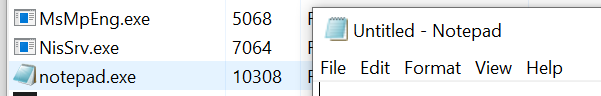
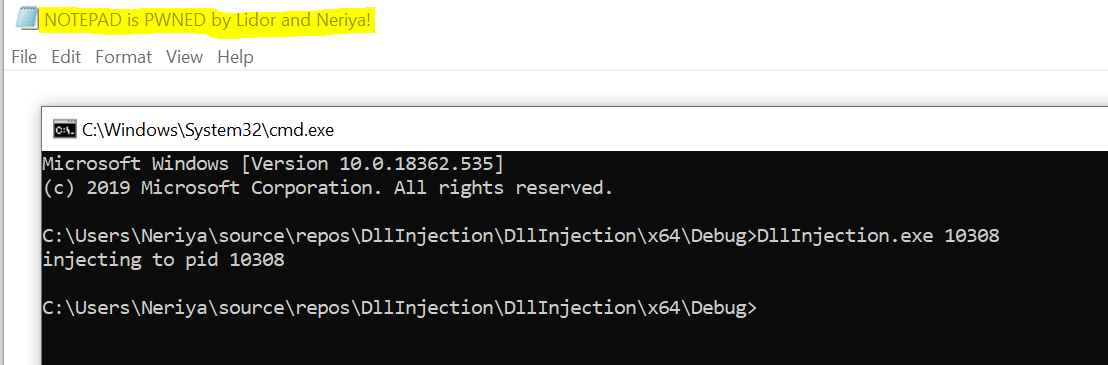
**Exercise 1: Dll Injection**

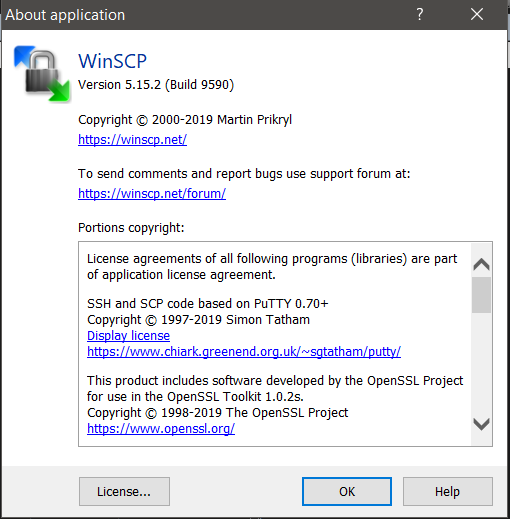
1. We wrote a dll that changes the window title of the current process when the dll is attached to a process.  
In order to do that we used **EnumWindows** to enumerate over all the open windows and find the window that related to the current process (which is the process the dll injected to).  
Then we used **SetWindowText** in order to change the window text.

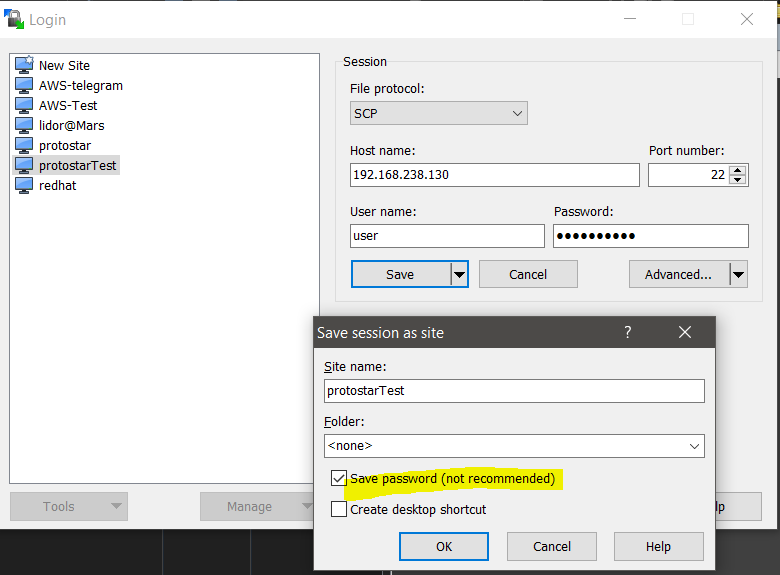
2. In order to make this code run, we needed an execute file to inject this dll into notepad.exe.  
We received a pid as argument and used **OpenProcess** to get handle to this process.  
Then we used **GetModuleHandle** and **GetProcAddress** in order to get the address of **LoadLibrayA** function. In that stage we needed to inject a string which is the path of the dll, we did that using **VirtualAllocEx** and **WriteProcessMemory**.  
Finally we called **CreateRemoteThread** to make **LoadLibrayA** run on the remote process.  
**Screenshots**:  
Find notepad pid:  
  
