CM0588 – Malware Reverse Engineering Lab

Paolo Falcarin – April 2025

1. Disabling a malicious trigger in Ghidra.

Download the **malware-example-ghidra.zip** file and extract the following files:

1. malware.c,
2. malware.exe,
3. Obfmalware.c,
4. Obfmalware.exe

Don't worry, this is not real malware just a toy example! 😊

The program has been obfuscated using the following command:

```
tigress --Environment=x86_64:Linux:Gcc:4.6 --Transform=Flatten --Functions=main --Trans-
form=InitOpaque --Functions=main --out=Obfmalware.c ./malware.c
```

You have been given a program `Obf_malware.exe` that has been generated and obfuscated for you. Using NSA's Ghidra tool, find and disable the malicious trigger.

Examining the program:

`Obf_malware.exe` is a number guessing game. Try different numbers to see if you can get the correct number. Once you get the correct number, you may see something like this.

```
liger@liger:~$ ./malware.exe
Guess a number between 0-9.
5
Yes, that's my number, congratulations.
You've activated the malicious trigger!
```

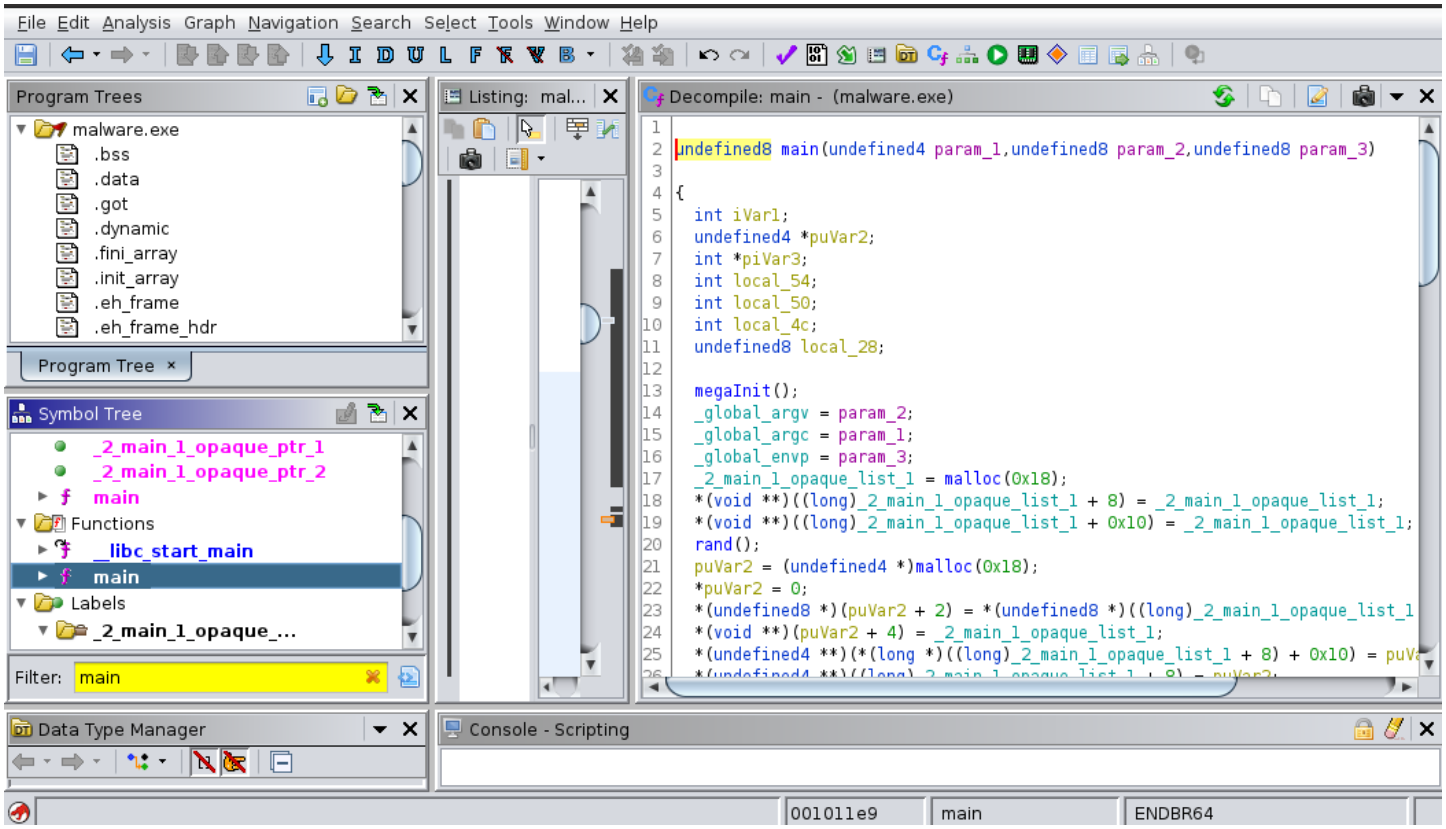
This program activates malicious code once the user guesses the correct number. We want to disable this so that the number guessing game can be played normally, without activating the malicious code.

Analysing the program:

Open ghidra, import *Obf_malware.exe* with default settings, and when asked to perform an analysis, select yes, and analyse with the default settings.

