Practical Malware Analysis and Triage Malware Analysis Report

Sample: Dropper.VBScript.vba.mal.7z
Source Code Analysis

November 15, 2023

Muhfat Alam

Summary

Inside the zip Dropper.**VBScript.vba.mal.7z** file, there are three different files were saved. Those are **crtupdate.vbs**, **one.crt**, and **two.crt**. MD5 and SHA256 hash value for **Dropper.VBScript.vba**

For this report, we are going to work on **crtupdate.vbs** sample. The hash value for the **crtupdate.vbs**

MD5: 0d9a977f3a20f7f17bccbf1ab917672e
SHA256: 3ff0f51ec1b0f3e2d4c9685c52a1d0605435288b53f54cd048b28104c7539959

DNS Record: N/A

Network Connection: N/A

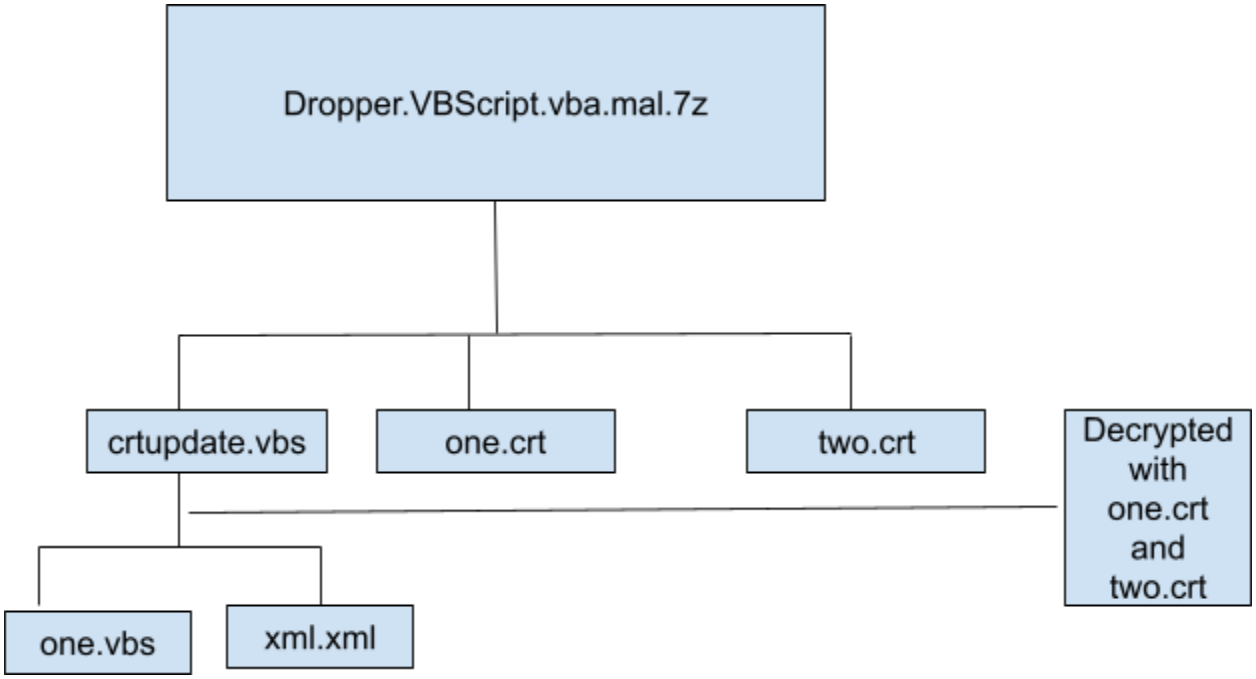
File Written/Read/Execute: crtupdate.vbs, one.vbs, xml.xml

Technical Summary

Through our analysis, it appears after running the **crtupdate.vbs**, two files will be created in the **C:\Users\Public\Documents** directory by decrypted with **one.crt** and **two.crt**.

1. one.vbs
2. xml.xml

After further analysis, **one.vbs** is an obfuscated **vbs** script. After deobfuscating the code, it spawned to **xml.xml** code that is written in C# language which might be built using **MSBuild.exe** from **one.vbs** to run the malicious shellcode to perform the malicious function of adding "**wsadmin**" user, adding to the local group of remote desktop users and administrator.



Static Analysis

Get the File Hash:

Get the hash value for the source code of **crtupdate.vbs** on the cmder by running the following command (Figure 01),

```
sha256sum.exe crtupdate.vbs //in sha256 hash
md5sum.exe crtupdate.vbs //in md5 hash
```

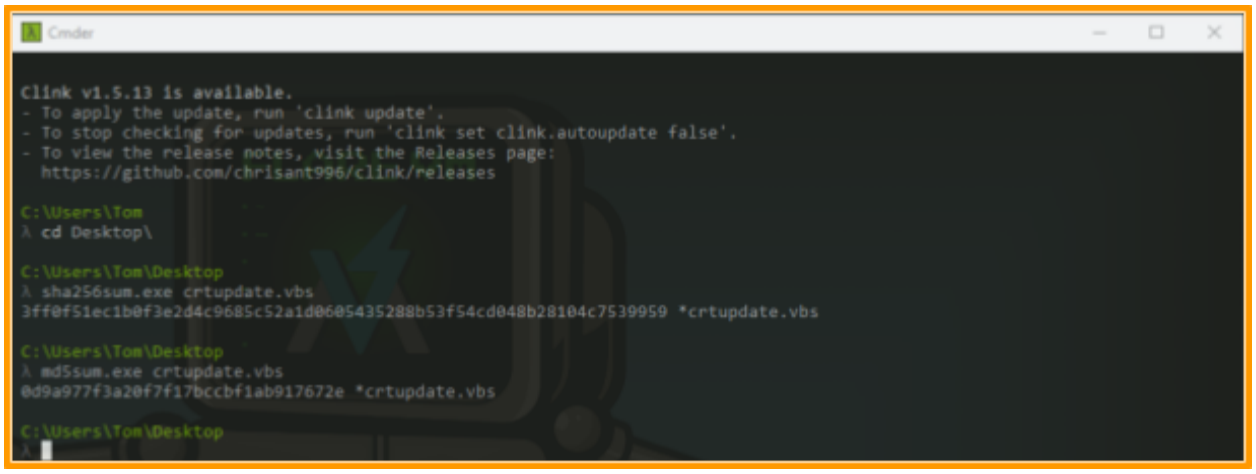


Figure 01

OSINT Tool: As we have the hash value from the command line, we can use some OSINT Tools, like VirusTotal and MetaDefender for any flagged by security vendors.

