Malware Reverse Engineering Documentation

Eli Ledford, David, Calvin Vang, Adrian

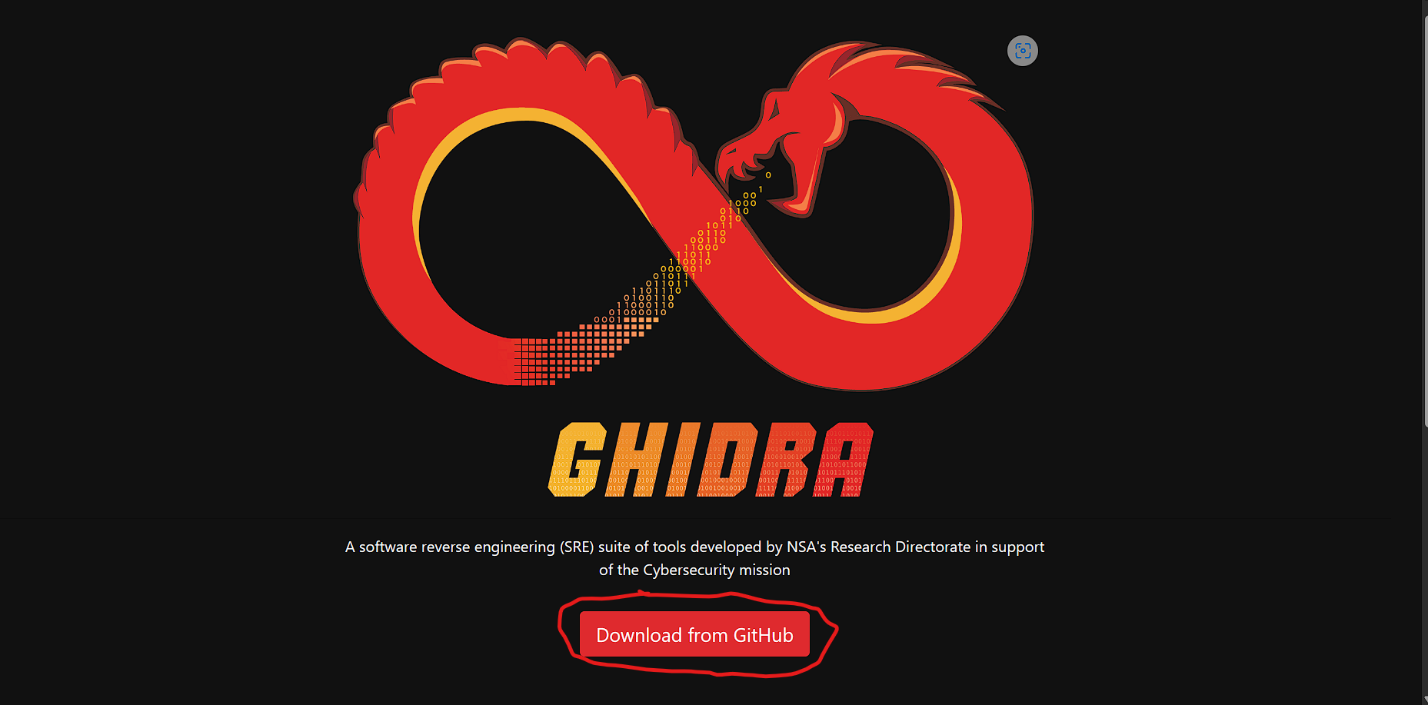
Ghidra Installation for Windows:

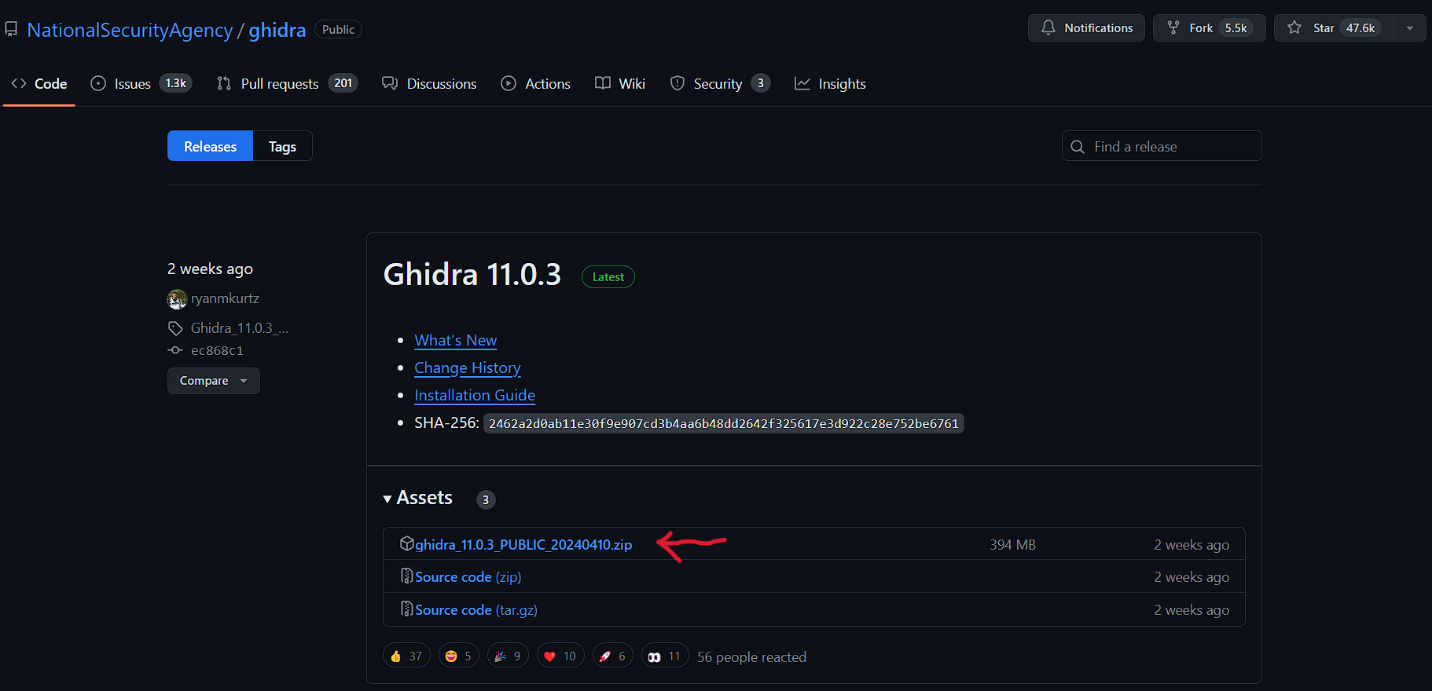
To install Ghidra, simply extract the Ghidra distribution file to the desired filesystem destination using any unzip program (built-in OS utilities, 7-Zip, WinZip, WinRAR, etc)

Downloading from Github, the Ghidra

Assets

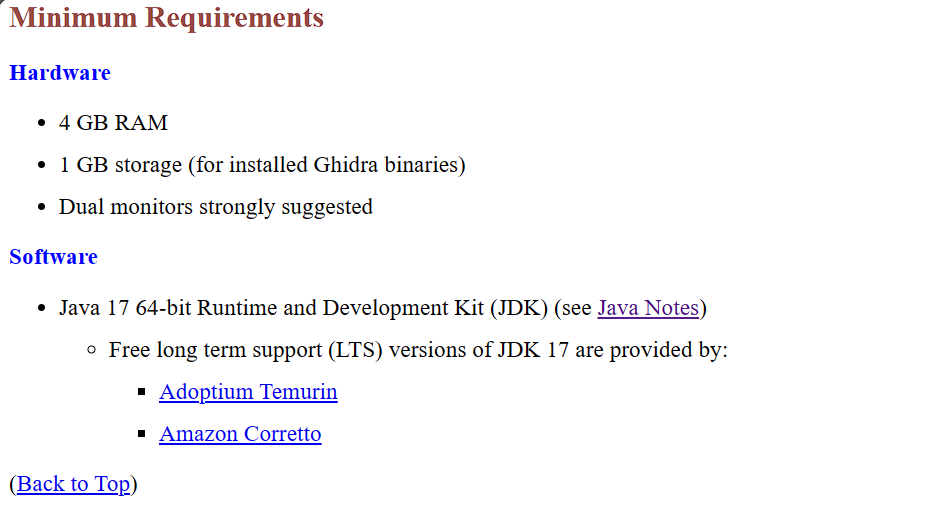
ghidra\_11.0.1\_PUBLIC\_20240130.zip (Current version)

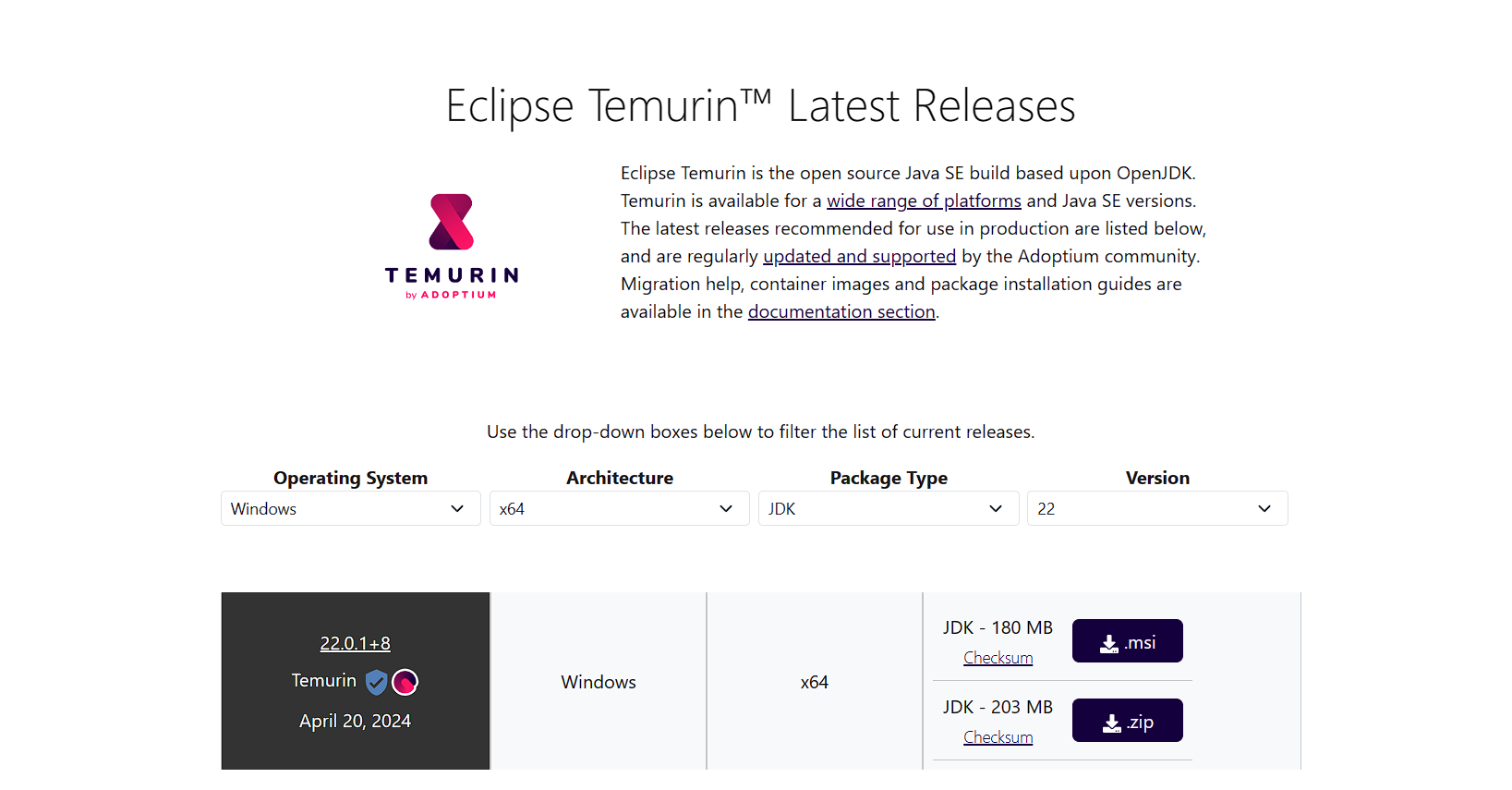




Ontop of that, you need to install a Java 11 64-bit Runtime and Development Kit (JDK)