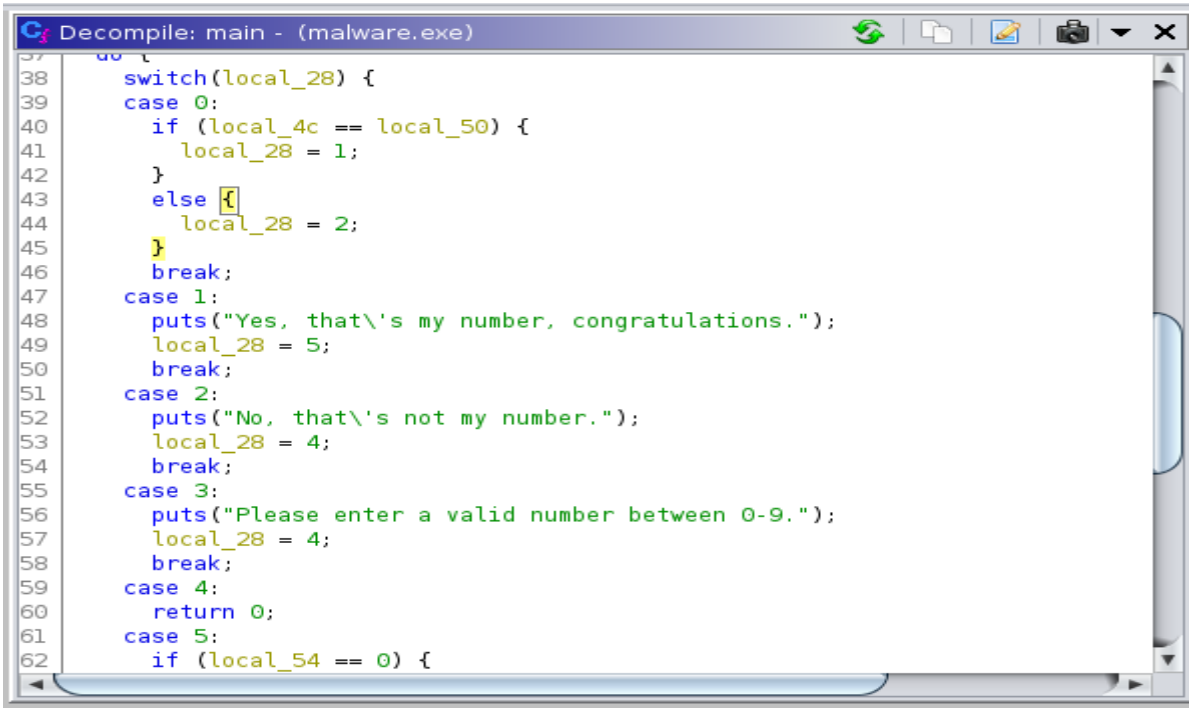
To begin disabling the malicious trigger, let's start by looking in the main function.

In the System Tree window, filter for the function *main*



If you click on the main function, you should see some code in the *Decompile* window.

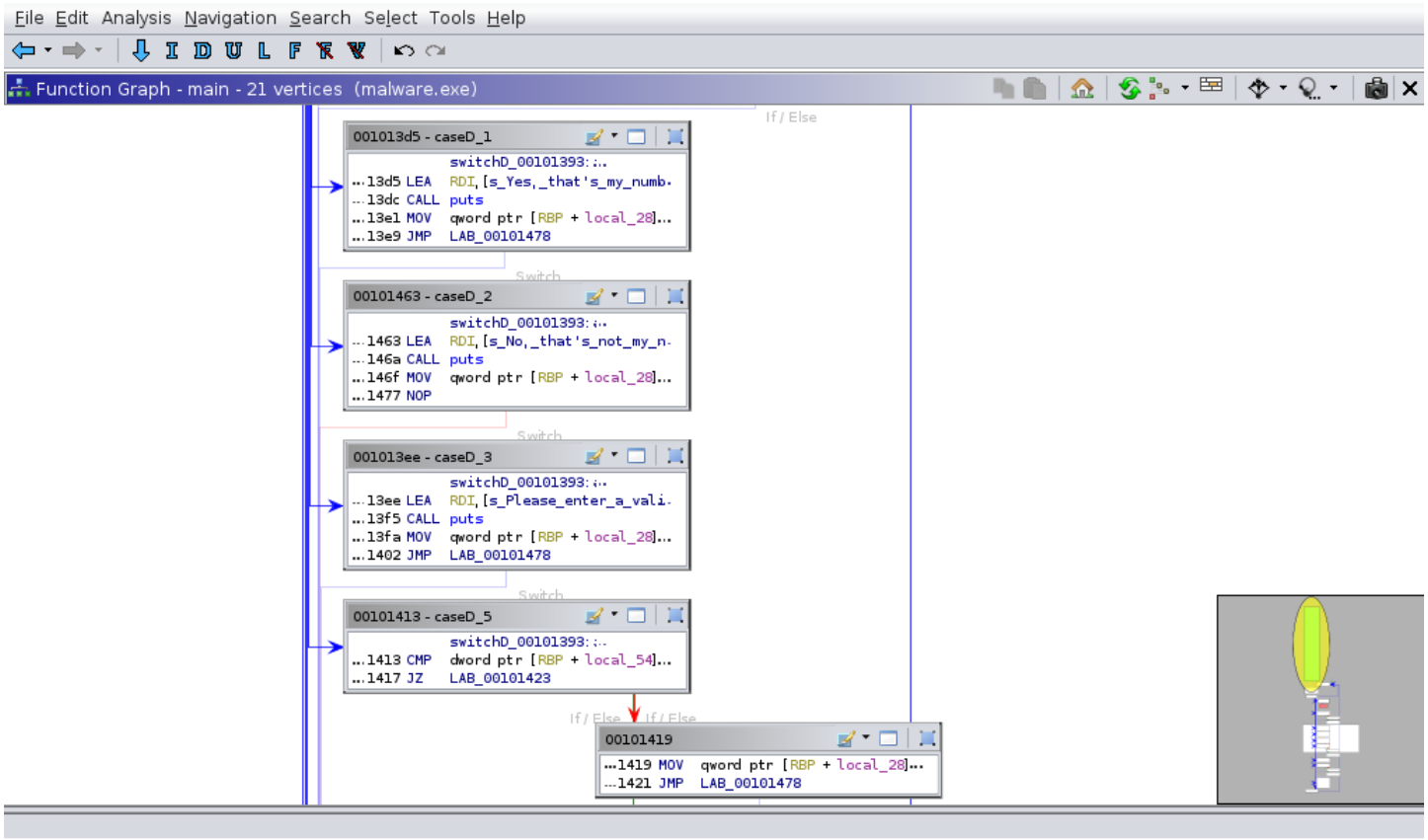
Let's explore the main function a bit more. If you scroll down, you should run into this.