VirusTotal Verdict: There are **4/59** security vendors flagged this **crtupdate.vbs / 3ff0f51ec1b0f3e2d4c9685c52a1d0605435288b53f54cd048b28104c7539959** as a Trojan Threat of sagent / vsnw15b23 family. (Figure 02)

Figure 02

MetaDefender Result: Thread name identifies as **Trojan/Sagent.** (Figure 03)

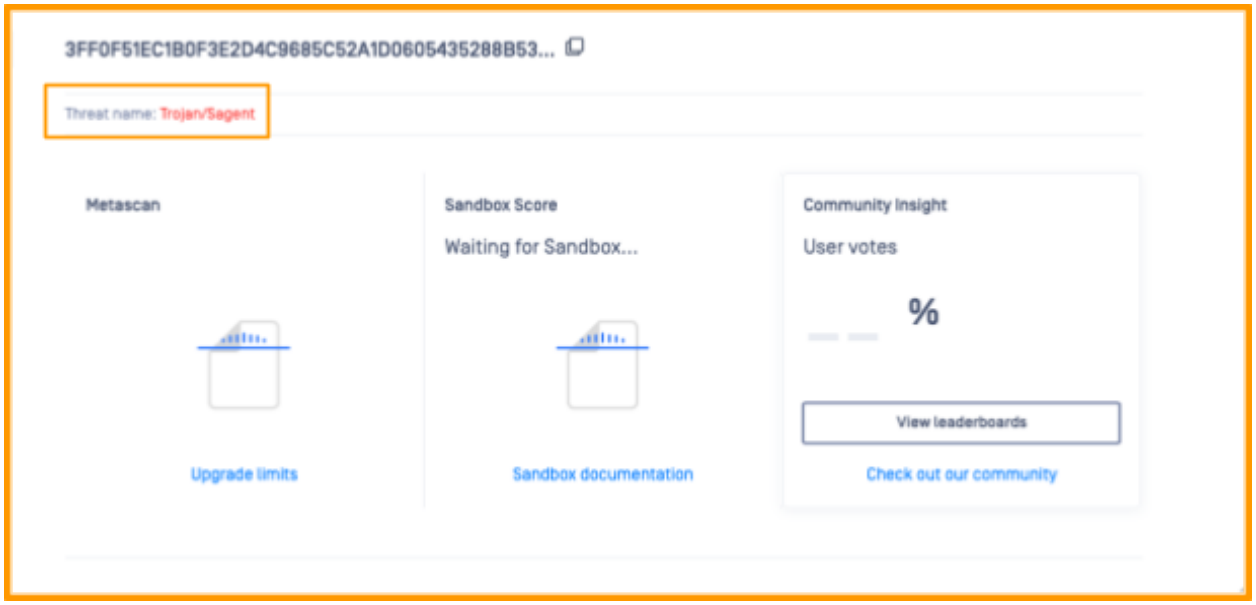


Figure 03

Code Analysis:
After opening the **crtupdate.vbs** file in the **VS Code Editor**, it is evident that the programming language used is **Visual Basic Scripting Language**. (Figure 04)

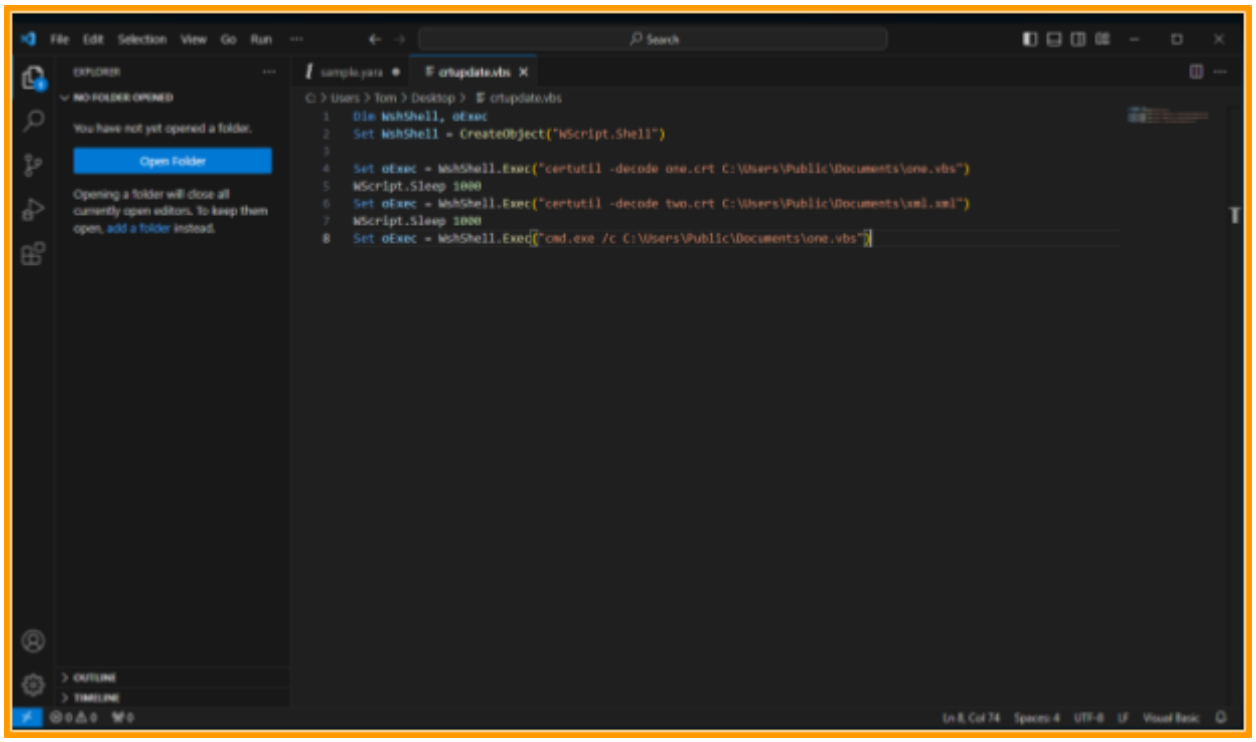


Figure 04

Visual Basic is a scripting language that's built into the Windows OS that allows someone to write scripts that are more functional than a batch script because we can tap into powerful primitives in the OS to do different scripting things.

Inside of the **crtupdate.vbs** script, Windows script was invoked to create a shell object. Then **"WshShell.Exec"** is used to call two "certuil", one is decoded with one.crt and saved into the **C:\Users\Public\Documents** directory. Another one also decodes with **two.crt** and saves into the same directory. It will sleep in between those calls. Then the final call is to execution of "**cmd.exe**" and point at one.vbs.

Dynamic Analysis

For dynamic analysis, we need to run the **crtupdate.vbs** script and note some unexpected behaviors.

