Csi

Lab - Reverse Engineering WannaCry Ransomware using Ghidra

**Overview**

**In this lab, we will reverse engineer the WannaCry Ransomware and try and find the killswitch.** **discovered by Marcus Hutchins aka MalwareTech. Marcus reverse Engineered WannaCry and discovered the program checks a certain URL that was not registered and inactive. If the domain remained inactive, the ransomware would install. Once Marcus registered the domain, it shut down the ransomware.**

**May 2017, a ransomware worm quickly spread across Internet targeting computers running Microsoft Windows Operating System by encrypting data and demanding ransom payment in the form of Bitcoin. While the attack was stopped within a few days after the discovery of killswitch, it is reported that almost 200,000 computers were affected across 150 countries and the total damage ranging from millions to billions of Dollars.**

**WannaCry once gets installed on your windows machine, it encrypts the file on PC’s hard drive, making them almost impossible for users to access, it then demands a ransom to be paid using bitcoins.**

**Reverse engineering is a crucial process for malware threat intelligence researchers working backward from malware being used to carry out attacks — to understand how it works, what its capabilities are, and who wrote it or where it came from.**

**Reverse engineering is also an important way for defenders to check their own code for weaknesses and confirm that it works as intended.**

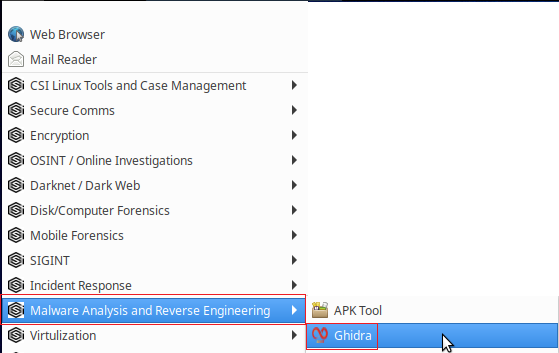
**Lab Requirements**

* One installation of VirtualBox with the extension pack.
* One virtual install of the latest version of CSI Linux.
* VirtualBox network adapter set to NAT network.

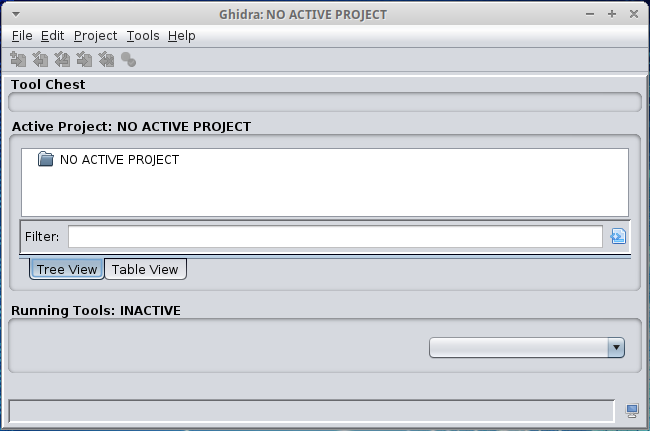
**Launch Ghidra**

Installing Ghidra onto CSI Linux was covered in a previous [lab](https://www.dropbox.com/s/k23vl9zleukjrmc/Lab%20-%20Introduction%20to%20Using%20Ghidra.pdf?dl=0). This lab assumes you have completed this step.

From the application menu in CSI Linux, scroll down until you come to Malware Analysis and Reverse Engineering. Then, expand the context menu, and from the option, click Ghidra to start the application.

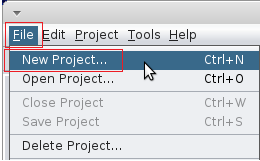


After a short pause, the Active Project screen opens.

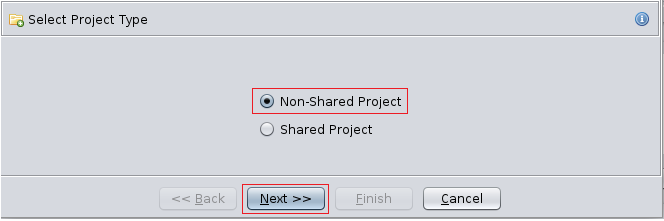


**Creating a new project**

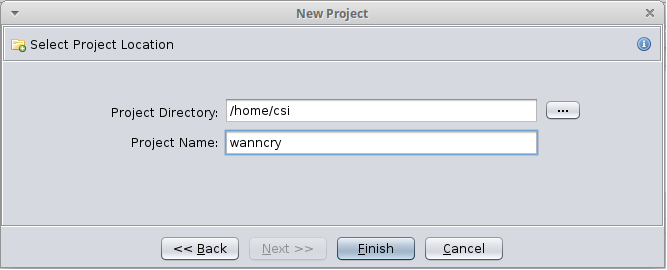
We next need to create a new project. From the Ghidra taskbar, go to File>New Project.