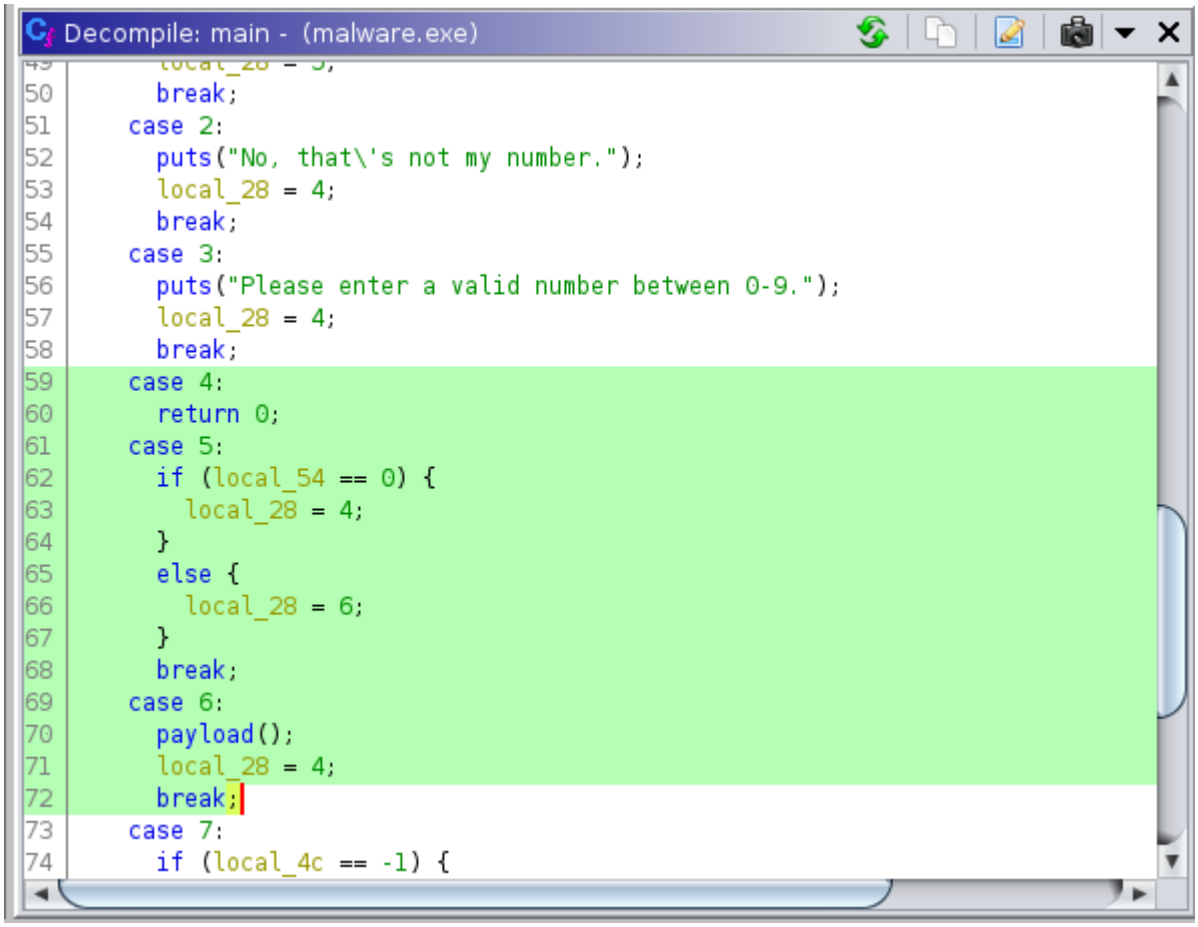
Now we see where the text that prints out the congratulations message appears. We know that the malicious trigger happens after this text.

You may notice a large switch statement, using the variable *local_28*. Let's follow the switch statement to see what happens after the congratulations message.

(If you have trouble following the switch statement, while the main function is selection, you can open the `Function Graph` under the `Window` tab.)



After the program prints the congratulations message, the value of `local_28` changes to 5. Let's see what happens at case 5.