Run the injector:  


**Exercise 2: IAT Patching**

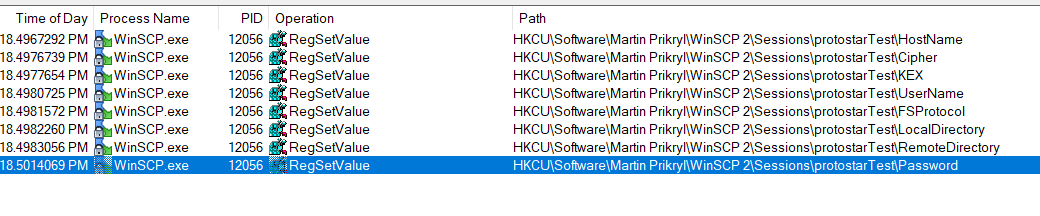
**Part 1**

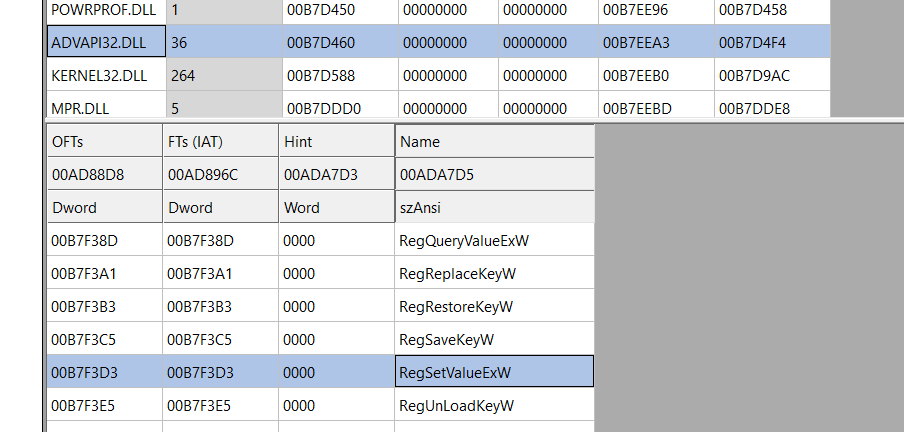
We chose WinScp.exe as Windows application which allow to user to connect to SSH server and move file to/from the server.



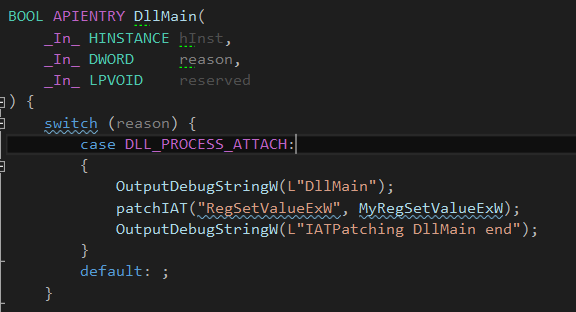
WinScp allow the user to store the password to the server in order to make it easier to connect to the server again:

We used procmon and found out that registry is used in order to store the passwords that the user choose to save:

We used CFFexplorer and realized that the WinAPI function **RegSetValueExW** from **ADVAPI32.dll** is used to save the password in the registry:



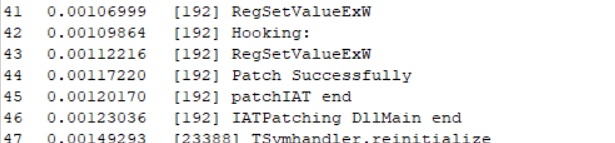
We wrote DLL that Patch IAT. it patches the IAT and call to **MyRegSetValueExW** instead of **RegSetValueExW:**