The **Select** **Project Type** screen opens. Ghidra can be used as a collaboration tool or as a stand-alone, non-shared project. We will be working individually on a project, so click "Non-Shared Project." Then Click Next.



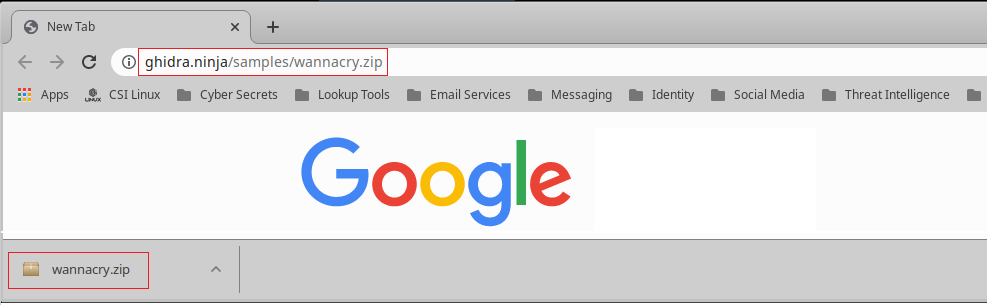
On the next screen, we give our new project a user-friendly name and select a location to save our work. In this example, I have named project, **wanncry.** I will accept the default location for saving my work. You are free to name your project as you see fit and save it where you want. Click Finish.



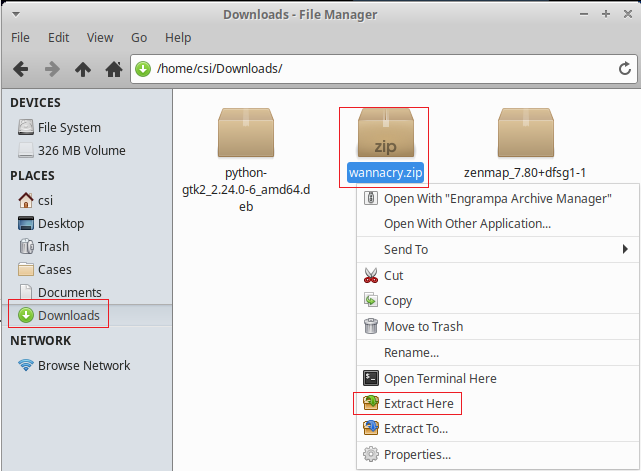
We next need to download an example of wannacry to analyze.

Open a browser in CSI Linux and in the address bar, paste the following URL into your browsers address bar to download the example.

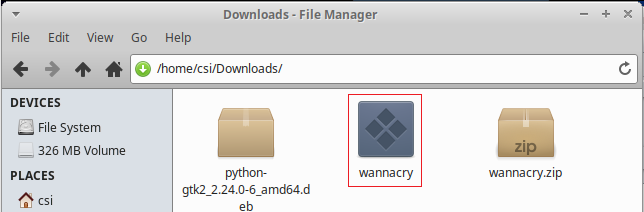
<https://www.ghidra.ninja/samples/wannacry.zip>



Close out your browser. From your desktop, open you Home folder. From the left-hand windowpane, open your Downloads directory. Find your wannacry.zip archive. Right click on the archive and from the context menu select, Extract Here.

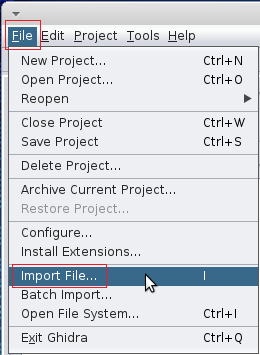


The archive will need the following password. The ZIP’s password is **ghidra.ninja**