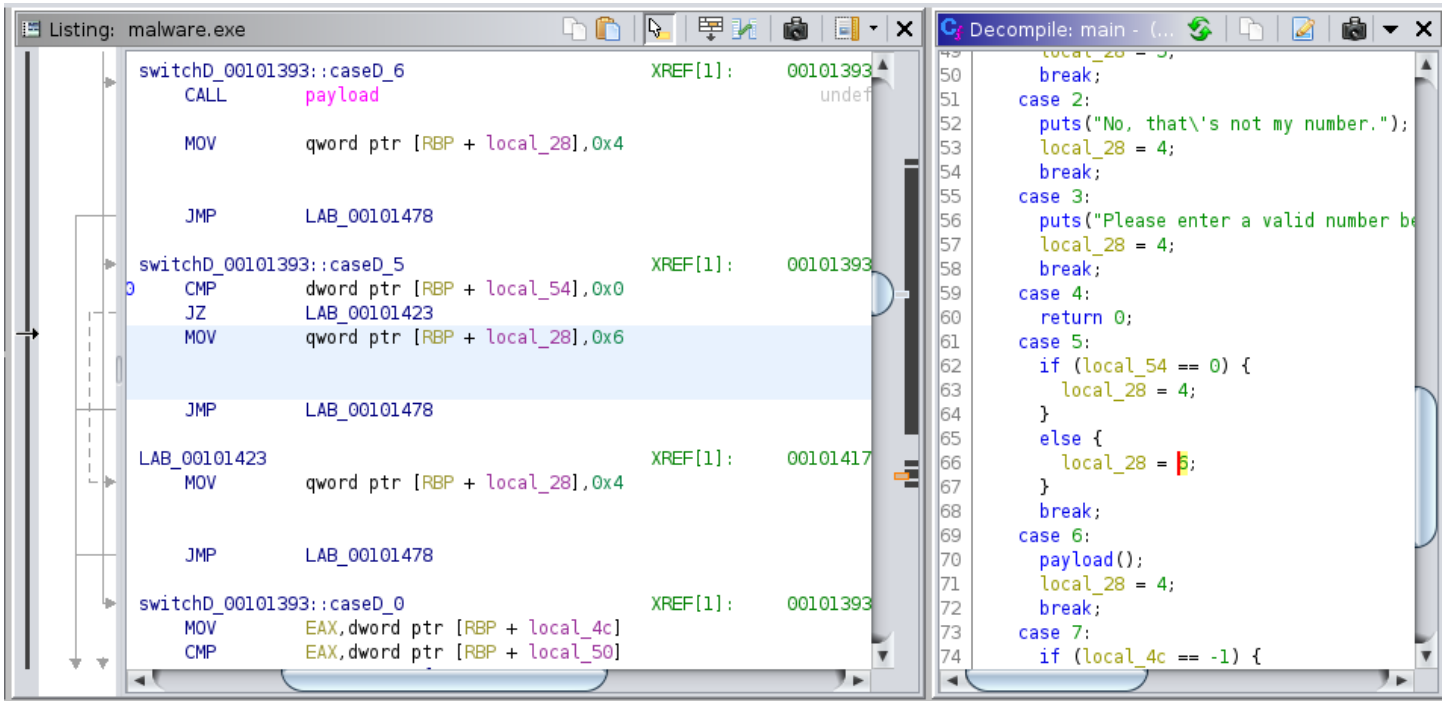
We can see here that after case 5, there are two options. `local_28` changes to 4, or `local_28` changes to 6.

If we look at case 4, it returns 0, ending the program. If we look at case 6, the function activates a payload. This is the malicious trigger that happens.

Patching instructions:

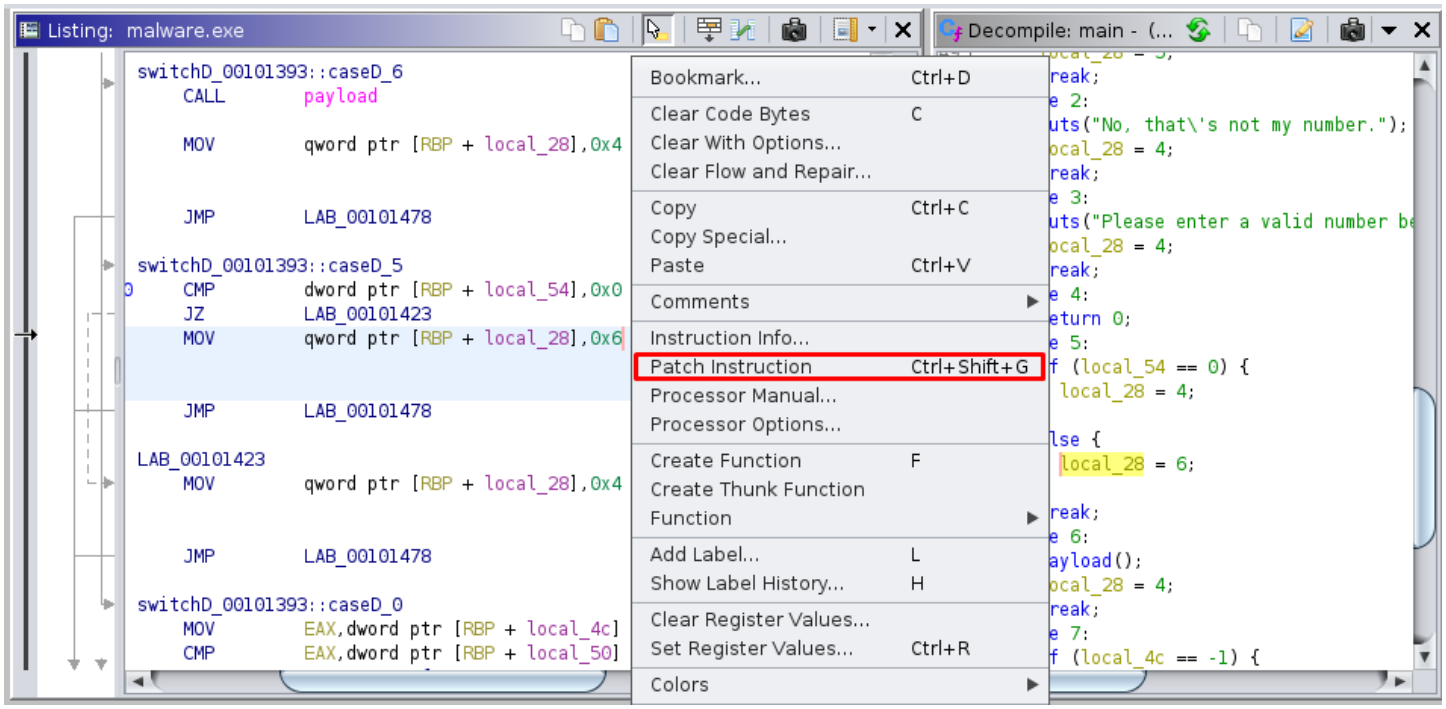
We want to prevent the malicious trigger from occurring. One way to do this is to ensure that case 6 will not occur.