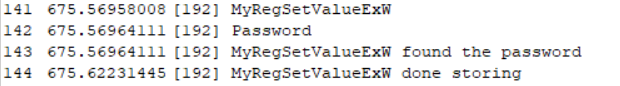
Our function, **MyRegSetValueExW,** just store the password in c:\output.txt, and return the call to the original **RegSetValueExW** in order to not breaking winscp application from operating successfully:



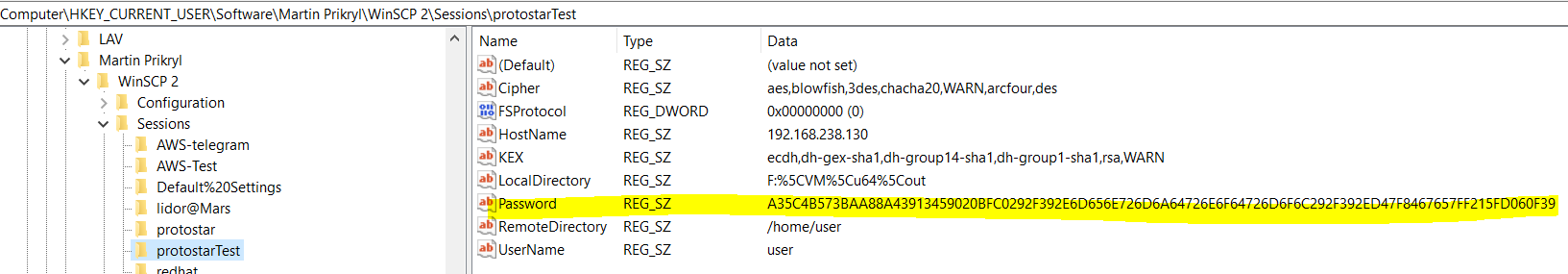
Now we tested the code; we used OutputDebugString to debug our code and view the debug messages from dbgview. The IAT Patching worked succefully:

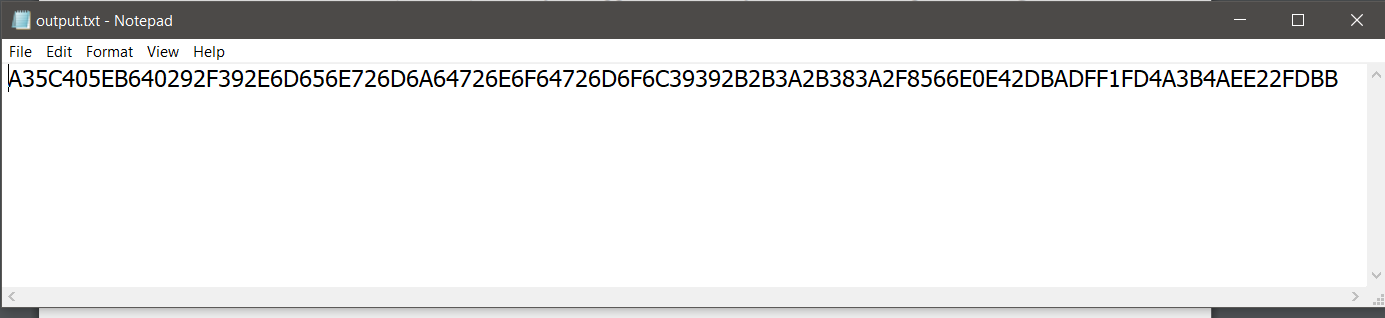


Then we store the password, and according to debug messages it works!