A Free long term support (LTS) versions of JDK 11 is provided by





AdoptOpenJDK

Based on current systems (mine)

Operating System - Windows

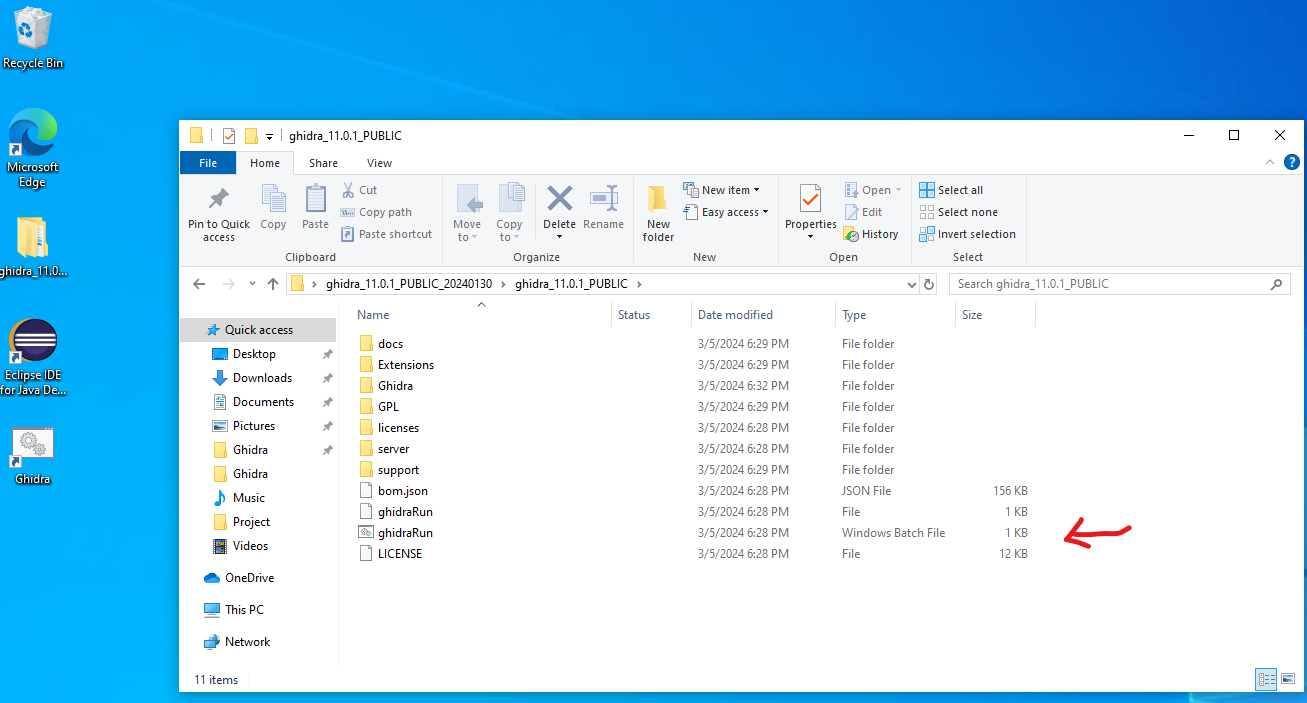
Architecture - x64

Package Type - JDK

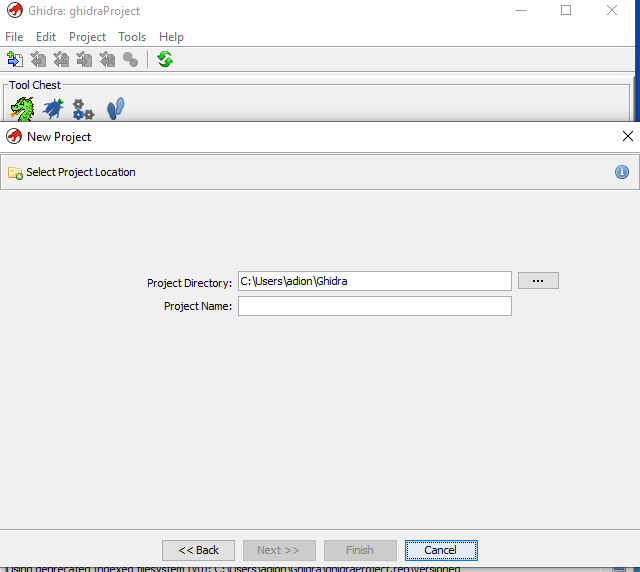
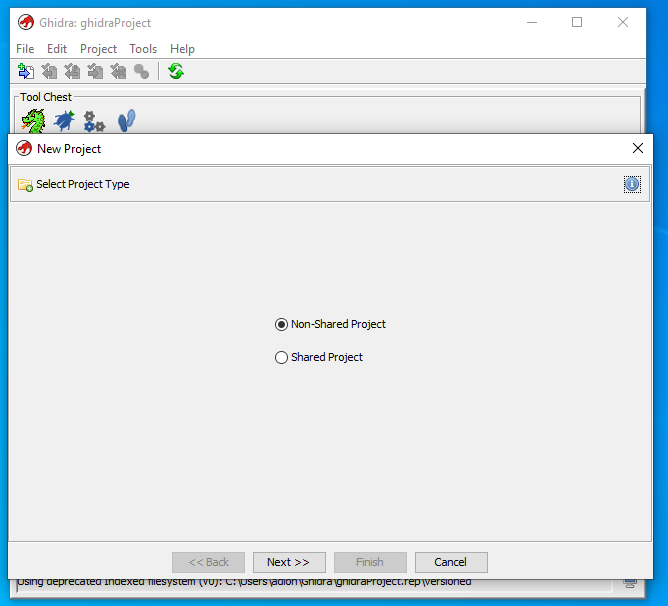
Version - 21 - LTS

The version to download is JDK - 179MB (MSI) or other current version.

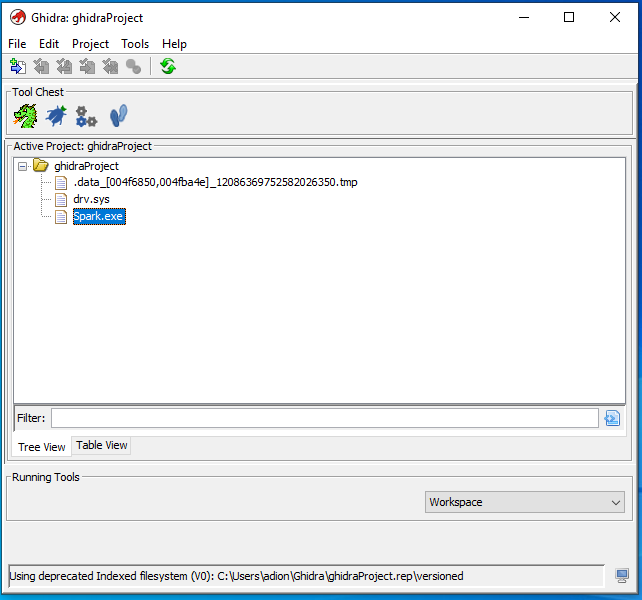
Starting Ghidra, after successfully unzipping and downloading the JDK. Run the ghidraRun or Windows Batch File



To create a new project, click File, New Project, and Select Project Type (Non-Shared Project or Shared Project), and Select Project Location



Or in our case import files



Ghidra Malware Analysis

Task:

Analyze Point of Sale (PoS) malware, where this malware scrapes the RAM memory of PoS systems to steal credit card and debit card information

To start, setup a safe analysis environment, and look for malware indicators in the sample.

* VirtualBox
* VirtualTotal, malware analysis tool
* Code found in: <https://github.com/PacktPublishing/Ghidra-Software-Reverse-Engineering-for-Beginners/tree/master/Chapter05>

Code in action:

The malware consists of:

Windows driver

Portable Executable

A screenshot of a computer

Description automatically generated

What we found in the file:

The user for this computer is named Benson, who seems to have complied with this malware. Based on these strings, multiple attributes of the malware could be found based on these results.

Why is there a reference to windefender.exe? On top of that, the address c:\\drivers\\test\\objchk\_win7\_x86\\i38\\ssdthook.pdb, where the SDDT and the SHELLCODE\_MUTEX are hooks that can be malicious.

SDDT - data structure in Microsoft Windows operating system that maps system services numbers to addresses of matching system service routines.

Reviewing a program's strings can be a good start to reveal whether the cause of the problem is a malware issue without further analysis.