Let's try changing `local_28` = 6 in case 5 to `local_28` = 4.

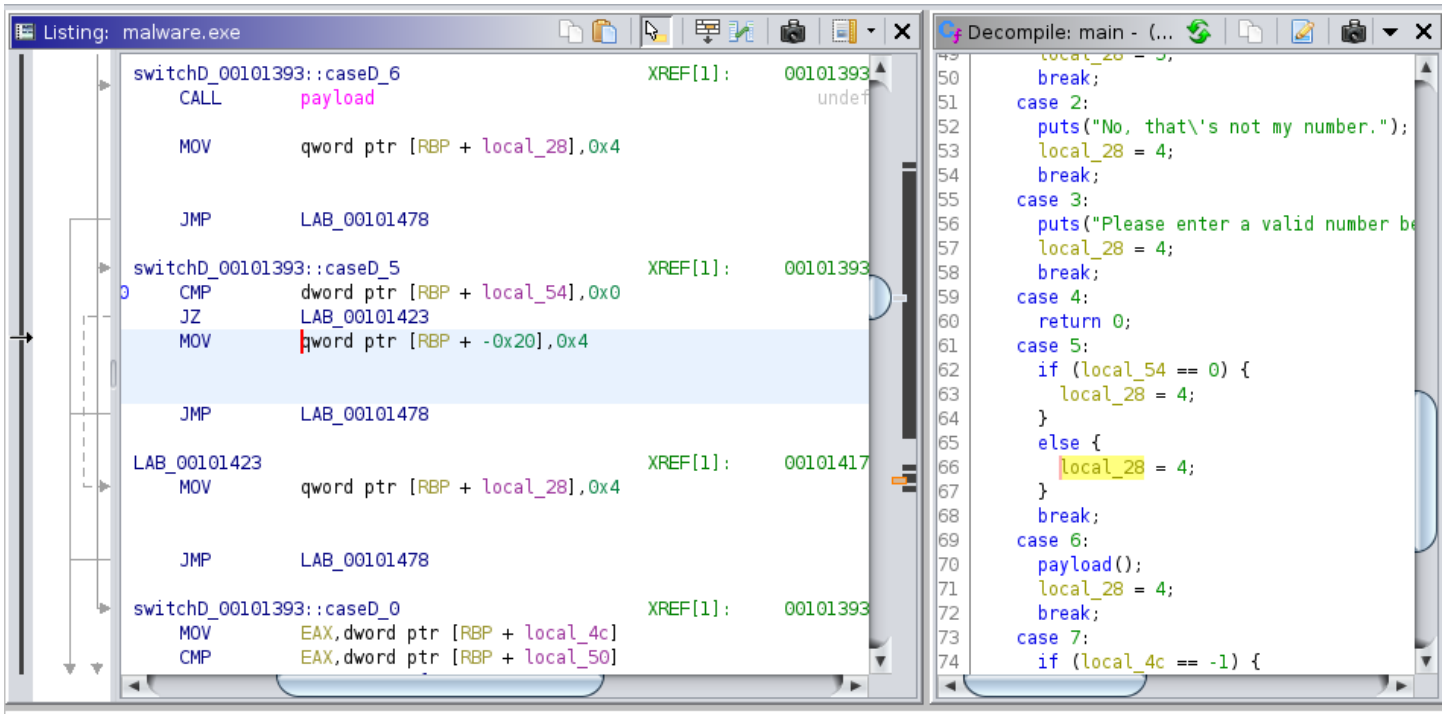


If you click on `local_28` = 6, the Listing window will jump to the assembly instruction that it corresponds to. It appears that the value `0x6` is moved to `local_28` using a MOV instruction.

Let's change the `0x6` into `0x4`. Right click on the assembly instruction, and select `Patch Instruction`.



You should be able to edit the instruction now. Change `0x6` to `0x4`, then press Enter.