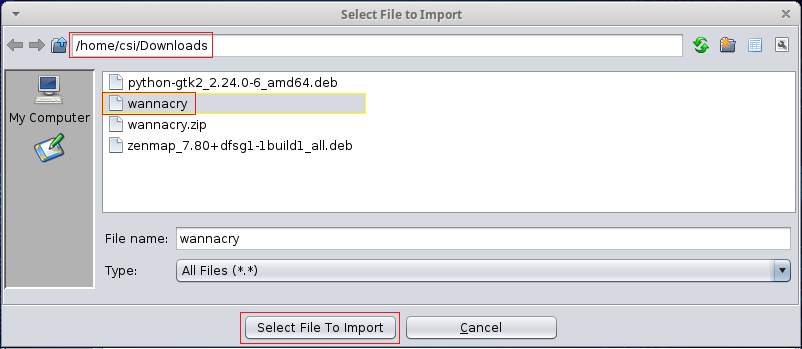


**Import the wannacry file into Ghidra**

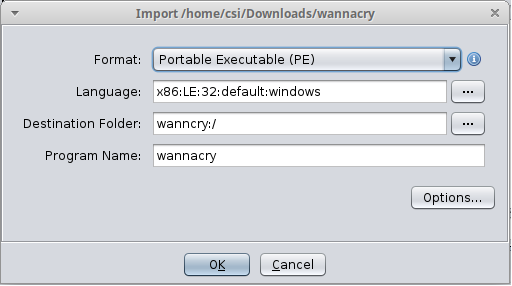
From Ghidra’s Active project screen, click on File> Import File.



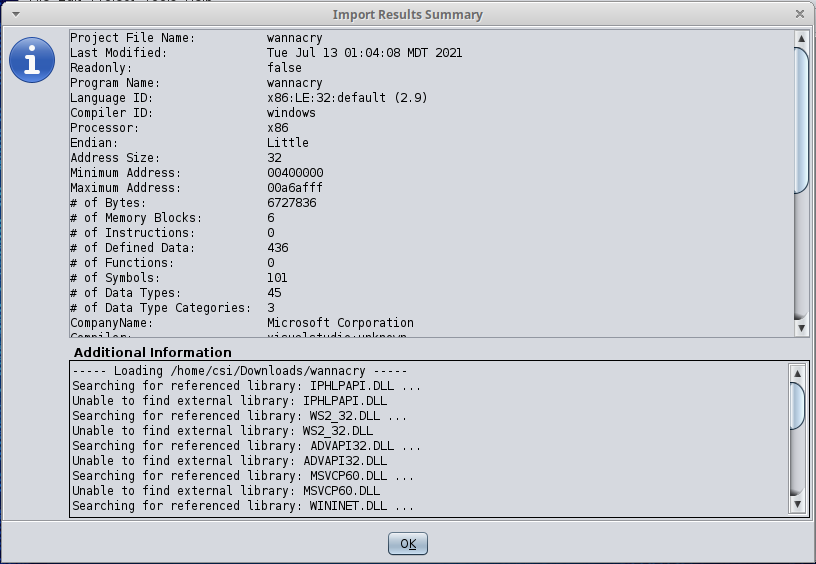
Browse over to your Downloads directory. Find the extracted wannacry file. Highlight the file and press the Select File to Import button.



Once the importation process begins, you are presented with information about the file. Press OK.

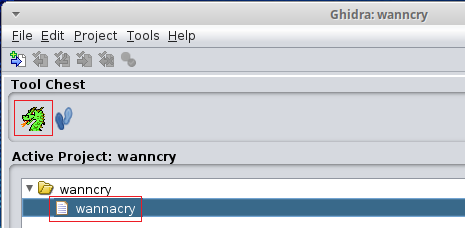


Once the importation process completes, you are given the import summary. Press OK.



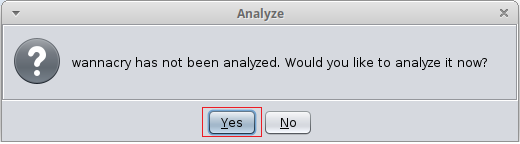
**Begin the Analysis**

In the Active Project window, highlight the wanncry file and click the green dragon icon to begin the analysis of the file.

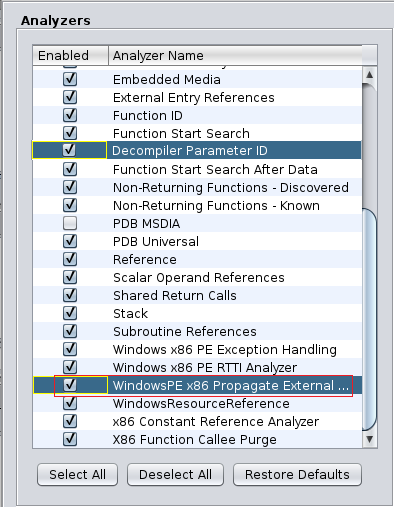


If the code browser shows no content, close the browser, and start the analysis one more time.