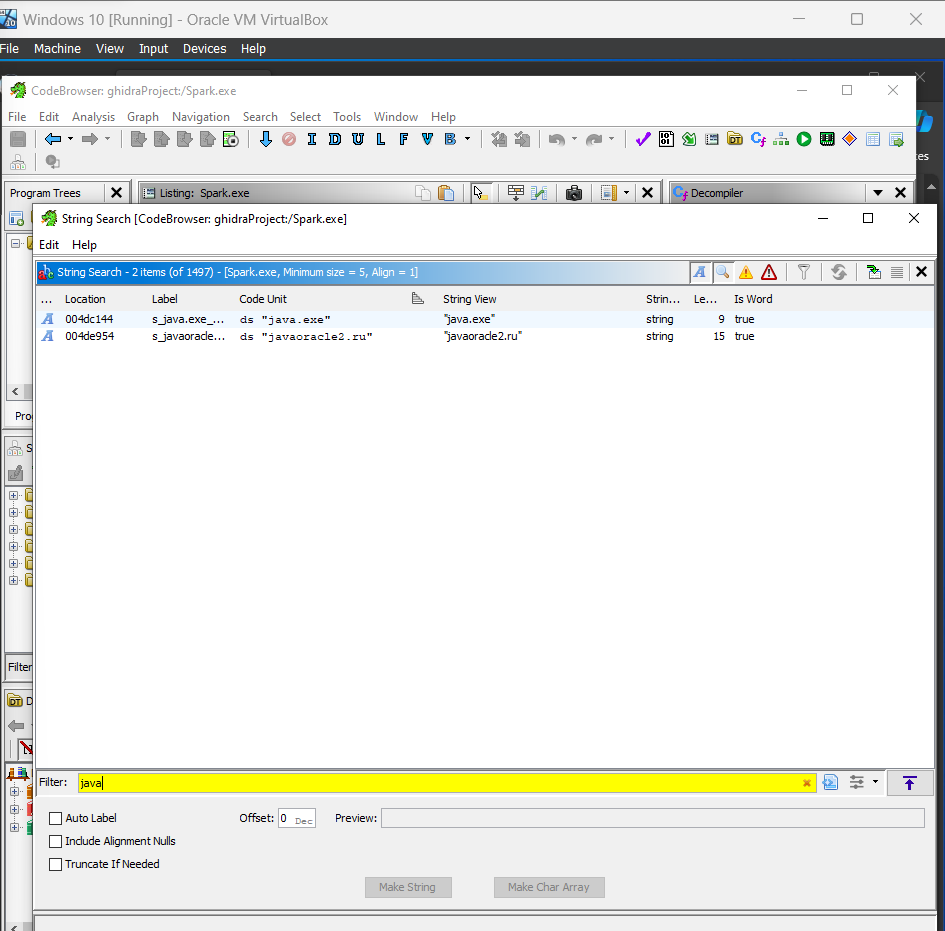
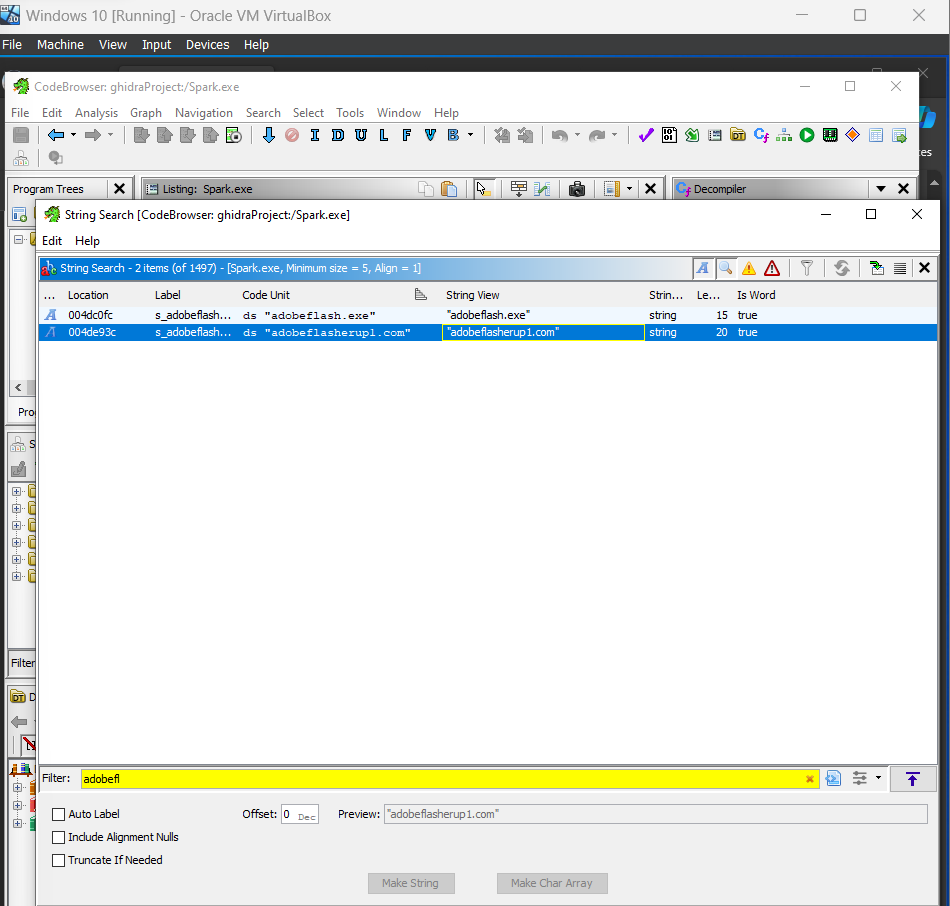
Further investigation of strings

Domain names may help reveal further if there seems to be a potential danger such as malware.

To analyze it, we use the following website [VirusTotal - Home](https://www.virustotal.com/gui/home/url) , to check if there are dangers within the domain name.

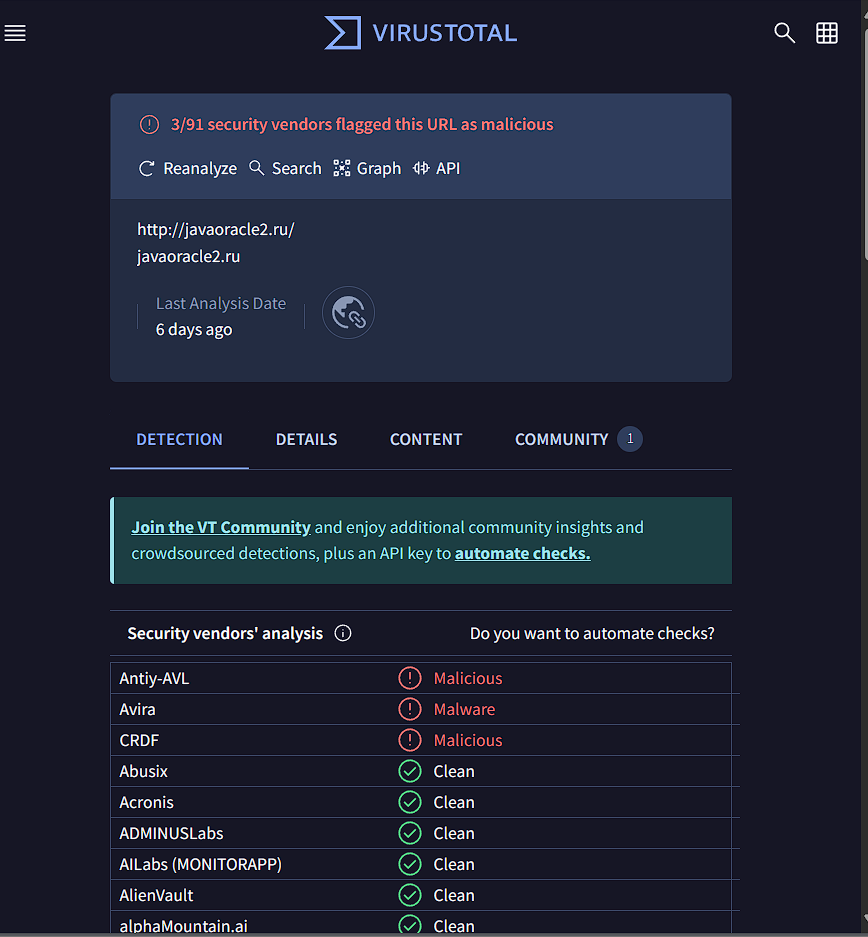
I found these domains within the program:

* adobeflasherup1.com
* javaoracle2.ru



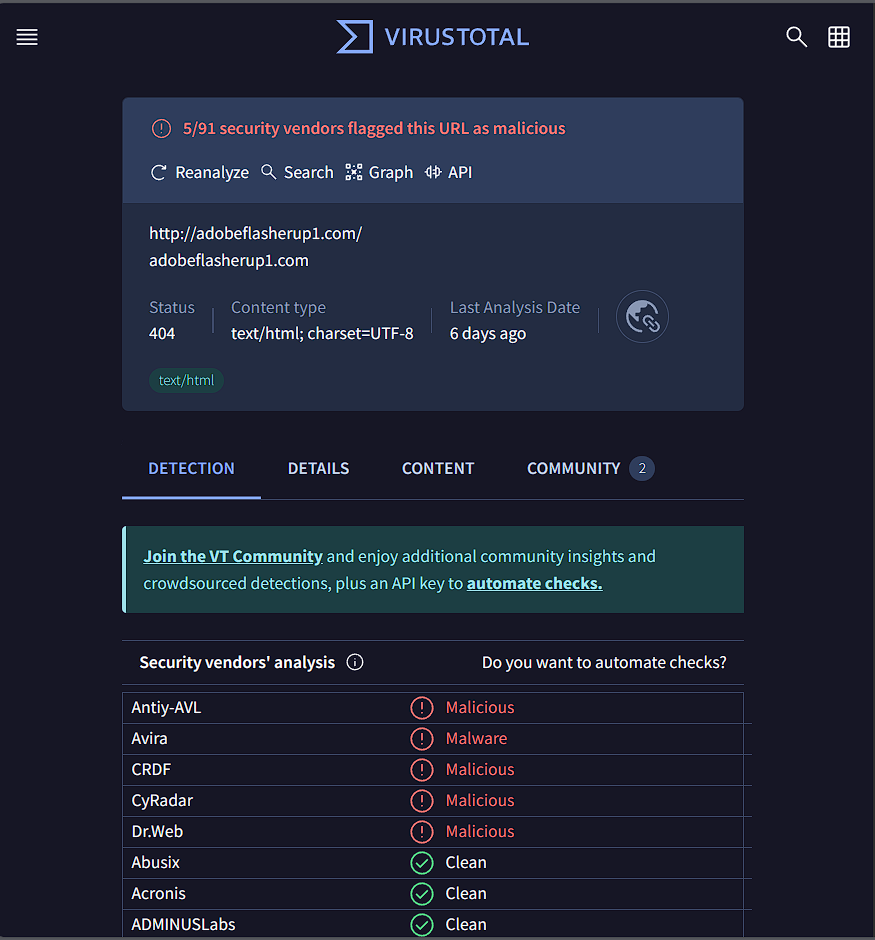
Searching these domains in the VirusTotal website, we found that:

* javaoracle2.ru, there is 3/91 security vendors in this URL that could be malicious.
* adobeflasherup1.com, there is 5/91 security vendors in this URL that could be malicious.