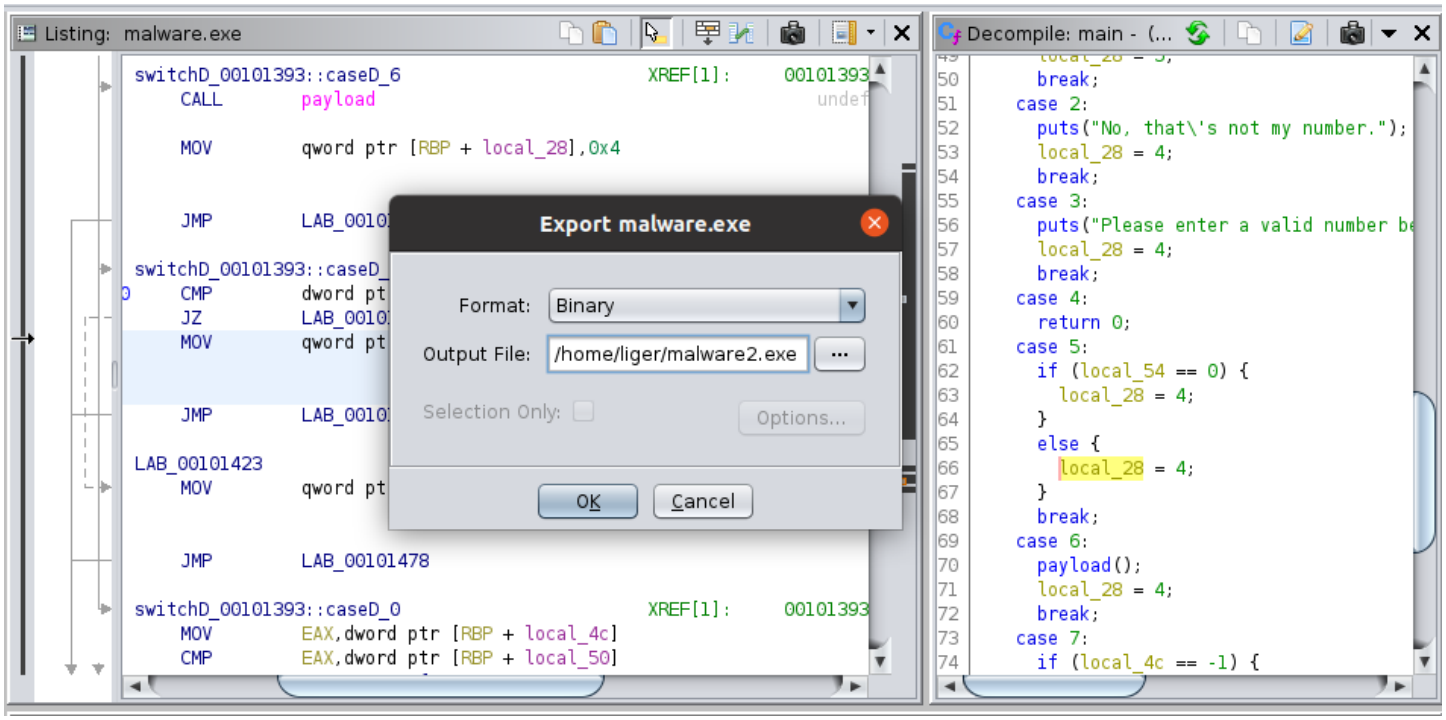


You should see now that the instruction has been changed. In the decompile menu, the code has been changed as well.

Now, when the user correctly guesses the number, instead of jumping to the payload, it should simply end the program as normal.

Exporting as binary:

To export the edited program, go to `File` > `Export Program`, and export as Binary.



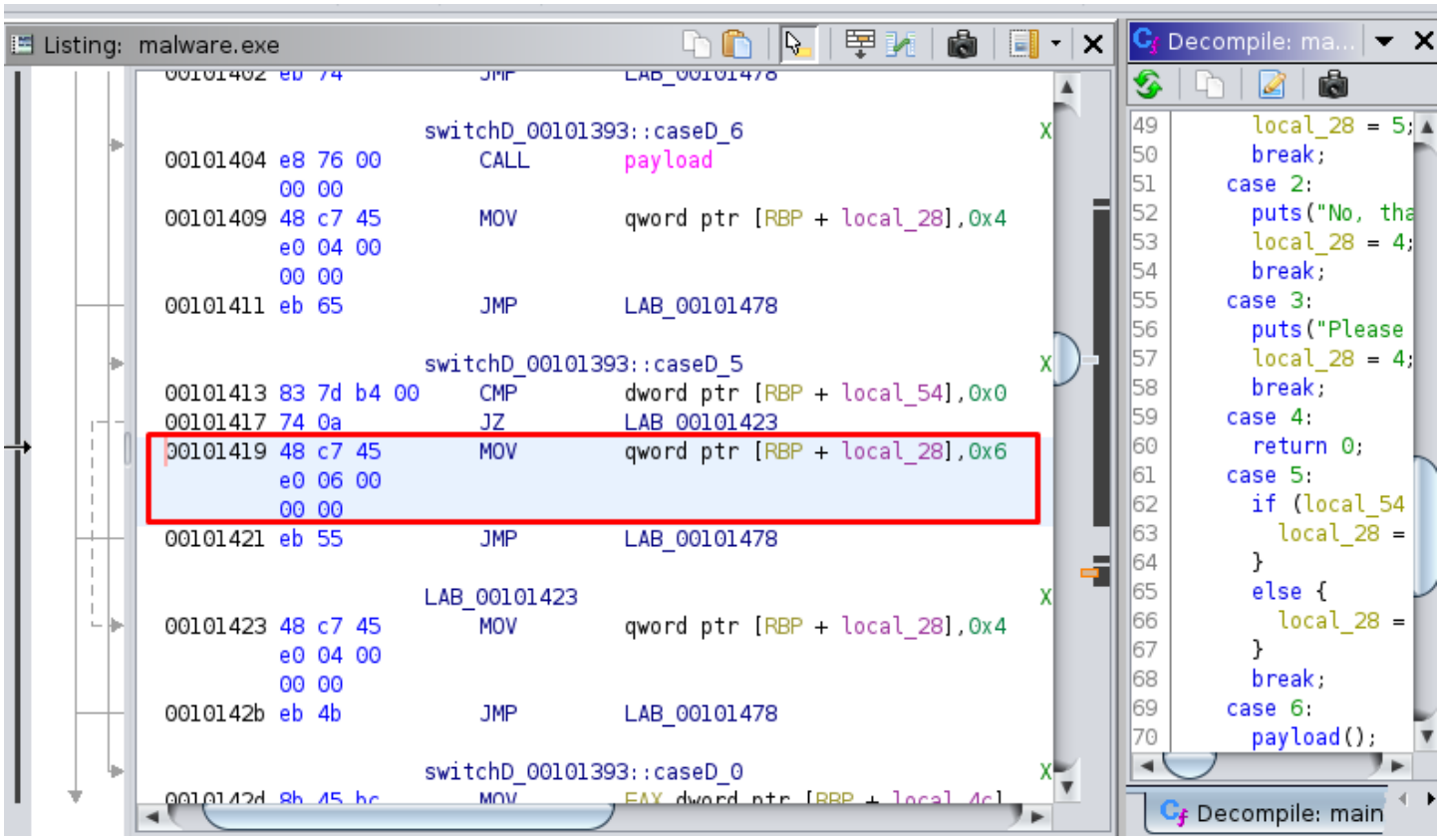
You can rename the output file here. In this case, I have renamed it to malware2.