After running the file **crtupdate.vbs** with a double click, we see cmd pop up a couple of times and disappear. Now if we notice inside the **C:\Users\Public\Documents** directory, we can see there are two new files are created. Those are "**one.vbs**" and "**xml.xml**". (Figure 05)

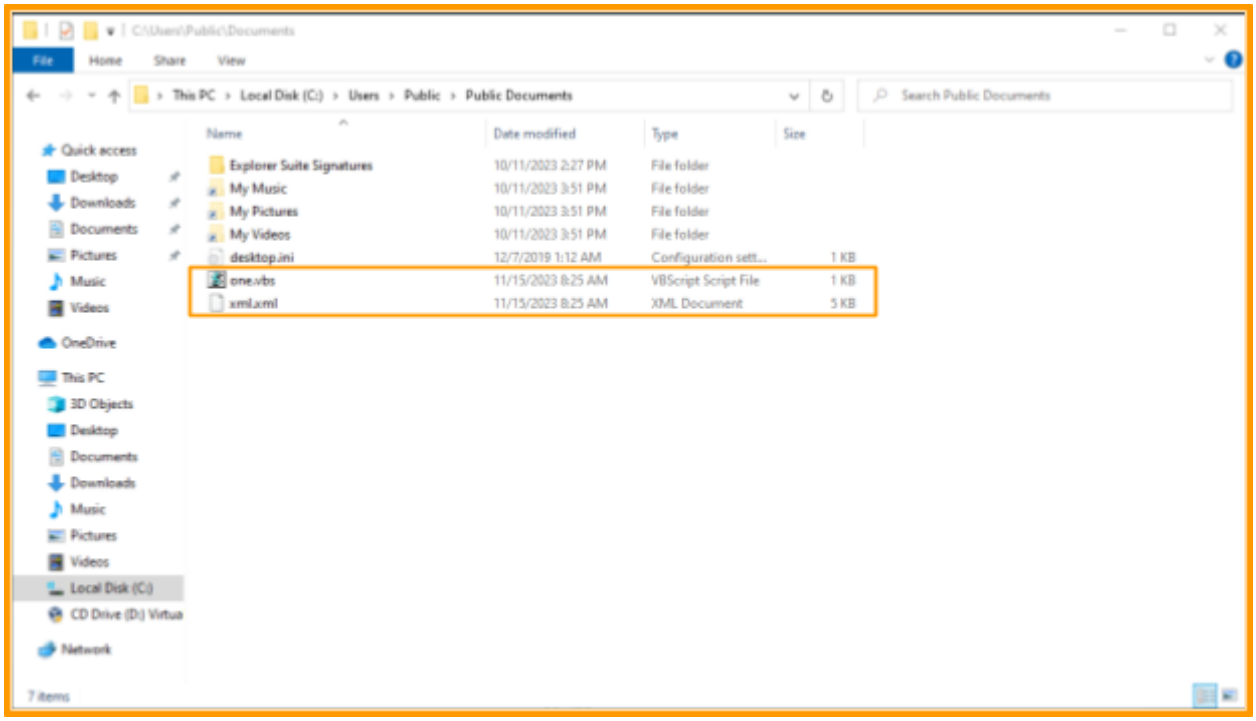


Figure 05

Analyse “one.vbs” code inside the VS Code Editor:

Through the analysis process, we open up the **one.vbs** and **xml.xml** script with Visual Studio Code Editor and understand the process.

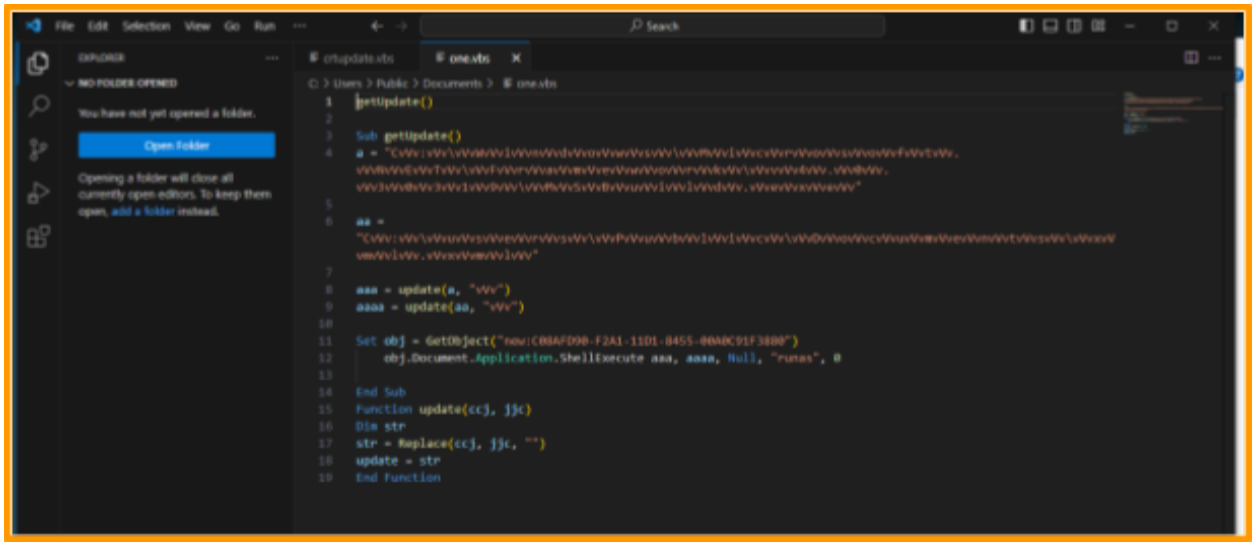


Figure 06

In the **one.vbs**, there is a call function “**getUpdate()**” which is defined as a subroutine in the next line. This subroutine has two string values, one is “**a**” and “**aa**” which are garbage text to confuse users. Then we have an “ **update**” function which is passed two values (“a” and “aa”).

After that “**Set obj = GetObject("new: CO8AFD90-F2A1-11D1-8455-00A0C91F3880**" this line creates an object using the “**GetObject function**”. The object is created with the class identifier "**CO8AFD90-F2A1-11D1-8455-00A0C91F3880**." The purpose of this object is not clear yet from the provided code snippet. But then it will use that object to “**ShellExecute**” the contents of “**aaa**”, “**aaaa**” and passing it to “**runas**” as a parameter.

“**aaa**” is the updated version of the single “**a**” and “**aaaa**” is the updated version of “**aa**”.

In line 15, the update function takes in two arguments “**ccj**” and “**jjc**”, then in line 17, it just replaces the value of “**aaa**” and “**aaaa**” which is the updated version of “**a**” and “**aa**” with an empty string. This is a string builder, string obfuscation routine.

After deeper analysis, we can make a copy of this file and replace the “**vVv**” with “ ” (**empty**) string, we can see much more clear text. (Figure 07)