Security vendor:

* Antiy-AVL, a mobile antivirus engine, detects malicious contents within the domain name
* Avira, German computer security software company, has detected malicious content within the domain name
* CRDF, an independent IT security lab, has also detected malicious content in the domain name



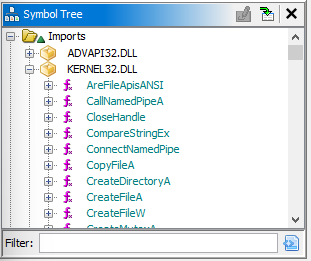
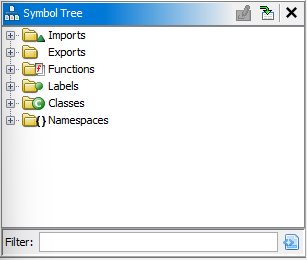
Security vendor:

* Antiy-AVL, a mobile antivirus engine, detects malicious contents within the domain name
* Avira, German computer security software company, has detected malicious content within the domain name
* CRDF, an independent IT security lab, has also detected malicious content in the domain name
* CyRadar, a cybersecurity company, detects malicious contents within the domain name
* Dr.Web, popular antivirus software, has also detected malicious content in the domain name

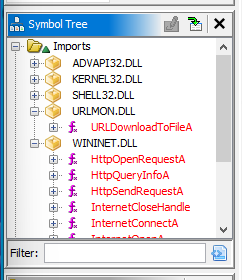
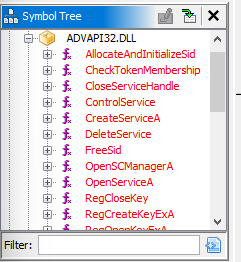
Thus, can help when analyzing malware by adding more details from public resources

Symbol Tree Exploration

Binary references malicious servers must be done by a network communication, or in this case by an HTTP protocol. This is seen by the import functions in the Symbol Tree.

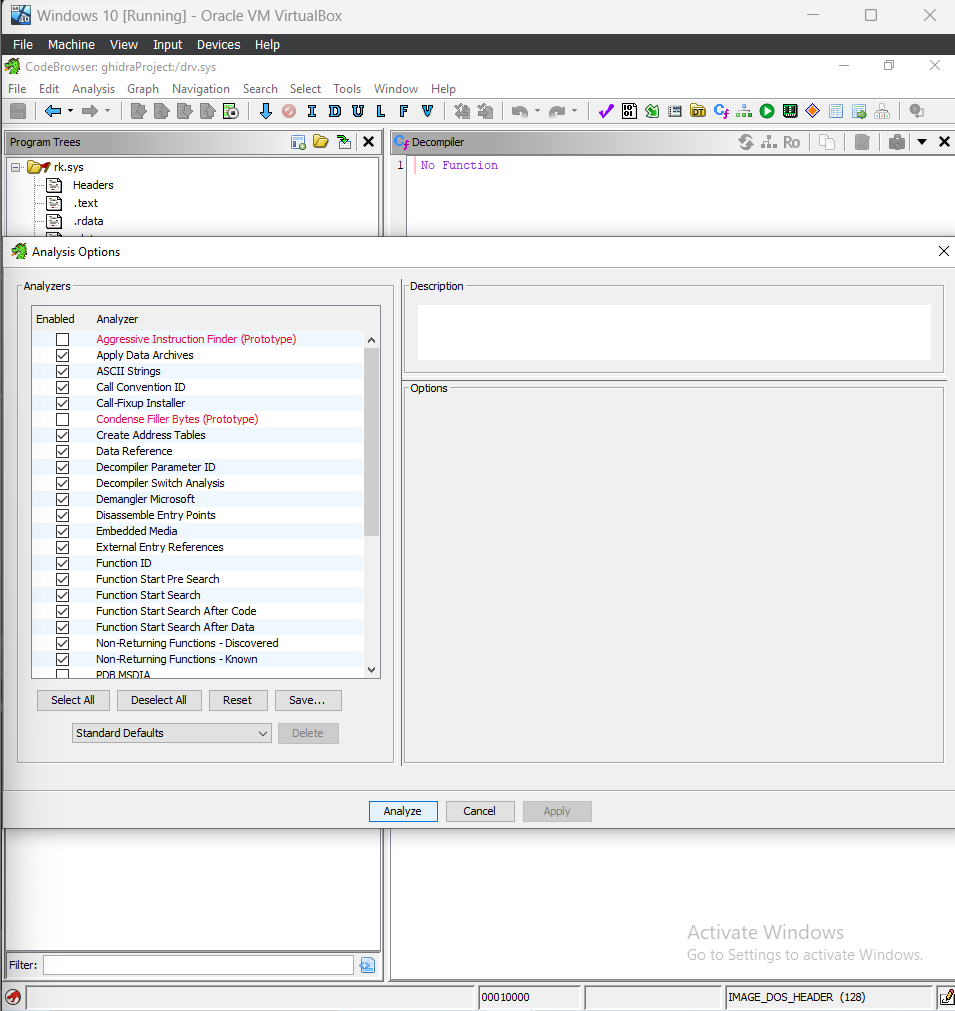






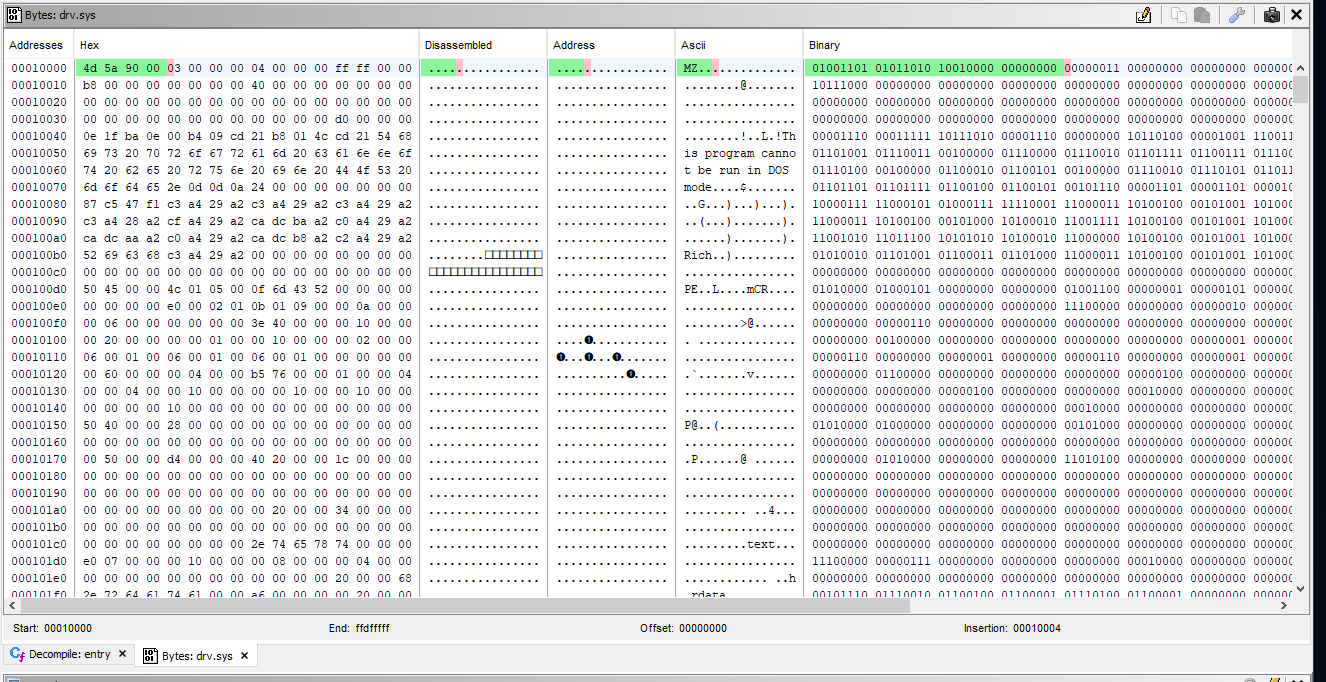
Within the ASVAPI32.DLL, there is certain functions named Reg\* that deals with Windows Registry, Service/SCManager that deals with the Windows Service Control Manager.

Compilation for the file:

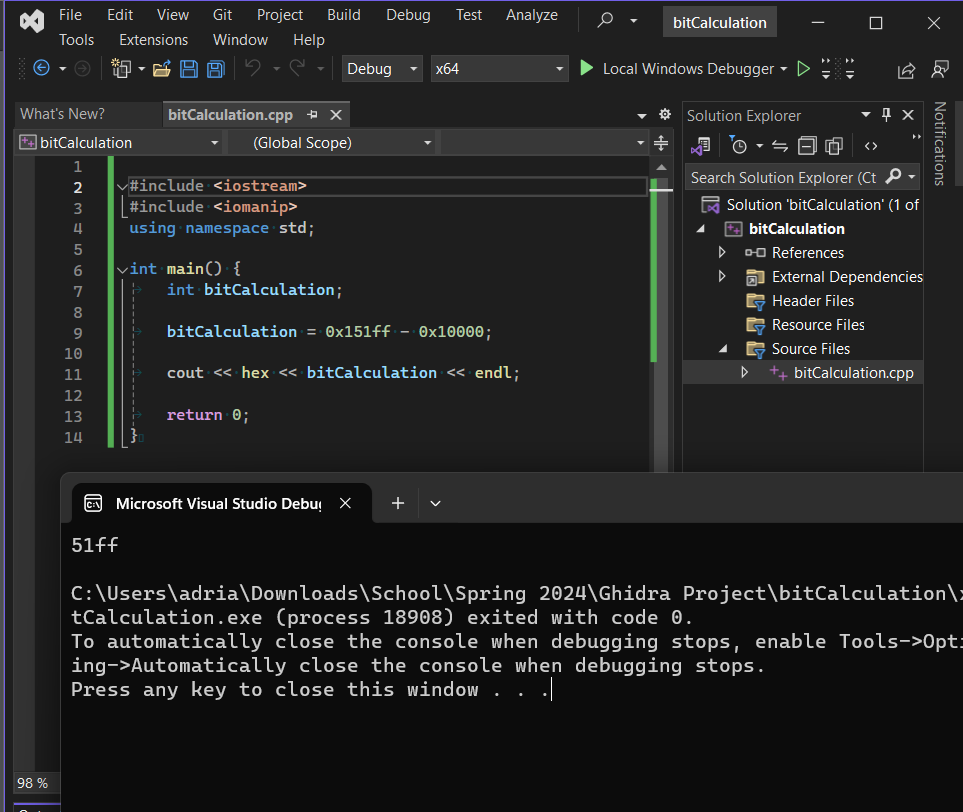


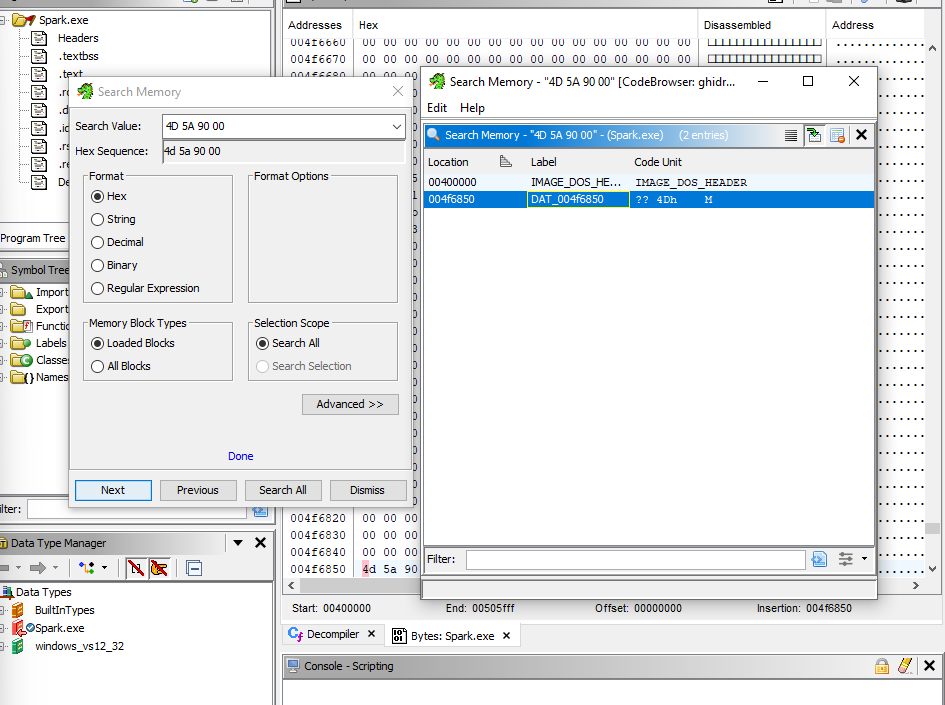
Malicious files tend to generate other files, and in this case the spark.exe must be clicked on to work.