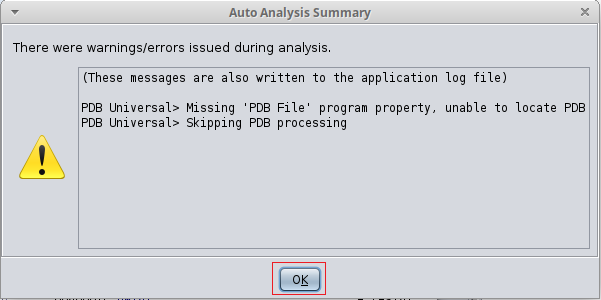
Once the analysis begins you are asked if you would like to analyze now. Click the Yes button.



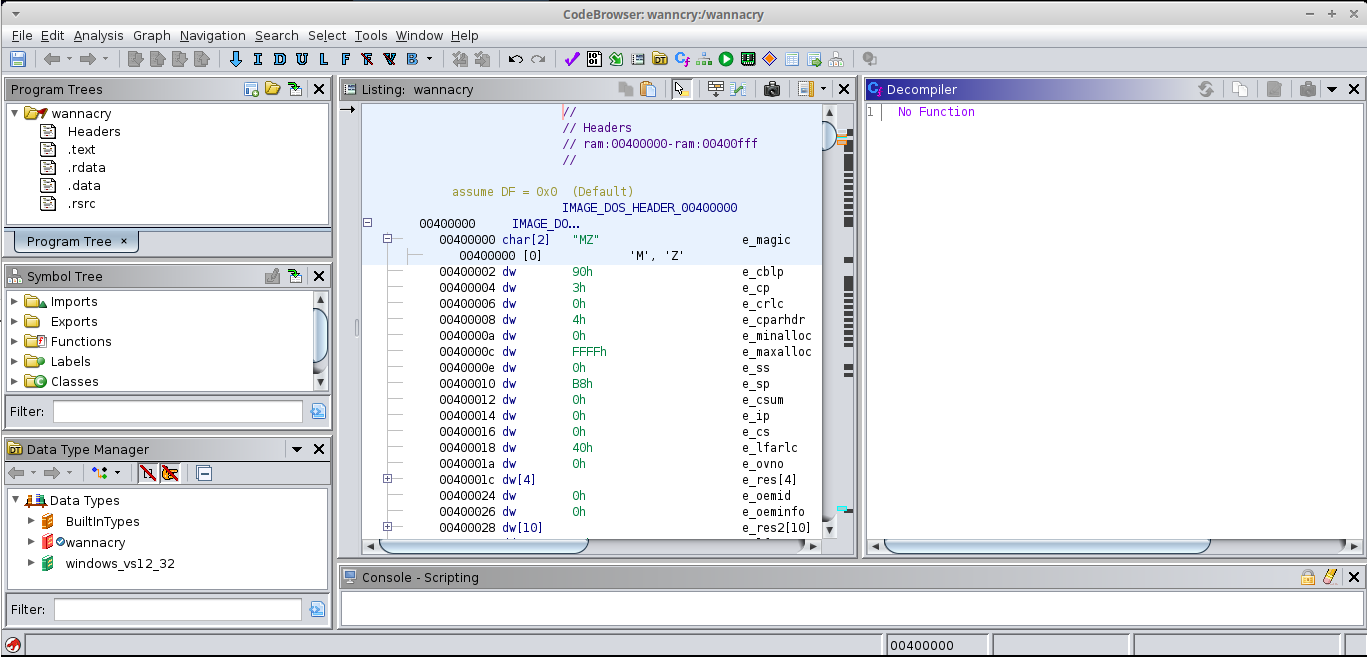
On the next screem you can select what analyzers to use. Add the Decompiler Parameter ID and WindowsPE x86 Propagate External Parameter option as additional analyzers. Press Analyze.