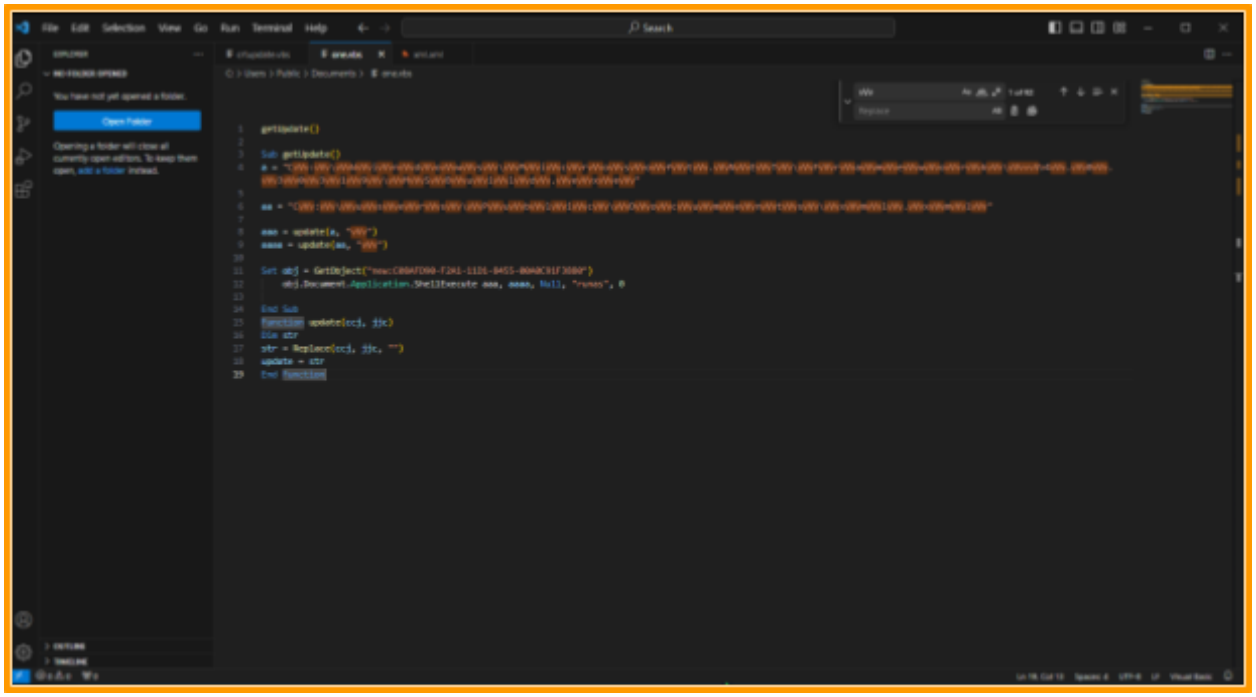


Figure 07-01

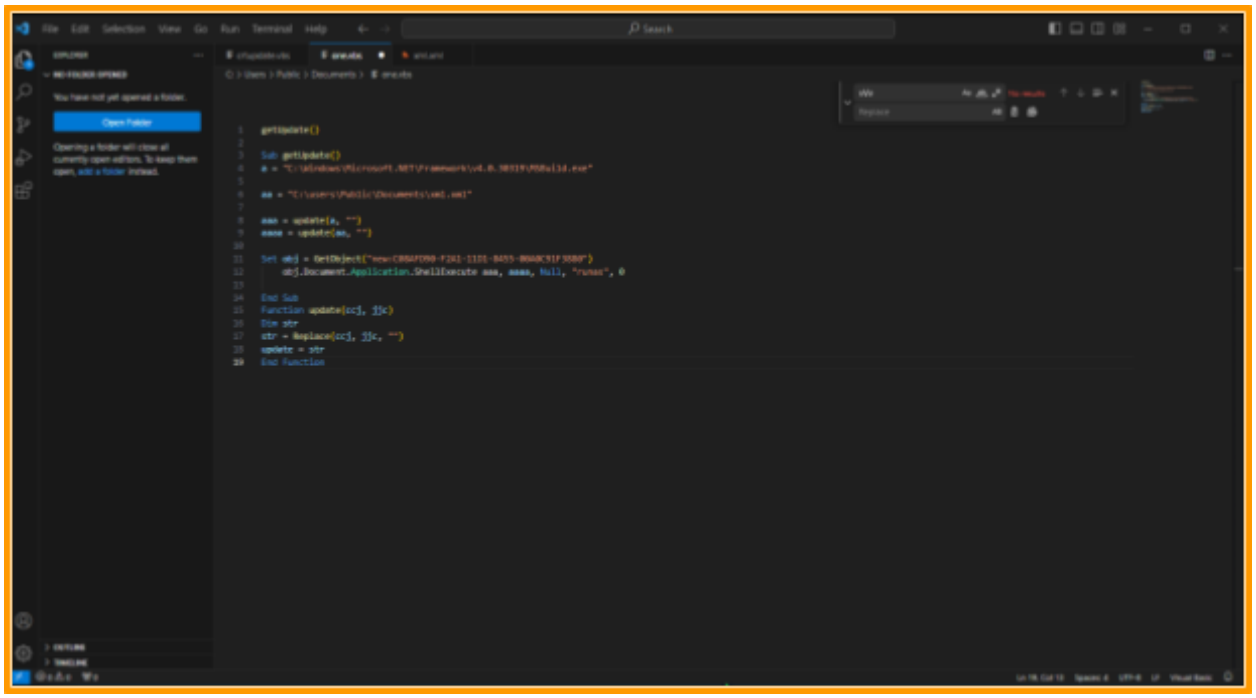


Figure 07-02

After the “**update**” function has run, we are now passing in two strings which means "**C: \Windows Microsoft.NET\Framework\v4.0.30319\MSBuild.exe**" with passing along with "**C: \users\Public\Documents\xml.xml**".

After further analysis, "**new: C08AFD90-F2A1-11D1-8455-00A0C91F3880**" is a class ID.
Based on the "**Shell.ShellExecute method**" on the Microsoft documentation website, it invoked the same way the documentation mentioned. In the arguments,

So the "**sFile**" argument is a string that contains the name of the file where "**ShellExecute**" will perform which matches our **one.vbs** code with "**aaa**" in line 12.

The second part in the "**vArguments**" is a string that contains parameter values for the operation. So for **MSBuild.exe**, if there are any arguments required to actually use **MSBuild.exe**, we pass those in "**aaaa**" which is an updated deobfuscated version of **C:\users\Public\Documents\xml.xml**.

Then we have "**vDirectory**" which is a null value in our code.

After that, "**vOperation**" will performed. This value is set to one of the verb strings that are supported by the file.

In the end, the "**vShow**" is a recommendation as to how the application window should be displayed initially. In our **one.vbs,** the value is set to 0. which will open the application with a hidden window. That matches the behavior while we ran this program.