The analysis of the imports, we notice that spark.exe has APIs to deal with the Windows Service Control Manager. Starting bytes is 4d 5a 90 00, can be used as the signature of the file; indicates the file is a Portable Executable

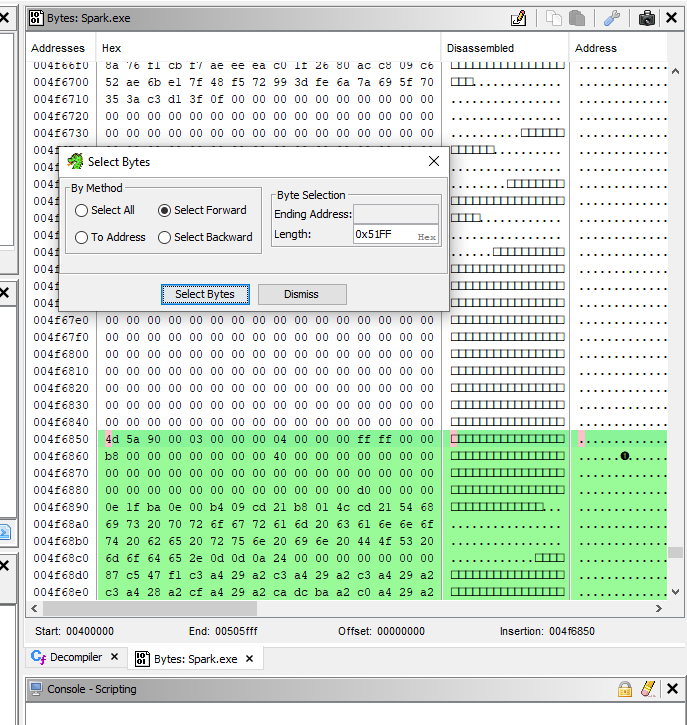


Calculate the difference with the start address and end address, we get the size of the file which is 0x051ff. This will be used when extracting rk.sys file within the spark.exe.

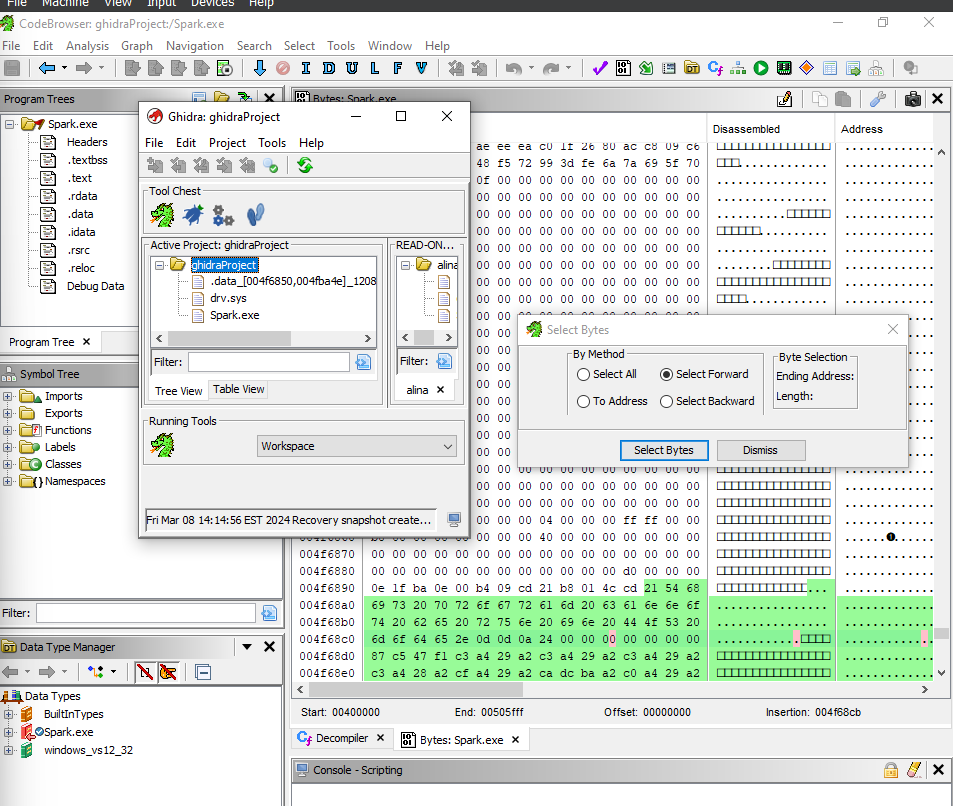




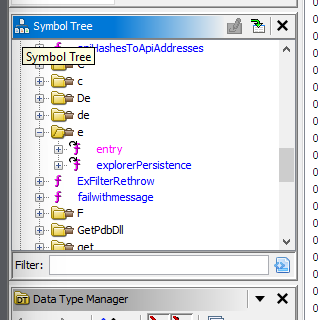
Start by with the spark.exe, and search within the memory for the 4D 5A 90 00 pattern



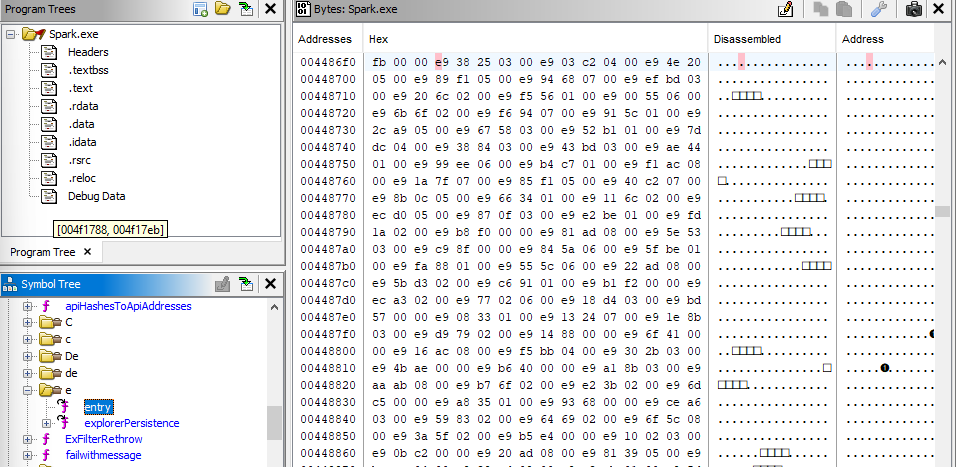
After the calculation from earlier, we get 0x51FF bytes in size, and then we will select the bytes with the next few steps.

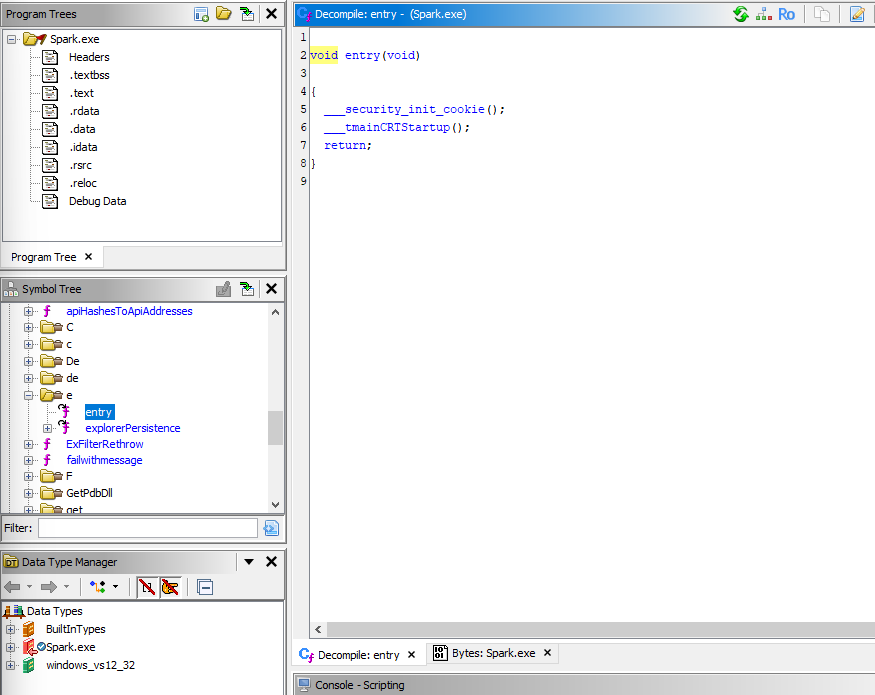


Once found we will extract and import, where the .data file is added to the project with the selected bytes.



Now, we begin to analyze the malware at the main entry of the program. Open the CodeBrowser, and we begin to look for the entry point.