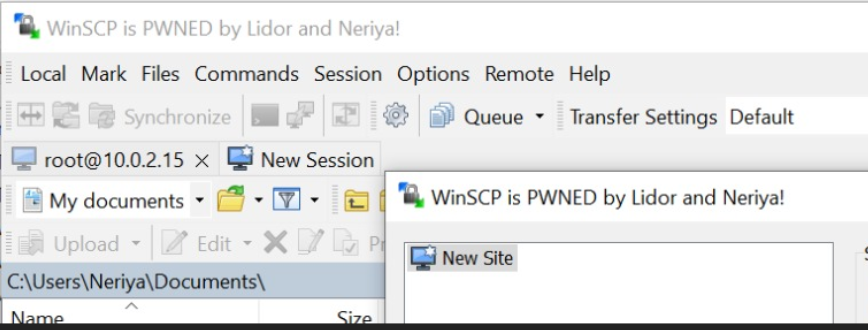
We also check that the contents saved to registry is same as contents saved to c:\output.txt file:





And it is the same!

Also we changed the application's title bar:



**Part 2**

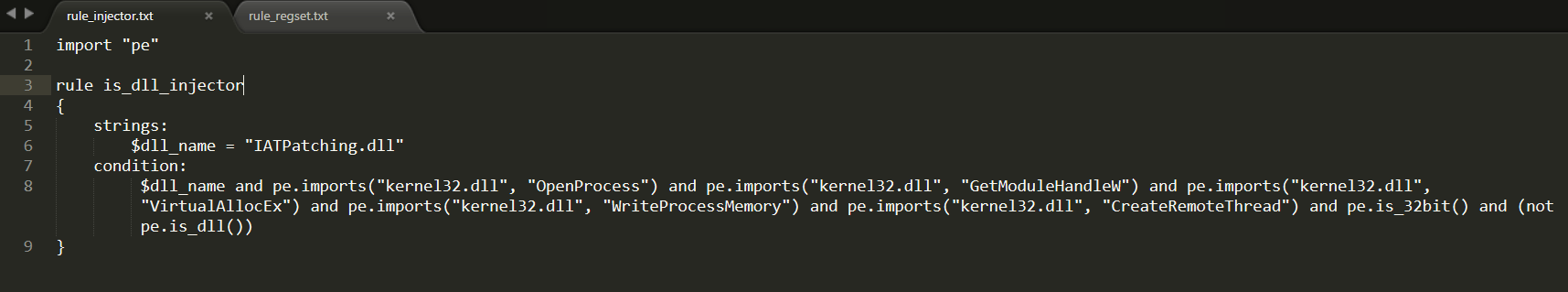
**a)**

We started with YARA rules for the **Injector.exe**:

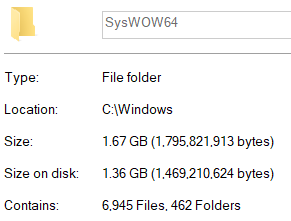
We know its inject DLL name IATPatching.dll so we will search for this string.

We know that in order to do DLL injection we need to use WINAPI: OpenProcess(to get process handle), GetModuleHandleW, VirtualAllocEx, WriteProcessMemory(to copy DLL name to the other process), and CreateRemoteThread (to run code in the other process).

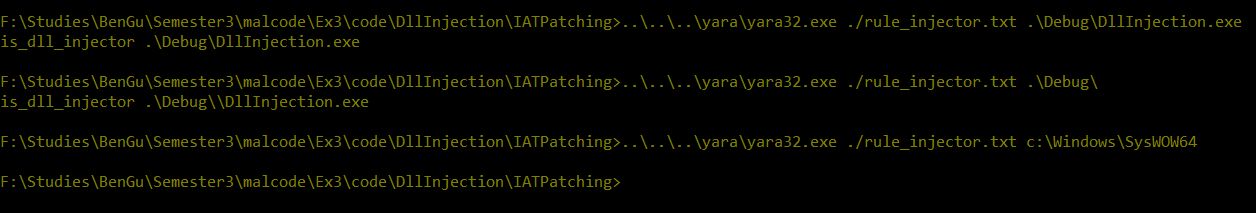
We also know that it is 32 bit executable.