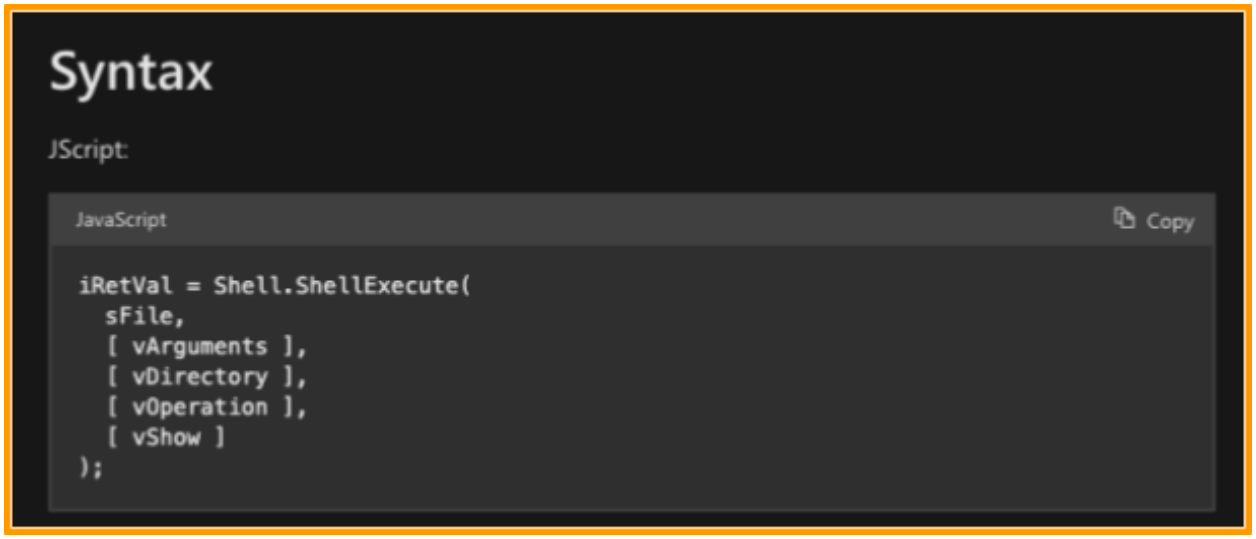


Figure 08-01

Parameters

sFile [in]

Type: BSTR

A **String** that contains the name of the file on which **ShellExecute** will perform the action specified by *vOperation*.

vArguments [in, optional]

Type: **Variant**

A string that contains parameter values for the operation.

vDirectory [in, optional]

Type: **Variant**

The fully qualified path of the directory that contains the file specified by *sFile*. If this parameter is not specified, the current working directory is used.

vOperation [in, optional]

Type: **Variant**

The operation to be performed. This value is set to one of the verb strings that is supported by the file. For a discussion of verbs, see the Remarks section. If this parameter is not specified, the default operation is performed.

vShow [in, optional]

Type: **Variant**

A recommendation as to how the application window should be displayed initially. The application can ignore this recommendation. This parameter can be one of the following values. If this parameter is not specified, the application uses its default value.

Value	Meaning
0	Open the application with a hidden window.

Figure 08-02

```
Set obj = GetObject("new:C08AFD90-F2A1-11D1-8455-00A0C91F3880")
    obj.Document.Application.ShellExecute aaa, aaaa, Null, "runas", 0
```

sFile vArguments vDirectory vOperation vShow

Figure 09

Analyse “xml.xml” code inside the VS Code Editor:

Now that we have the deobfuscated code, we run the "**C: \Windows Microsoft.NET\Framework\v4.0.30319\MSBuild.exe**" with passing along with "**C: \users\Public\Documents\xml.xml**"

MSBuild.exe is the utility that’s used with **Visual Studio** so this is a kind of a stand-alone version of when you want to build something in **Visual Studio** and that could be anything from C# to C++ to a VBScript. MSBuild is like a command line invoked way to do that. So you passed the value of an **.xml** script that has the information that you want to build.

In this case, it is passing it in C# as a language and passing some other several codes. Those are invoking the value of these bytes (Figure 11) to execute this as ShellCode.

