\_HW\generated\server\308328467-316276450).  
We found out that we can also navigate using this command – for example: the command PEEK tools will show us the content of the tools folder.  
One more thing that we can do using the command PEEK is to peek a file and then use the LOAD command to see the content of this file.

We now know that PEEK runs a command. We want to run other commands. We will user basic injection for this in the following way:  
We call the command PEEK and, in the input, we will insert **“ \* ; <command>”**.  
Explanation: The char ‘\*’ is the value for the PEEK command, the char ‘;’ represents that we finished a command and after that we can write the command we want to run on the server. For example:

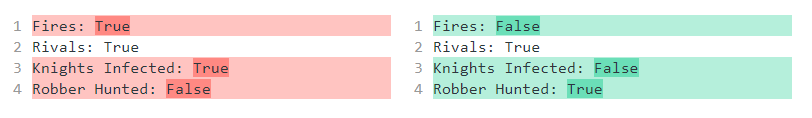


In the example we can see that the value we insert to the PEEK function is a ‘\*’ (as we said before) and then we run another function – date. We can see the output of date after the output of the PEEK.

\*\* A short description about the implementation in the python file \*\*

Part 5:

We want to find a way to stop the knights and extinguish the fires. We looked at all the files and one of the files we saw was attack.config. This file is interesting because we can see in the file information about the fires and about the knights (which we could not see anywhere else). We can see, for example, that the value of the fires is True and the value of Knight Infected is True. We want to change the values of this file because we do not want the knights to be infected and we don’t want fires on the board. We want to do the following changes:



From the previous part, we know that we can run commands on the server. Let’s insert the command that updates the config file. The commend is:

echo 'Fires: False\nRivals: True\nKnights Infected: False\nRobber Hunted: True' > config\attack.config

Of course, we need to put the command in the following way:

PEEK "config\attack.config ; echo 'Fires: False\nRivals: True\nKnights Infected: False\nRobber Hunted: True' > config\attack.config"

Note that in the beginning, we can write anything we want (instead of “config\attack.config”). The important part is the command after the ‘;’.

We can see that the file updates! We logged into the website (with archer user) and saw that the fire is out.

\*\* A short description about the implementation in the python file \*\*