Now, you can run the program that ghidra just exported. When you guess the correct number this time, it should display the congratulations message, and end the program without triggering the malicious code.

If the program works, you are done! However, you may encounter a seg fault instead. This is due to how the export feature works in ghidra. The next section below outlines a way you can work around this with an alternative way to patch the instruction.

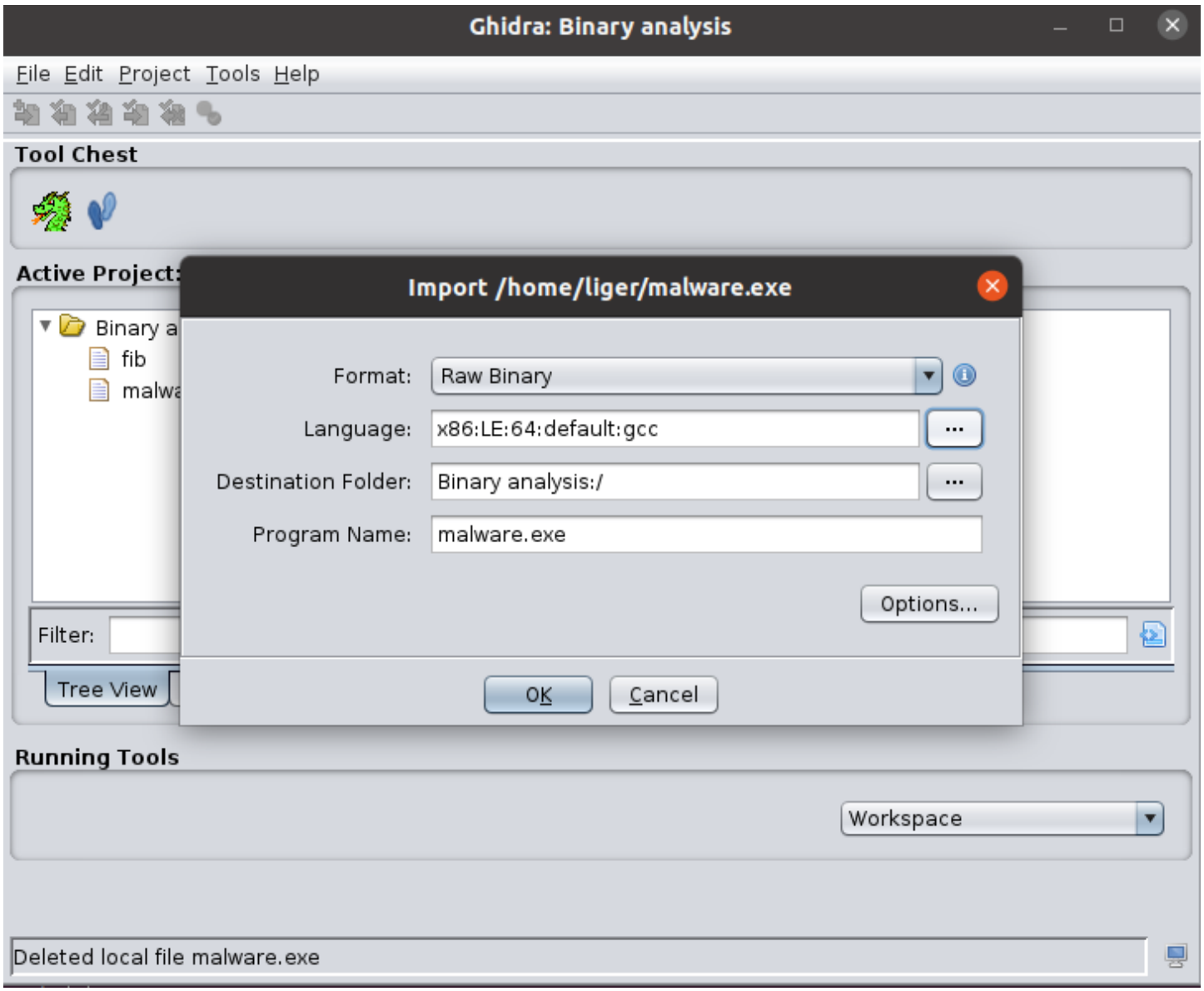
Patching using byte window:

If you run into problems, here's an alternative way to edit a binary in ghidra and export it. Keep note of the location of the instruction you intend to patch.