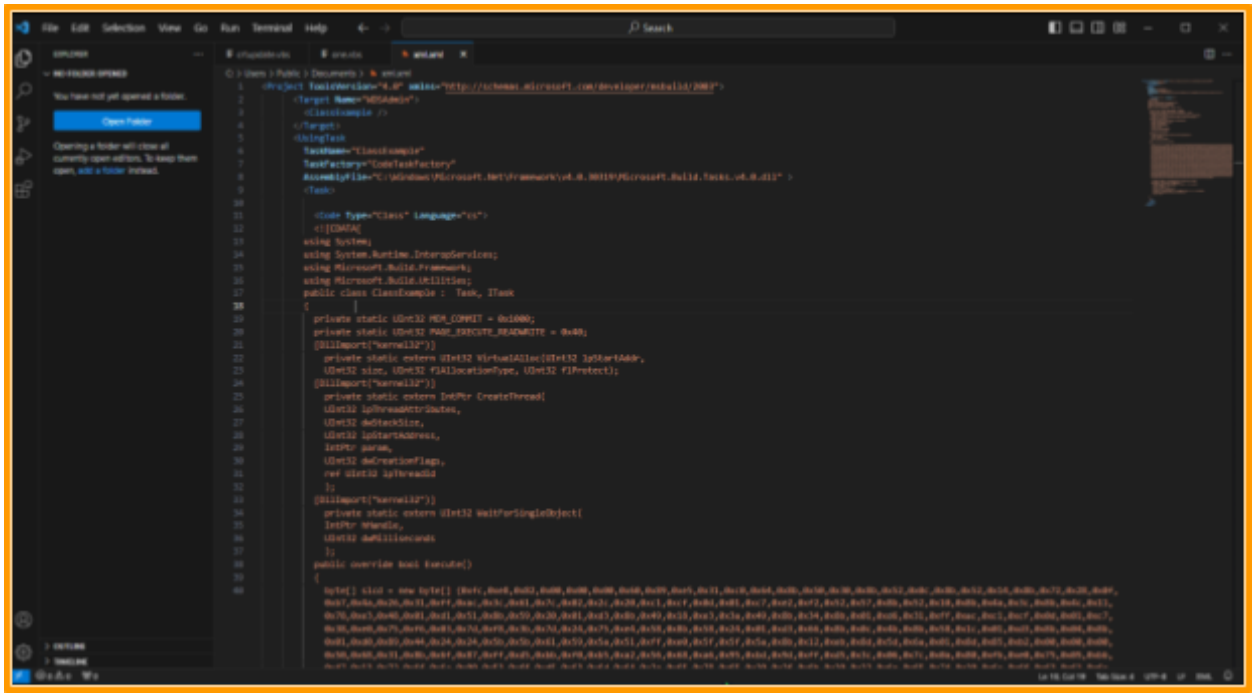


Figure 10

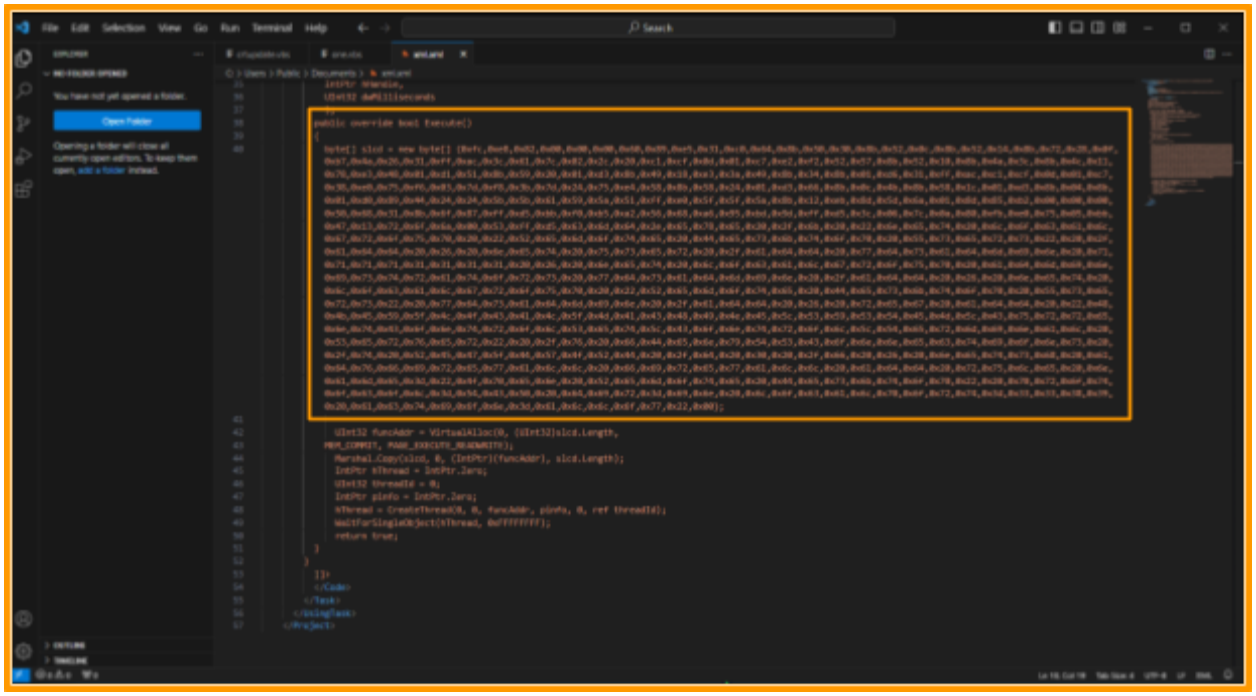


Figure 11

To analyze the shellcode, get the raw value of the data, copy the byte array into a text file (**ShellCode.txt**), and move it over to the **Remnux Machine**. Inside the **Remnux Machine**, we create a sample **Python code** (Figure 12) that removes everything except after “**0x**” which are two alphanumeric or numeric characters that are represented for the Hex Bytes.

Figure 12

Inside the Python code, give us a file “**out.bin**” in .bin format which will store the data of the byte everything except the raw hex value. (Figure 13)

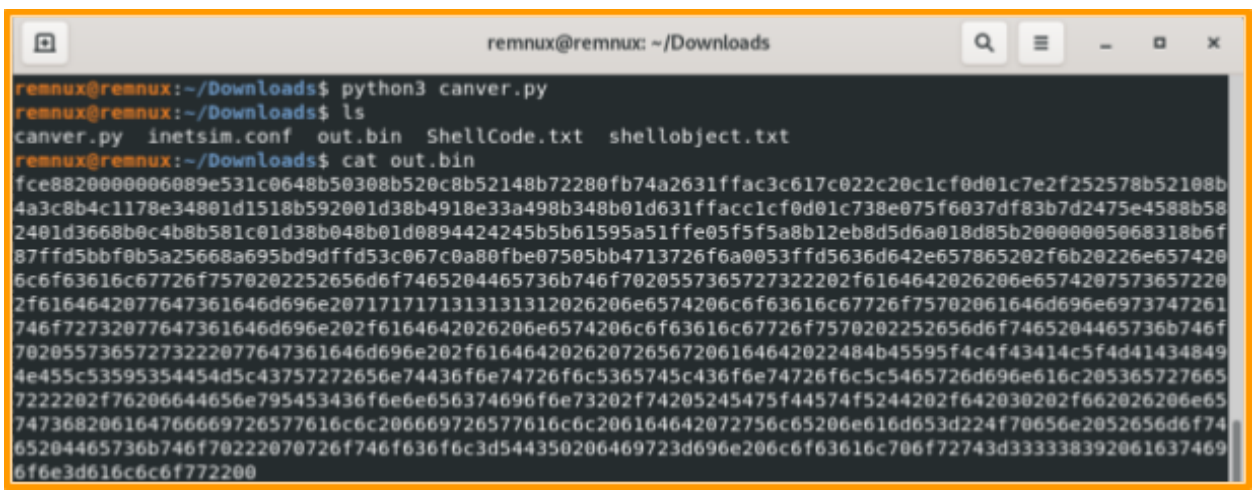


Figure 13

After getting the “**out.bin**” file in the Windows machine, we can use the **scdbg.exe** to see if any API is called or not.

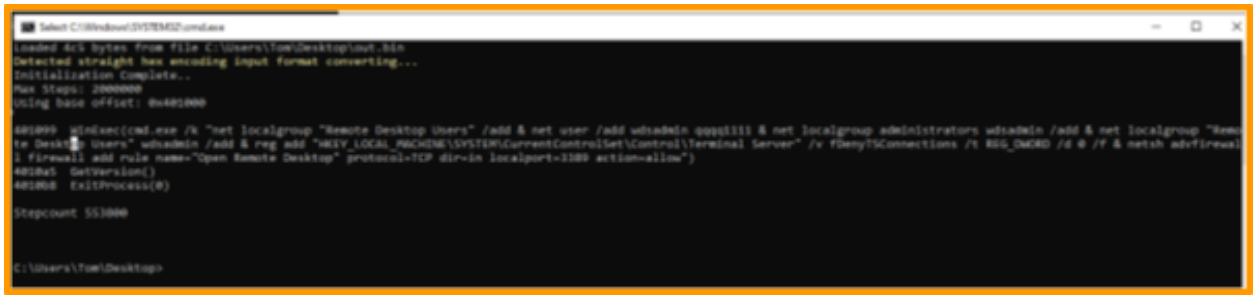


Figure 14

Based on the “**WinExce”** API utilizes the Windows command prompt to automate several tasks related to **user** and **group management**, **remote desktop access**, and **firewall configuration**. It creates a new user named "**wdsadmin**" with the password "**qqqq1111**," adds this user to the **local Administrators group** and the "**Remote Desktop Users**" group, and modifies the **Windows Registry** to allow **Remote Desktop** connections. Additionally, it establishes a firewall rule permitting inbound **TCP** traffic on **port 3389**, the default port for Remote Desktop Protocol.

Run the program by passing along with the argument:

We can run the "**C: \Windows Microsoft.NET\Framework\v4.0.30319\MSBuild.exe**" by passing along with "**C: \users\Public\Documents\xml.xml**" and monitor the process:

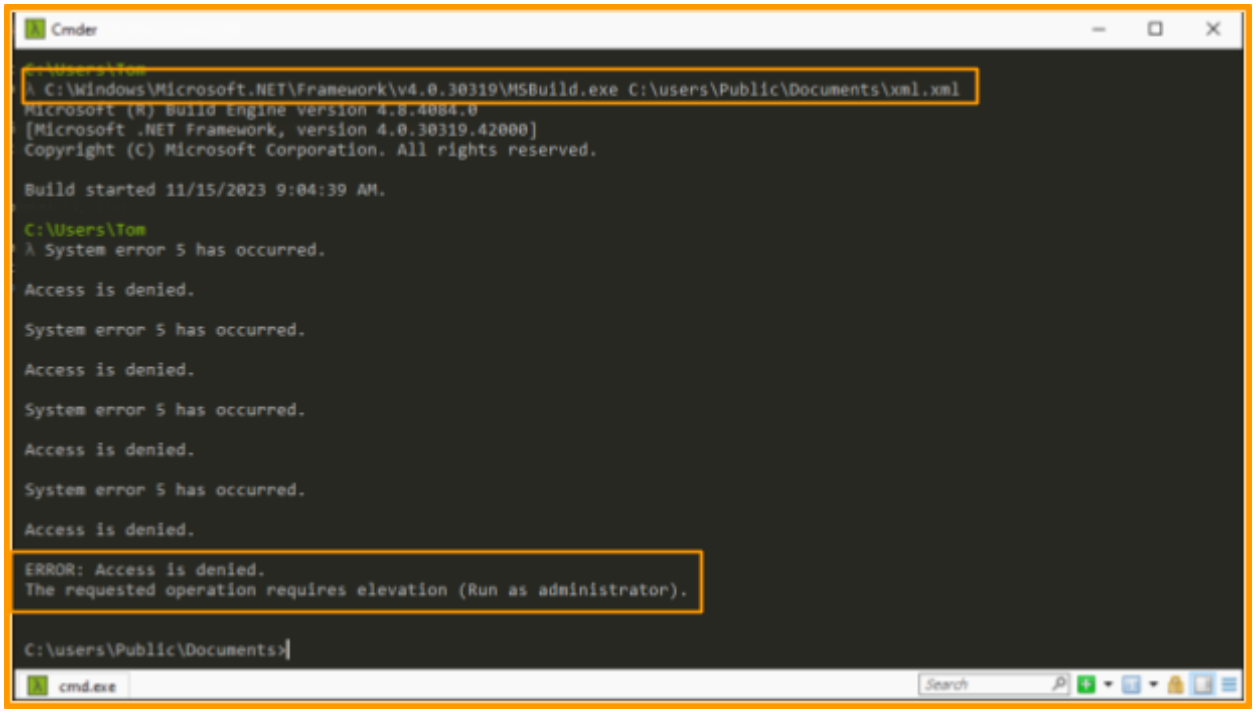


Figure 15: This process gets denied as this needs to Run an Administrator

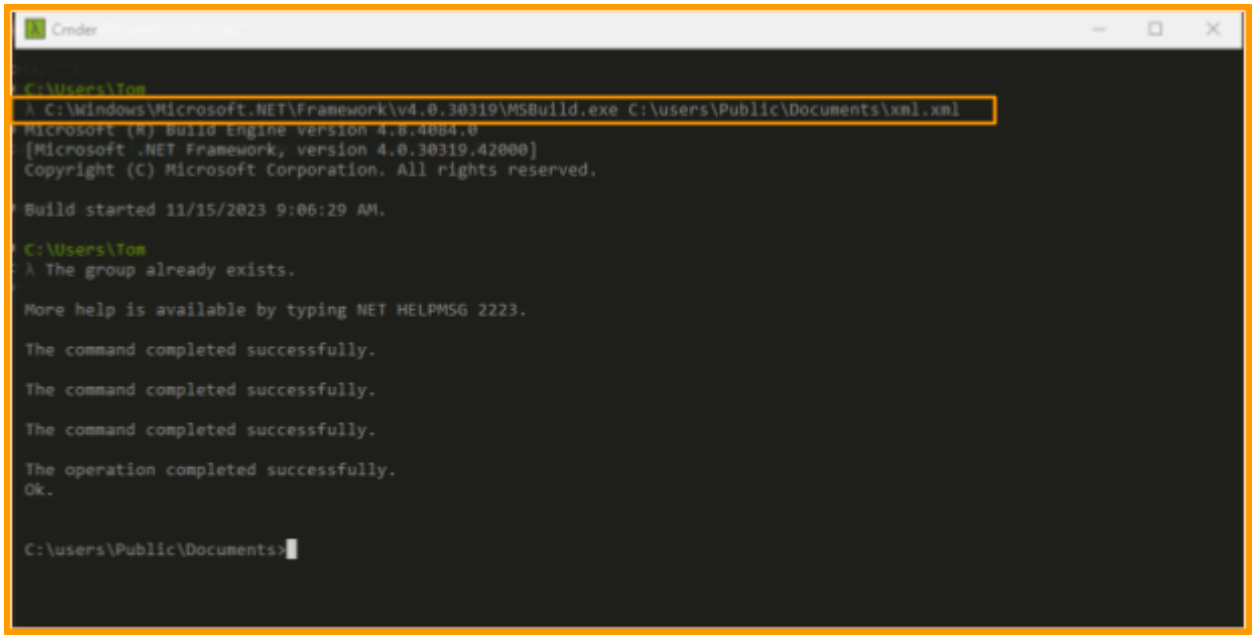


Figure 16: Run as Administrator

After running the process, we can see there is a user added as an administrator as well as in the group. This script invokes the shellcode that is passed into the byte code as we see earlier as a hex value (Figure 11). This ShellCode ends up adding a user to the remote desktop group, adding that user to the administrators' group, and opening up a port on the advanced firewall to open up the **RDP** session if is not already opened up. (Figure 17)

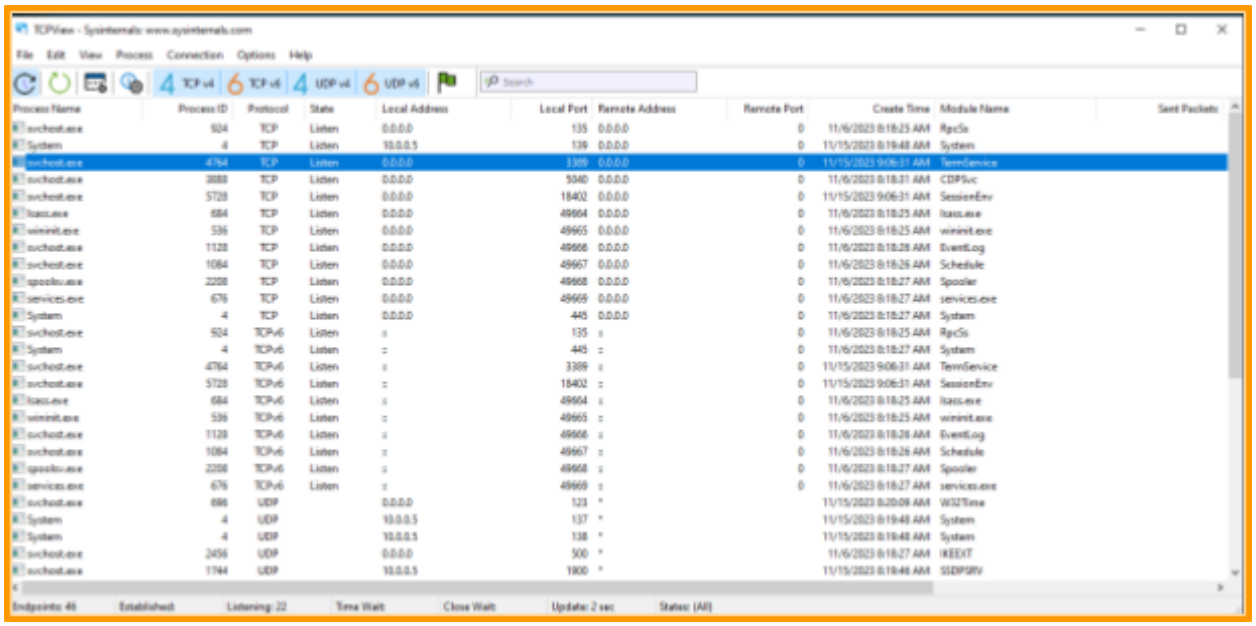


Figure 17

Now, if we check the user section before and after running the “**crtupdate.vbs**” in the cmder by checking

```
net user
```