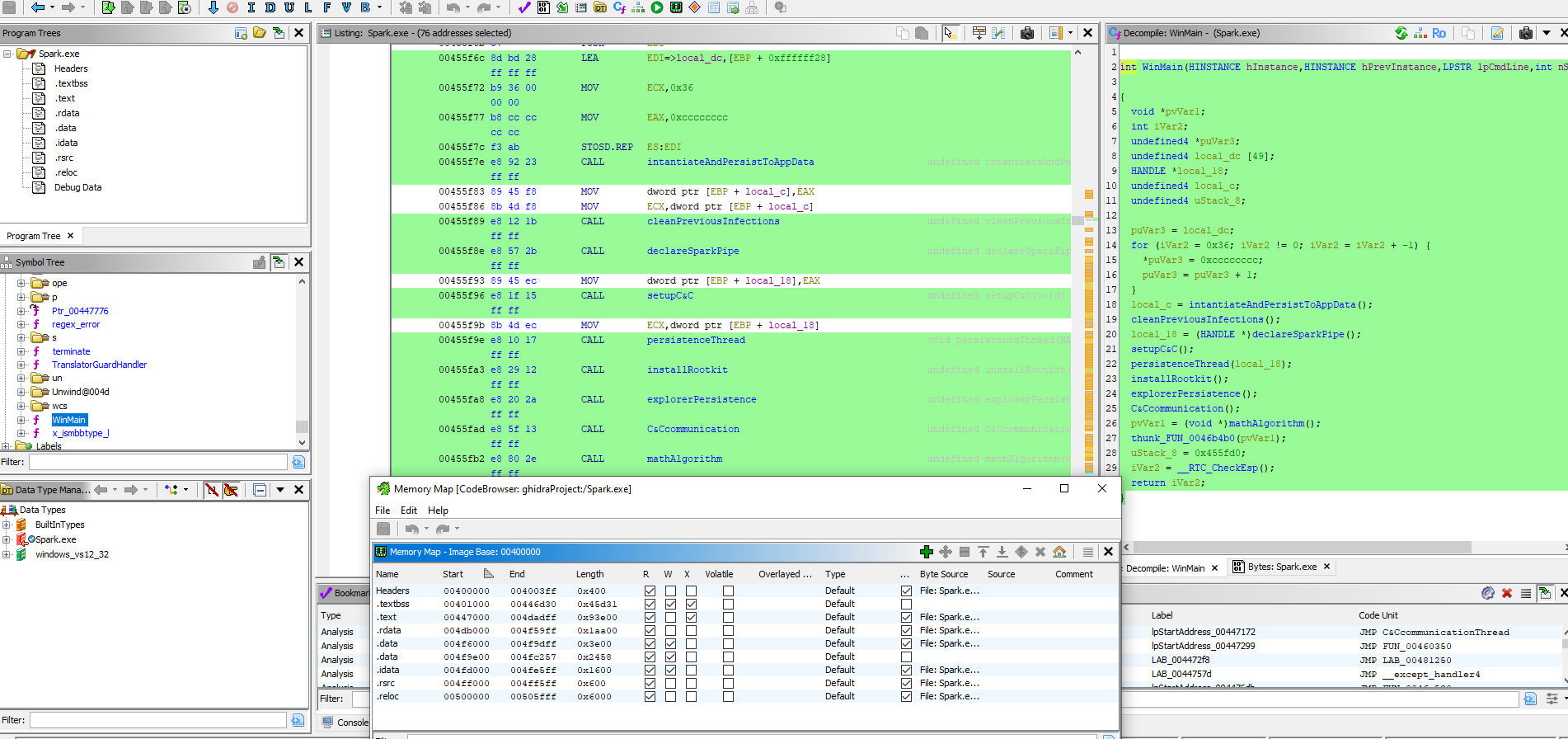


From here, we are looking at the \_\_security\_\_init\_cookie, and the \_tmainCRTStartup functions. The \_security\_init\_cookie is global security cookie used for exception handling, and buffer overrun protection in code complied with the Buffer Security Check. The \_tmainCRTStartup is an entry point function responsible for performing inintalization tasks before the main() functions is called.

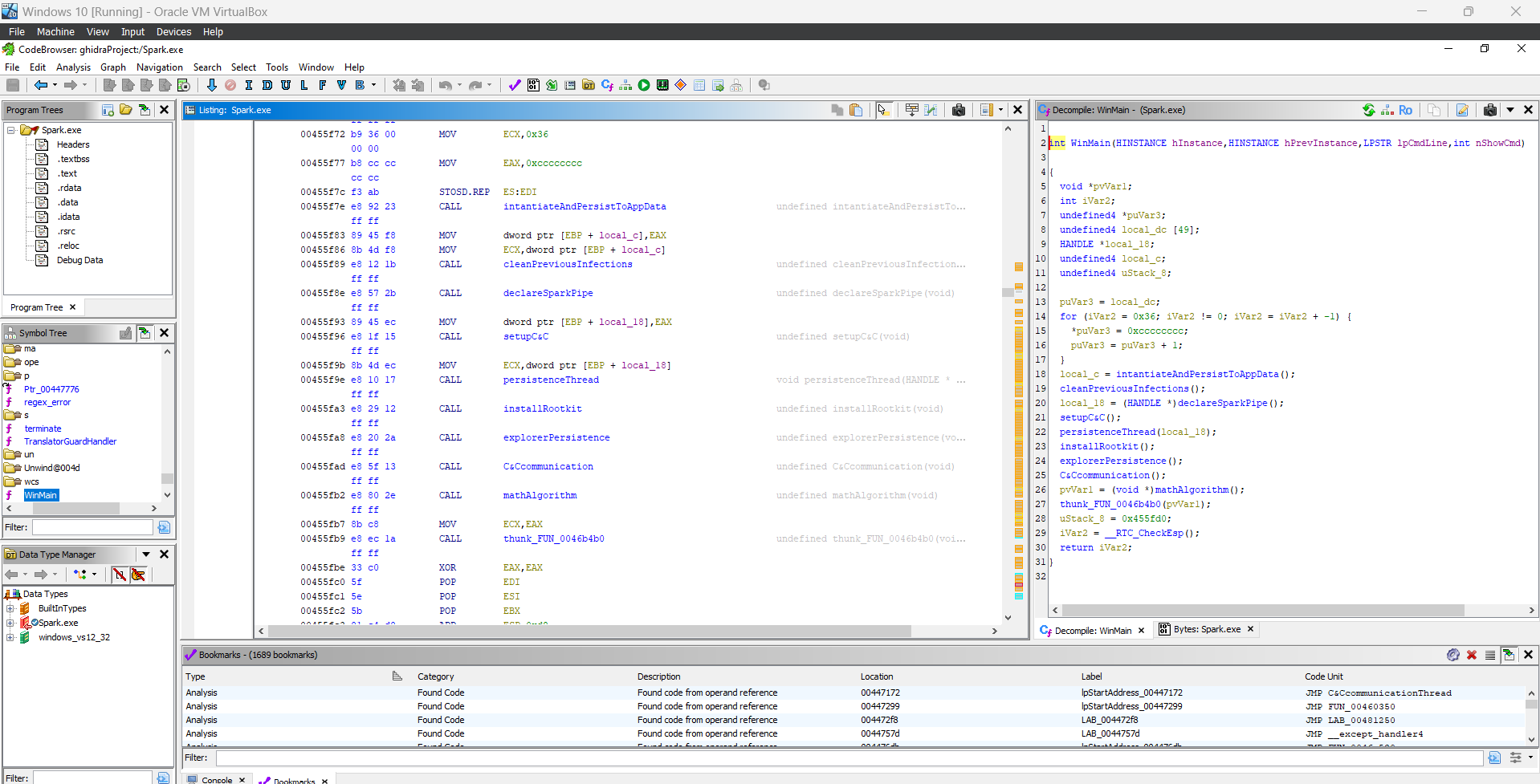
Part 1 Submission: 3/12

Ghidra Function – Memory Map

Memory maps allow you to see memory blocks of the binary and the associated permissions. In this case, we are looking at the WinMain function from the Spark.exe



Final Submission:



Listing for Spark.exe. View of the program's binary code represented as assembly language instructions.

00455f6c 8d bd 28 LEA EDI=>local\_dc,[EBP + 0xffffff28]

ff ff ff

00455f72 b9 36 00 MOV ECX,0x36

00 00

00455f77 b8 cc cc MOV EAX,0xcccccccc

cc cc

00455f7c f3 ab STOSD.REP ES:EDI

00455f7e e8 92 23 CALL intantiateAndPersistToAppData undefined intantiateAndPersistTo

ff ff

00455f89 e8 12 1b CALL cleanPreviousInfections undefined cleanPreviousInfection

ff ff

00455f8e e8 57 2b CALL declareSparkPipe undefined declareSparkPipe(void)

ff ff

00455f96 e8 1f 15 CALL setupC&C undefined setupC&C(void)

ff ff

00455f9e e8 10 17 CALL persistenceThread void persistenceThread(HANDLE \*

ff ff

00455fa3 e8 29 12 CALL installRootkit undefined installRootkit(void)

ff ff

00455fa8 e8 20 2a CALL explorerPersistence undefined explorerPersistence(vo

ff ff

00455fad e8 5f 13 CALL C&Ccommunication undefined C&Ccommunication(void)

ff ff

00455fb2 e8 80 2e CALL mathAlgorithm undefined mathAlgorithm(void)

ff ff

00455fb9 e8 ec 1a CALL thunk\_FUN\_0046b4b0 undefined thunk\_FUN\_0046b4b0(voi

ff ff

00455fcb e8 d9 25 CALL \_\_RTC\_CheckEsp undefined \_\_RTC\_CheckEsp()

ff ff

00455fd3 c2 10 00 RET 0x10

Decompile for WinMain. Process of converting machine code or assembly language instructions into a high-level programming language representation

Malware.c (after Exporting the current function to C in Ghidra)

int WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpCmdLine,int nShowCmd)

{

void \*pvVar1;

int iVar2;

undefined4 \*puVar3;

undefined4 local\_dc [49];

HANDLE \*local\_18;

undefined4 local\_c;

undefined4 uStack\_8;

puVar3 = local\_dc;

for (iVar2 = 0x36; iVar2 != 0; iVar2 = iVar2 + -1) {

\*puVar3 = 0xcccccccc;

puVar3 = puVar3 + 1;

}

local\_c = intantiateAndPersistToAppData();

cleanPreviousInfections();

local\_18 = (HANDLE \*)declareSparkPipe();

setupC&C();

persistenceThread(local\_18);

installRootkit();

explorerPersistence();

C&Ccommunication();

pvVar1 = (void \*)mathAlgorithm();

thunk\_FUN\_0046b4b0(pvVar1);

uStack\_8 = 0x455fd0;

iVar2 = \_\_RTC\_CheckEsp();

return iVar2;

}

As stated earlier, this part of the code seems to be the main issue for this executable file. Since we see the introduction of a rootkit, where a potential hacker uses gain root-level access. Where we see a set of programs installed for admin access in our case, malicious and stealthy changes can happen to the host O/S.