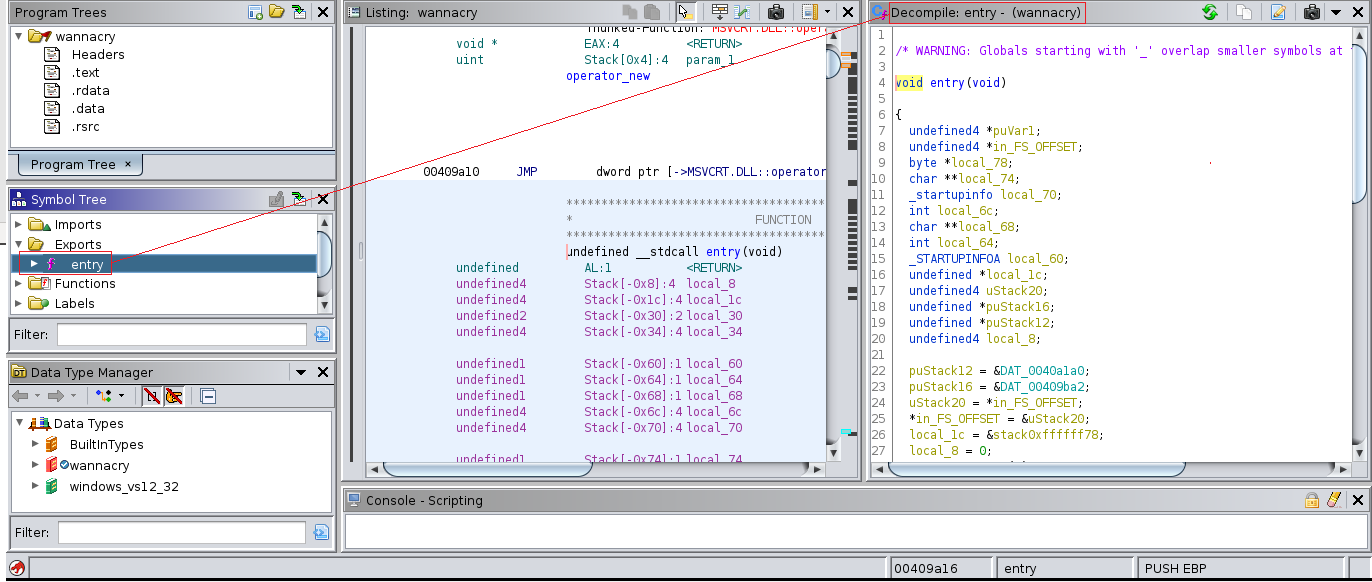
Ignore any warning.



Once the analysis has completed, you are presented with the compiled code broken out into the four different windows.



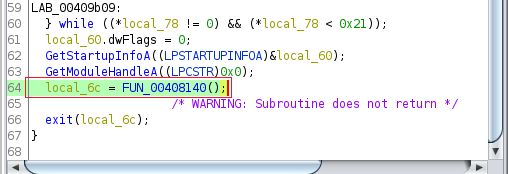
We need to find either a main() function or a WinMain() function. We see neither but inside the Symbol Tree window, under Exports, we find a function called **entry**. When we click the **entry** function, it is decompiled and the code for the function is shown to us in the Decompile window.



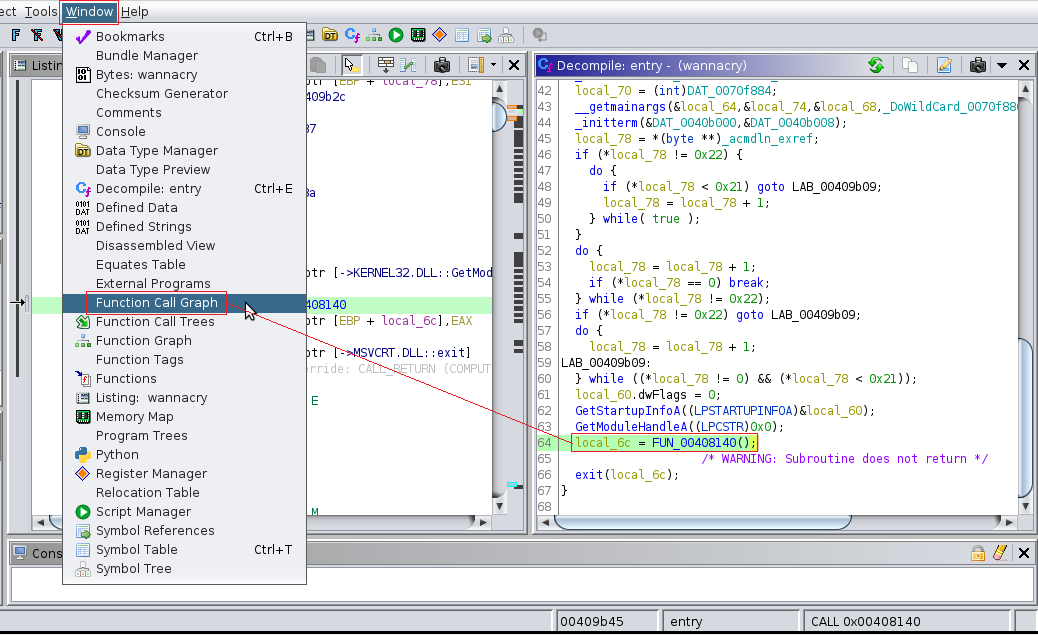
**What is Winmain?**

WinMain() is the C entry point function of any windows application. DOS/console-based applications use main() as the C programming entry point function however windows Win32 graphical applications use WinMain() as the entry point function.