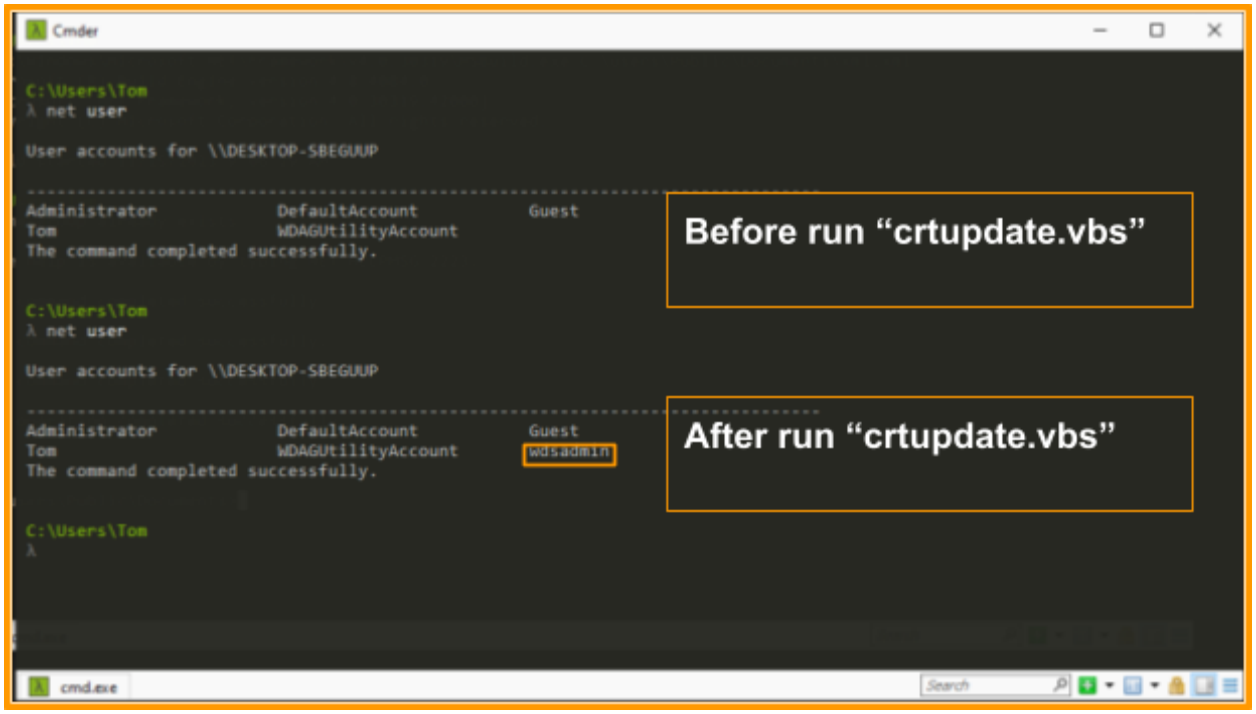


Figure 18: "**wbsadmin**" shows in the users' section.

Check the local group administrators for any additional users by typing

```
net localgroup Administrators
```

Figure 19: We can see the "**wdsadmin**" user has been added to the local administrator group.

Now if we check the Remote Desktop Users group by typing

```
net localgroup "Remote Desktop Users"
```

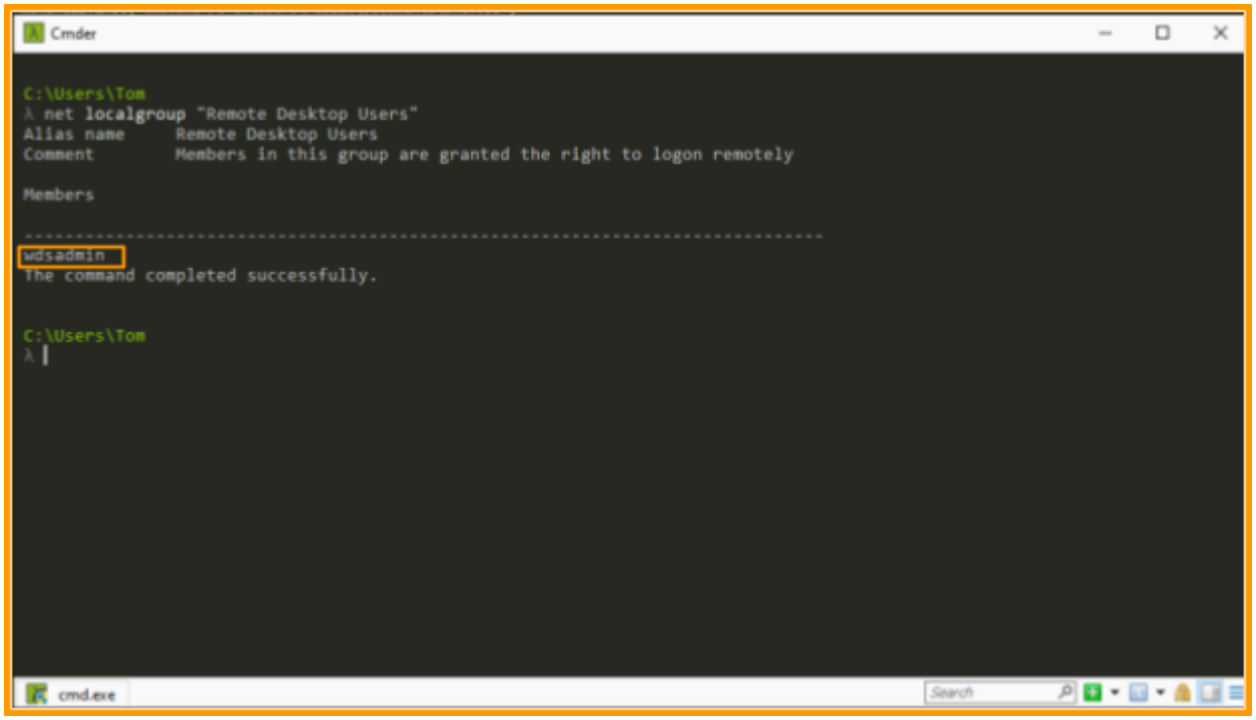


Figure 20

So, in the **xml.xml** code, all of the syntax here is set up so that the C# code that from line 13 to line 38 can be built using **MSBuild.exe**. And line 40 is the shellcode that's executed to perform the malicious function of adding the "**wsadmin**" user, adding them to the local group of remote desktop users and administrators.