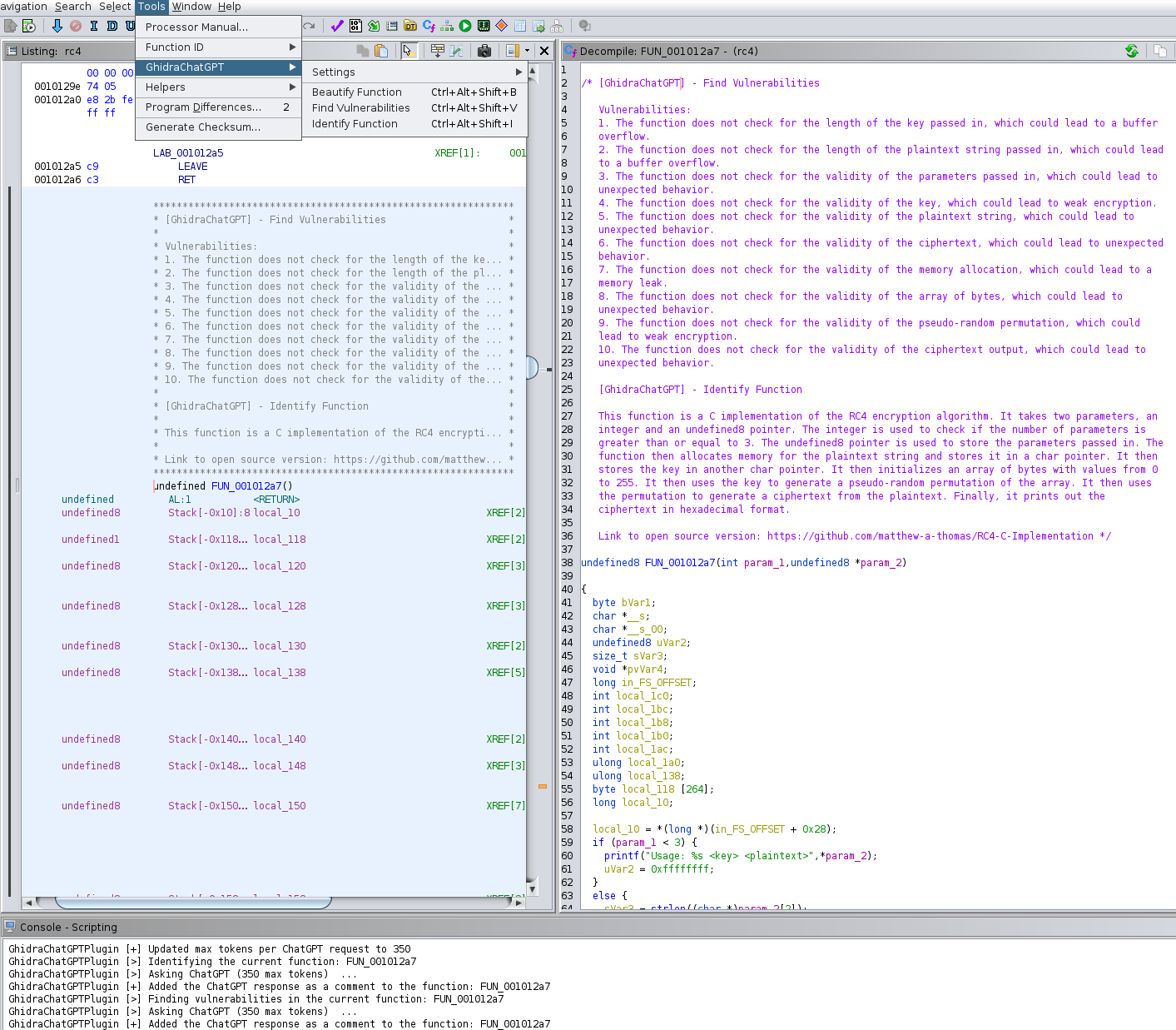
Ghidra Plugins

Note: Many of these plugins are not supported for the newer versions of Ghidra and require much older versions to run successfully. Below are some unique plugins we found.

GhidraChatGPT

Plugin that brings the power of ChatGPT to Ghidra

Note: Only currently supports Ghidra version 10.3.3

Example from Source:

Instructions to Install:

* Download latest release from the repository, and must match your Ghidra version
* Copy/Move the archive to the GHIDRA\_INSTALL\_DIR/Extensions/Ghidra directory
* Project window of Ghidra: File – Install Extension – Enable the GhidraChatGPT extension
* Restart Ghidra
* CodeBrowser window of Ghidra: File – Configure – Plugin icon in the top right – Enable the GhidraChatGPT plugin

Build

Plugin can be built with or without docker. Varies on method done to resolve the dependencies

* Clone repository
* Build using docker: ./build.sh -p YOUR\_GHIDRA\_INSTALL\_DIR -d or without: ./build.sh -p YOUR\_GHIDRA\_INSTALL\_DIR
* Project window of Ghidra: File – Configure – Plug icon in the top right – Enable the GhidraChatGPT plugin

**Source:** [GitHub - likvidera/GhidraChatGPT: Brings the power of ChatGPT to Ghidra!](https://github.com/likvidera/GhidraChatGPT)

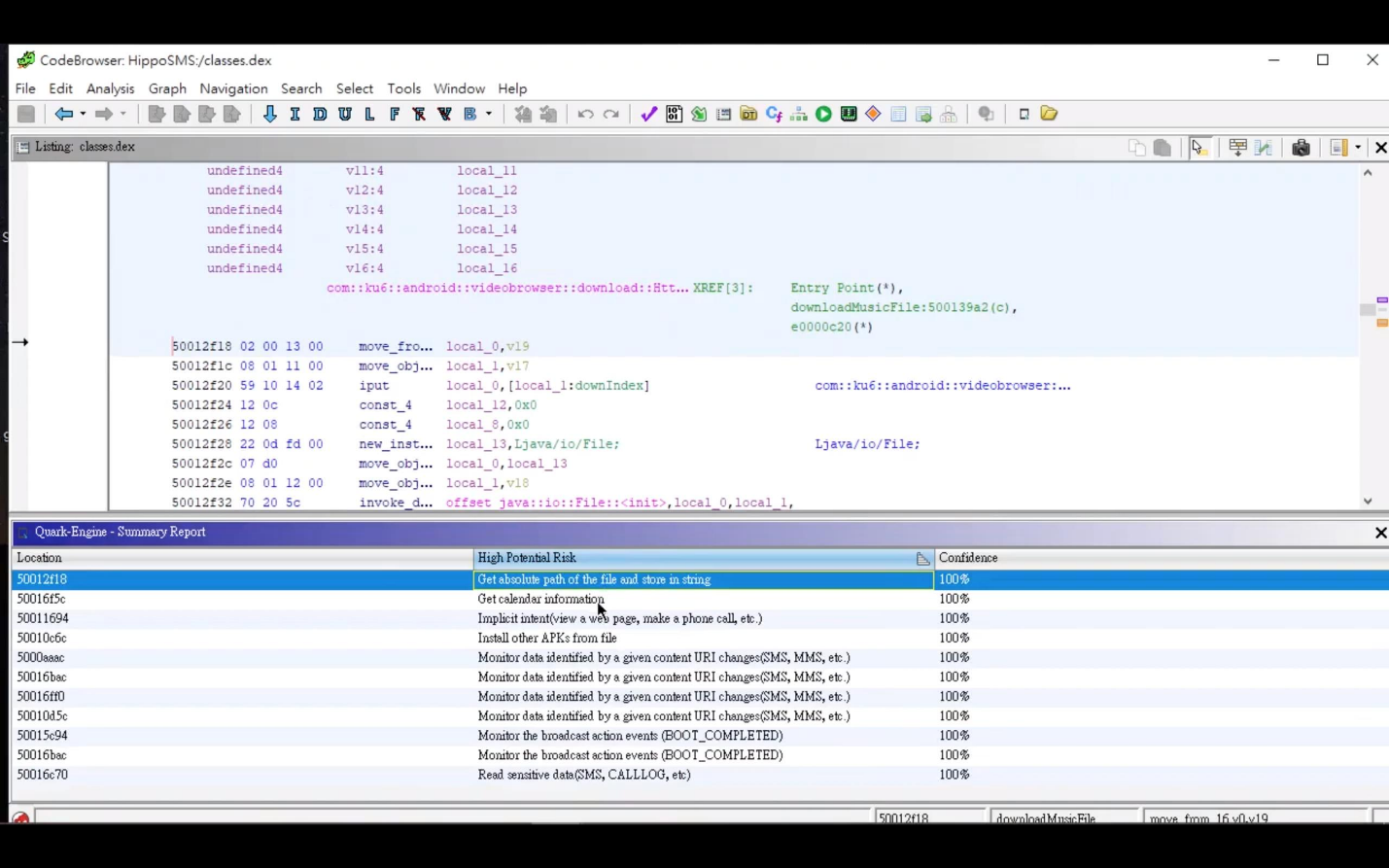
GhidraQuark

Another Ghidra plug-in that gives a powerful overview for Android Apps

Requirements:

* Note: Only supports Ghidra version 9.2.2
* Only one version available to download

Example from Source:



Installation

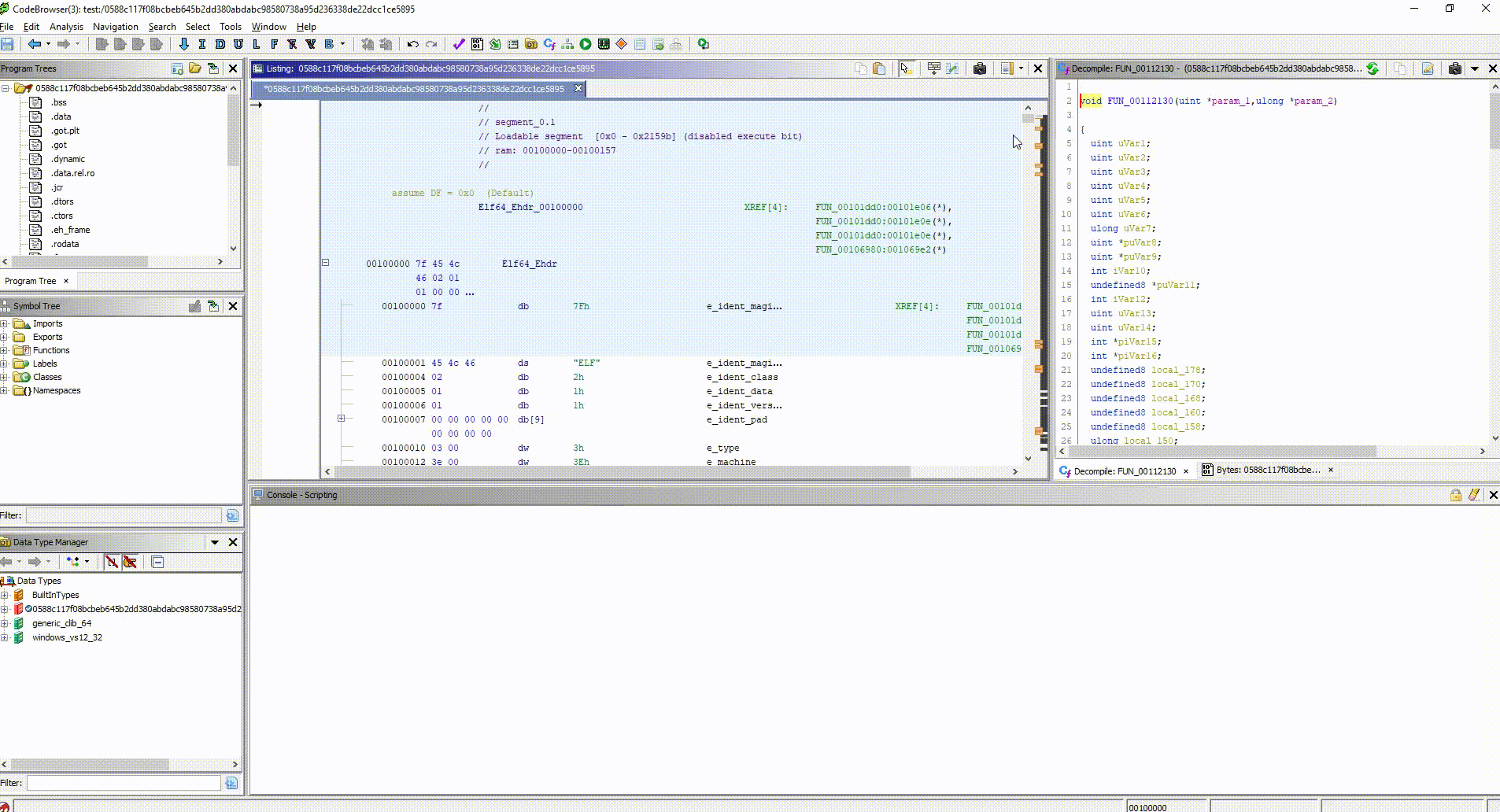
* Download the built extension from GitHub
  + Check the Releases Page for more recent version
  + Download the built extension zip file: ghidra\_9.2.2\_PUBLIC\_20210204\_QuarkEngineHelper or newest file
* Have Ghidra installed and install the extension by the following steps:
  + Startup Ghidra
  + Open File – Install Extensions...
  + Press the + icon on the upper right side of the Install Extensions window
  + Find the file location where you downloaded the zip file from previous step and choose it
  + Press ok, and restart Ghidra for the changes to take effect