In this case, it's `48 c7 45 e0 06 00 00 00`. We want to change the `06` into a `04`.

Import `Obf_malware.exe` again, this time, with different import instructions. Select the format `Raw Binary`, and select the same language as before.

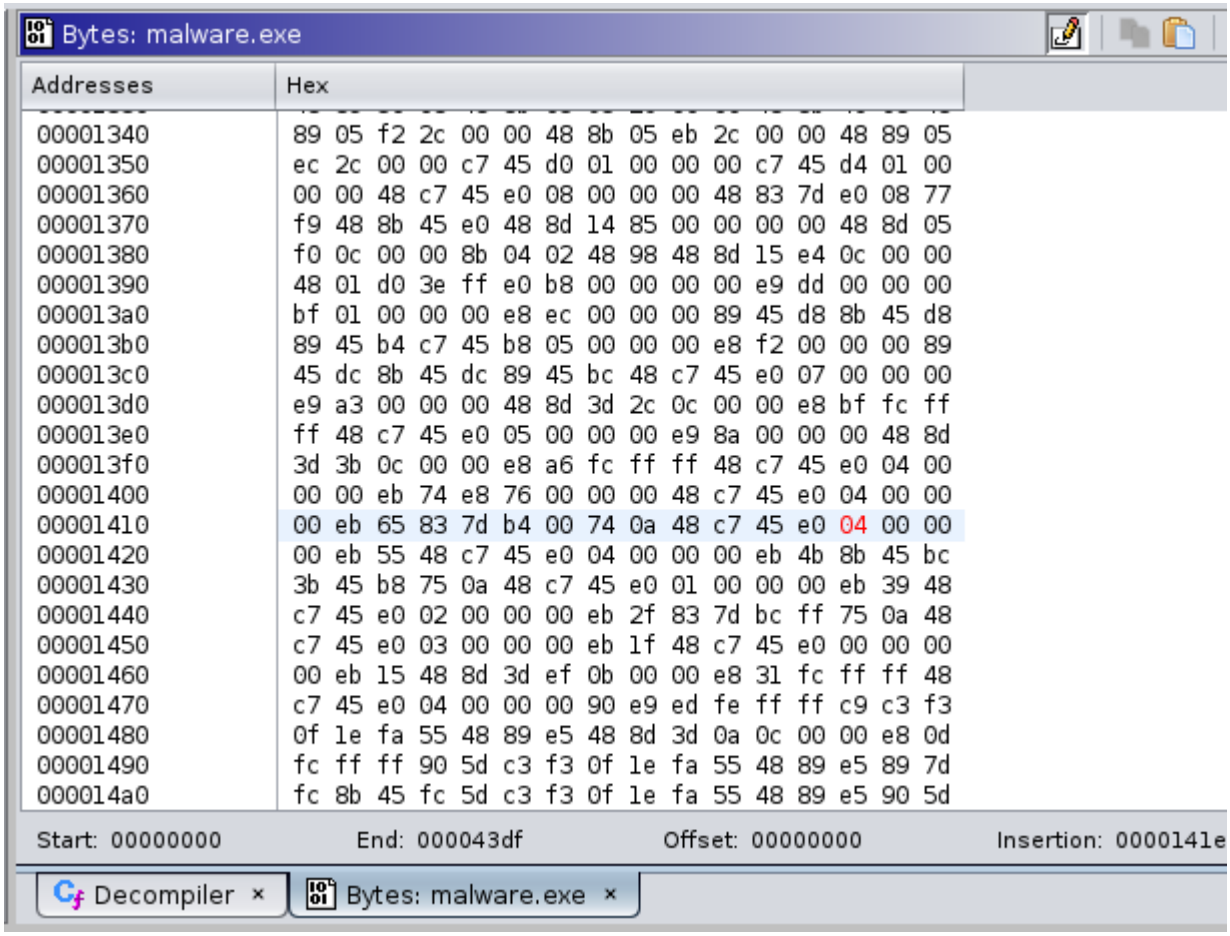


Run the analysis again. You may notice that the analysis this time is a lot less clear.

Press **s** to open up the search window. Select hex, and search for the hex of the position of the instruction you want to patch.

When you find the instruction, click on it and open up the Bytes window under the Window tab. At the top of the window, select the pencil icon, to allow editing. Now you can edit the instruction.

In this case, we change the `06` to `04`, highlighted here in red.



After editing the instruction, you can now export it as binary like before.

Run the exported program and try guessing the correct number.

```
liger@liger:~$ ./malware3.exe.bin
Guess a number between 0-9.
5
Yes, that's my number, congratulations.
liger@liger:~$
```

Now the congratulation message appears, and no malicious trigger is activated. You are done!