In the Decomplier windows, we scroll to the bottom of the code and we a call to another function.

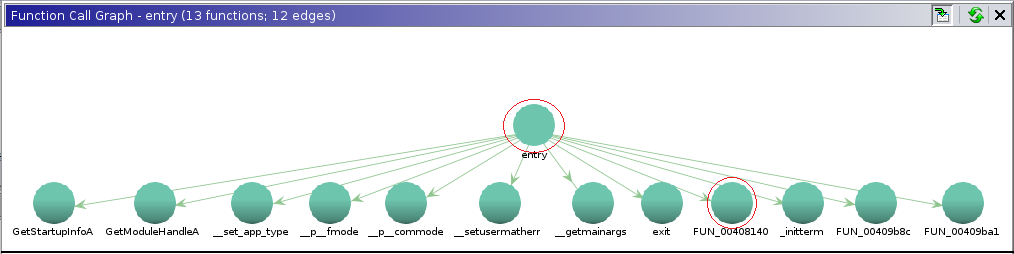


Highlight the function and from the Ghidra taskbar, click the windows tab and scroll down unto you find Function Call Graph.

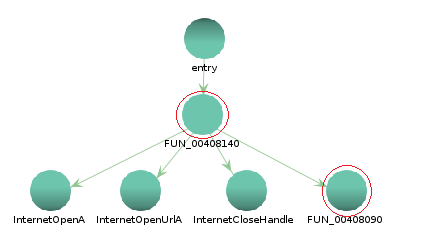


Once we open the Function Graph Call, we can see what the call from the entry function looks like.

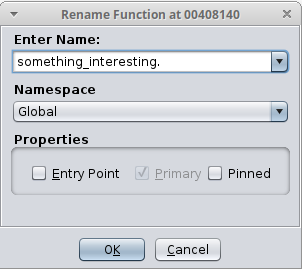
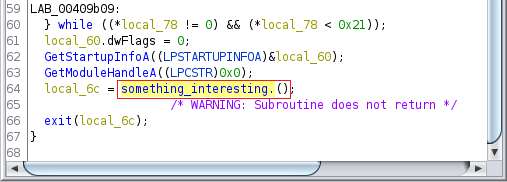
The Function Graph Call tool is a great way to see what function are being called on by another function. Our function of interest is **FUN\_00408140.**



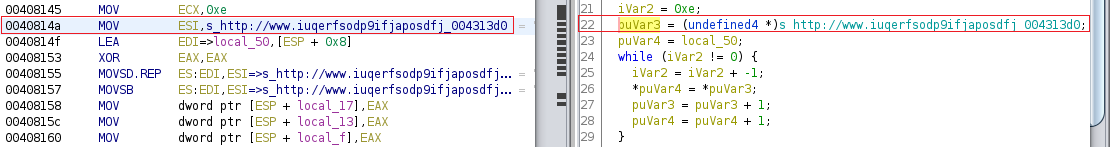
In the Graph window, if we x2 click the FUN\_00408140 function, we can see what it is calling on.



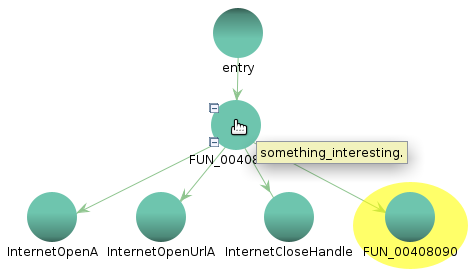
Minimize the Function Graph Call window. From within the Decompile window, right click on the FUN\_00408140 function and rename it to **something\_interesting**. Click OK.



X2 Click the function, **something\_interesting**. Notice the strange string on line 22 calling on a URL.



“s\_http://www.iuqerfsodp9ifjaposdfj\_004313d0”



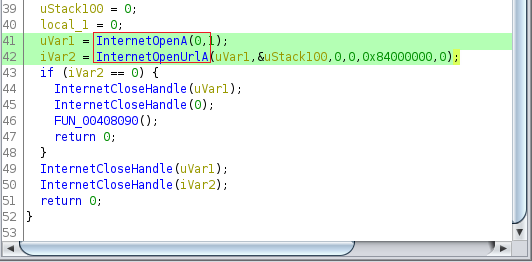
We can see that the function, something\_interesting is calling on the two functions, **nternetOpenA()** and **InternetOpenUrlA().**

**InternetOpenA() function** is used to initialize the application’s use of the WinINet functions.

The **InternetOpenUrlA() function** is used for opening a resource specified by a complete FTP or HTTP URL.