We check this rules on c:\windows\sysWOW64 which contains ~7K PE files:



And our rule is 100% precise; its only detect the malicious DllInjection.exe:



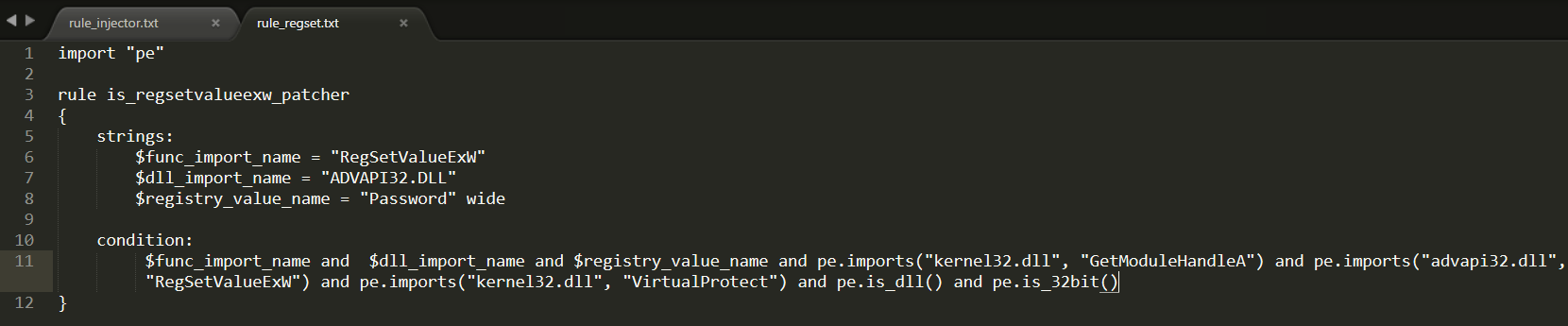
Then we wrote YARA rules for **IATPatching.dll:**

We know its patch the IAT for the WINAPI **RegSetValueExW** so we will search for this string. We also know that it search this string is the DLL **ADVAPI32.DLL** so we will search for this string also.

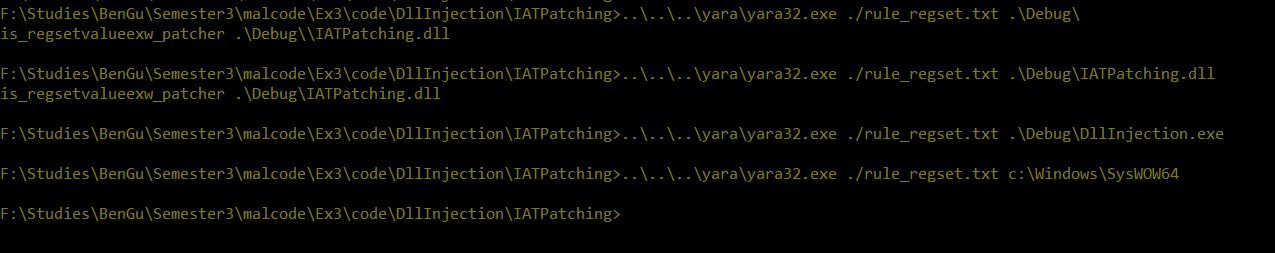
We know that in order to patch the IAT we need to use WINAPI: GetModuleHandleA(to gat base image of the Winscp process), VirtualProtect(to edit the IAT memory), and RegSetValueExW (to return the value of the original WINAPI).

We know that the value that its hook its registry writes is **Password** (in wide-string),so we will search for this string.

We also know that it is 32 bit DLL.



We check this rules on c:\windows\sysWOW64 which contains ~7K PE files, and our rule is 100% precise; its only detect the malicious IATPatching.dll:



**b)**

Both of these rules(rule\_injector.txt, rule\_regset.txt) are depends on 3 types of conditions, that all of them need to occur:

1. Dynamic API imported
2. Strings in file
3. Executable features(DLL vs Executable, 32/64 bit)

Malicious author can avoid being detected by our 1. rules by using Runtime Linking- he can use LoadLibrary and GetProcAddress and we won’t know what functions it imports.

Malicious author can avoid being detected by our 2. rules by using obfuscation - he hides his strings using symmetric encryption and the executable on disk will contains the encrypted strings. When his code needs the string, he use decryption to get the original string. This way- the PE wont contains the string and it won’t be exposed by our YARA rules.

The third type is very generic, and the malicious author won’t need to overcome this because almost all windows application is dll/exe or 32/64 (this rule doesn’t indicate of malicious software).