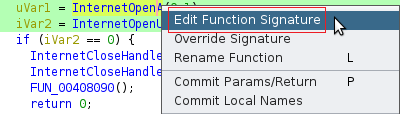
Read the following Microsoft documentation to learn more about these two functions.You will need the code form the pages for the functions **[InternetOpenA()](https://docs.microsoft.com/en-us/windows/desktop/api/wininet/nf-wininet-internetopena)** and [**InternetOpenUrlA()**](https://docs.microsoft.com/en-us/windows/desktop/api/wininet/nf-wininet-internetopenurla)**.**

In the Decompile window, scroll to the bottom and find the functions, InternetOpenA() and InternetOpenUrlA().

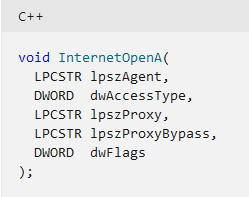


**Change the function signature for both functions.**

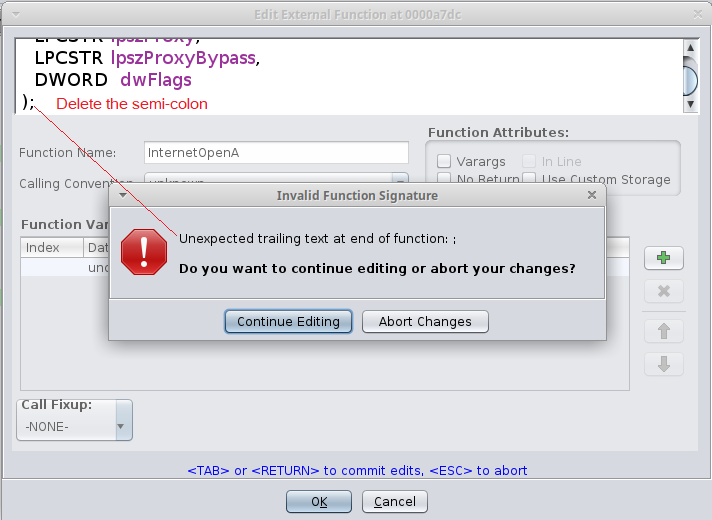
Starting with the InternetOpenA() function, we right click on the function and from the context menu we select, Edit Function Signature.



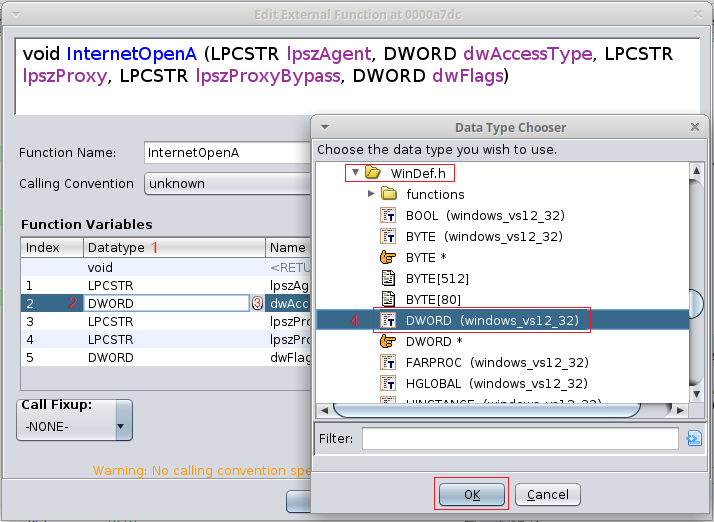
From the [Microsoft site](https://docs.microsoft.com/en-us/windows/win32/api/wininet/nf-wininet-internetopena), we copy and paste the following data into the text box.

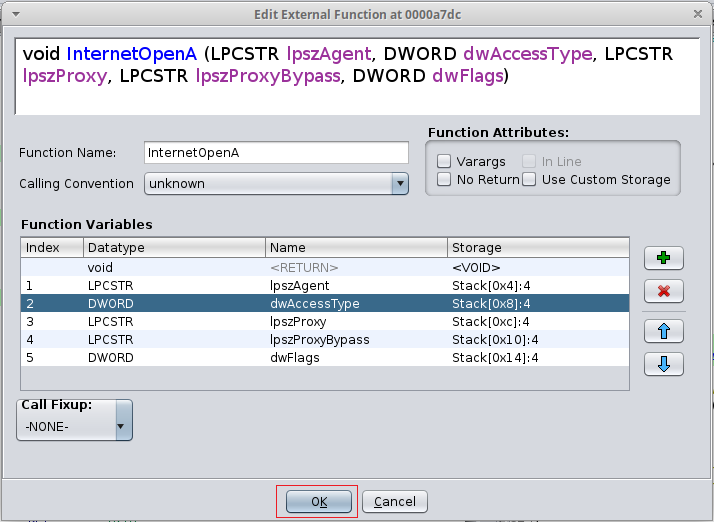


At the end of the text box is a semi-colon that must be removed.



In the Function Variables, click on Datatype (1). From the options, click on DWORD (2). Over to the right of DWORD you will see a small grey browse button (3). Click on the button and scroll to the bottom until you come see the book icon labeled wanncry. Scroll down the list until you come to the folder labeled **Windef.h**, Select the DWORD entitled, **windows\_vs12\_32** (4). Click OK. Back at the edit window, click OK one more time.



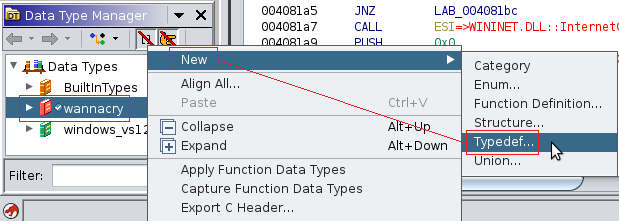


The **InternetOpenA()** has changed to the following.

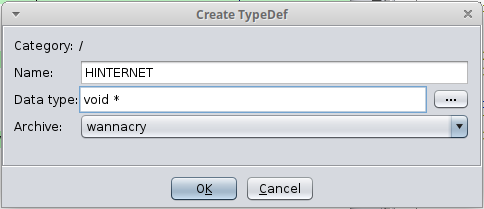


We next change the function signature for **InternetOpenUrlA(). Before we edit the signature, we first need to create a new typedef called HINTERNET**

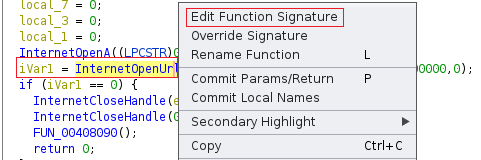
**From the Data Type Manager window, right click on the book labeled, wannacry and from the context menu, select New>**



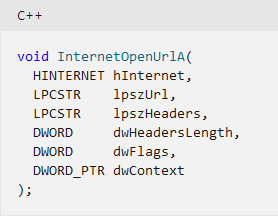
Type the following information in the text boxes. Click OK.



We can now change the function signature for InternetOpenUrlA(). From the Decomple window, right click on the **InternetOpenUrlA() function and form the context menu select, Edit Function Signature.**



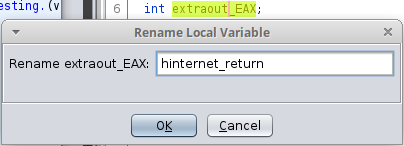
From the [Microsoft site](https://docs.microsoft.com/en-us/windows/win32/api/wininet/nf-wininet-internetopenurla), copy and pastes the following data into the text box. Remember to Remove the semi-colon at the end of the paste.



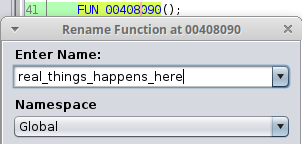
If asked to choose a Datatype, accept the default, and click OK. When you are finished, the edited function should look like this.



From the decompiler window, let us rename some more functions. Rename the function on Line 6, **extraout\_EAX** to read **hinternet\_return**.



Rename the function at line 41 FUN\_00408090, to read **real\_things\_happens\_here.**

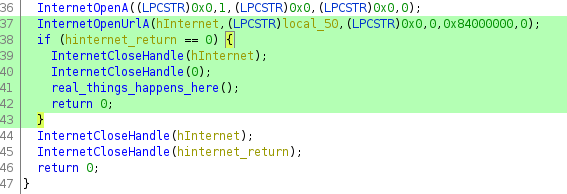


Looking at line 37 makes a request to the strange URL we saw earlier. Line number 38 tells us that, if the request fails, it returns the null handle and then it closes the handle and calls the function, **real\_things\_happens\_here().**

**If the URL request is successful, it closes the handle and quits the program. (Line 44–46)**

**Note**

**Ghidra will not analyze the same file the same way twice. If you close and restart the analysis, the Decompiler windows will show you a different analysis then before. This is Ghidra’s best estimation of what the original code would look like. The code is there, just on a different line the Decompiler window.**



This the famous killswitch discovered by Marcus Hutchins aka MalwareTech. He reverse Engineered the WannaCry and discovered that the ransomware checks for the above URL. The URL had never been registered and inactive. Once the ransom wear checked the URL and found it was active, the ransomware was shutdown.