Source: [GitHub - quark-engine/ghidraquark: GhidraQuark bridges Quark Engine into Ghidra](https://github.com/quark-engine/ghidraquark)

Intezer Analyze Ghidra Plugin

Plugin allows Ghidra to save time while reversing and focus on the malicious and unique functions

Example from Source:



Requirements

* Python request HTTP library at version 2.27.1 or newer
* Ghidra Version is tested on 10.1 up to 10.1.4

Installation

* Clone the repo: $ git clone <https://github.com/intezer/analyze-community-ghidra-plugin.git>
* Add the API key to the environment variables, and then add a new variable named INTEZER\_API\_KEY with the API key value from earlier
* Once in Ghidra, in the CodeBrowser tool: click Window menu > Bundle Manger
* In the Bundle Manger, Click the Green + button at the top right corner of the window, once hovered a “Display file chooser to add bundles to list” should appear.
* Go to the directory “analyze-community-ghidra-plugin" wherever you cloned it by git and click OK. The path should appear in a table, and should have a checkbox to the left of it check while being in a green text
* While in Ghidra, from the CodeBrowser window: click Window menu – Script Manager – Filter ‘intezer’ - Double click ‘intezer\_analyze\_gh\_community.py’.

Source: [GitHub - quark-engine/ghidraquark: GhidraQuark bridges Quark Engine into Ghidra](https://github.com/quark-engine/ghidraquark)

Final Phase- WinDbg explanation and using WinDbg

Intro to WinDBG

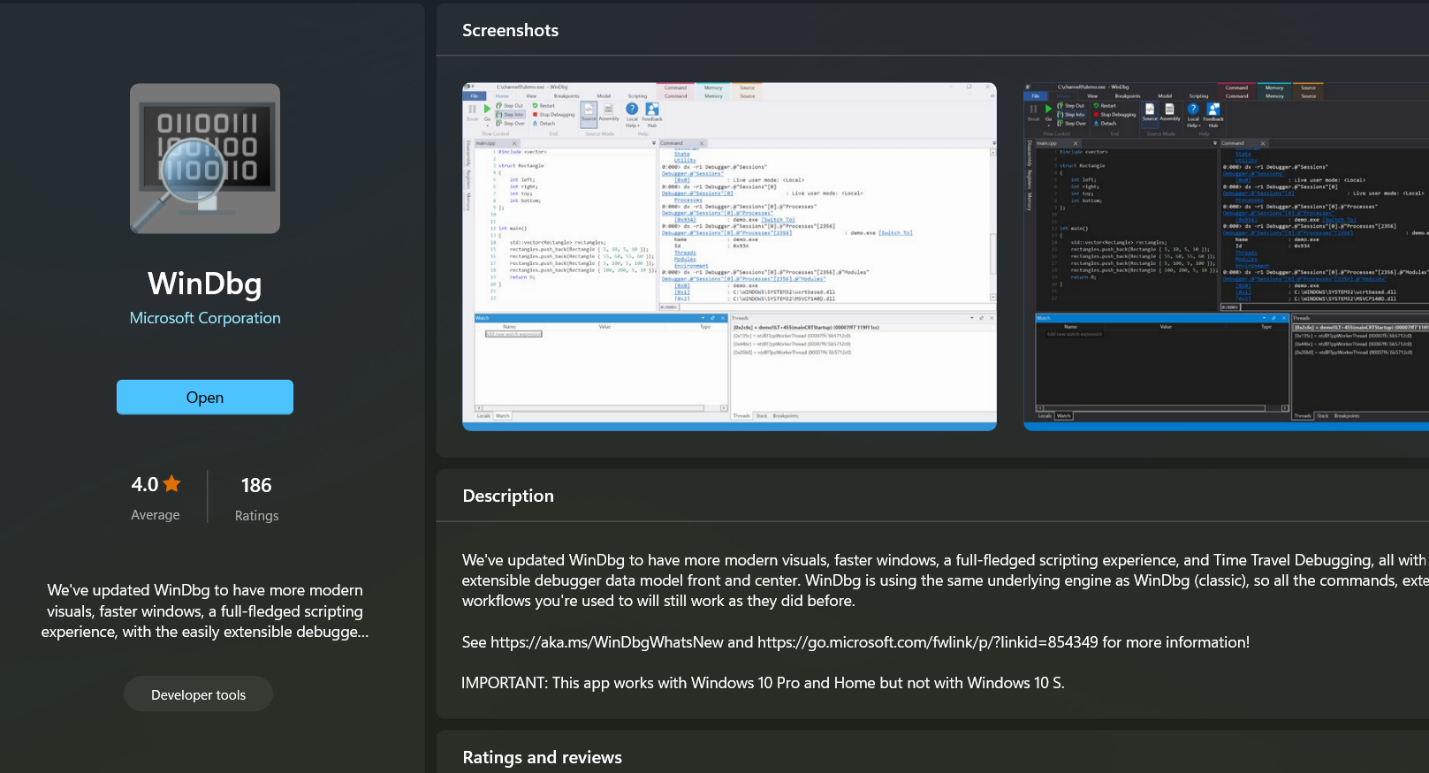
WinDBG is a powerful debugging tool by Microsoft for debugging Windows kernel-mode and user-mode programs. Used by developers, security researchers and system administrators to analyze and troubleshoot software and operating systems.

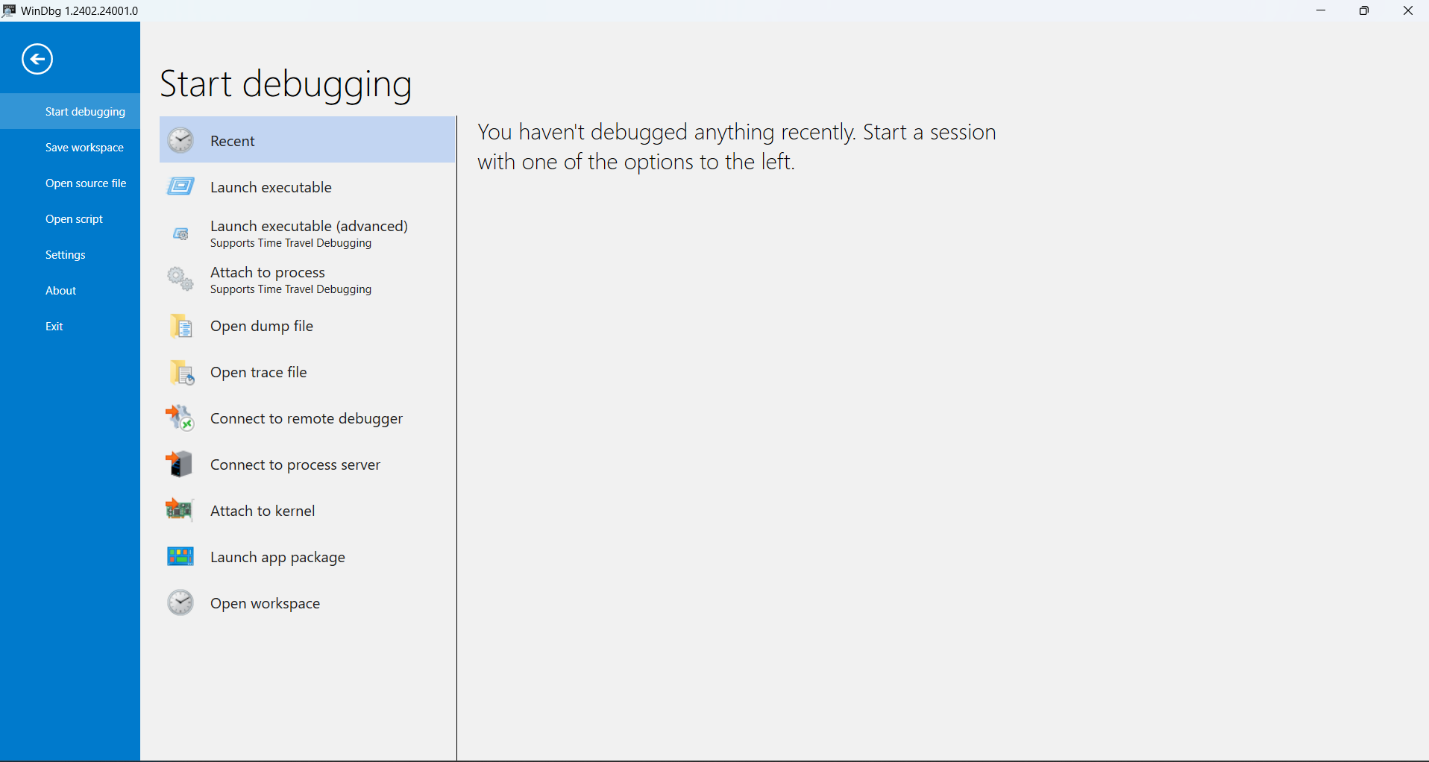
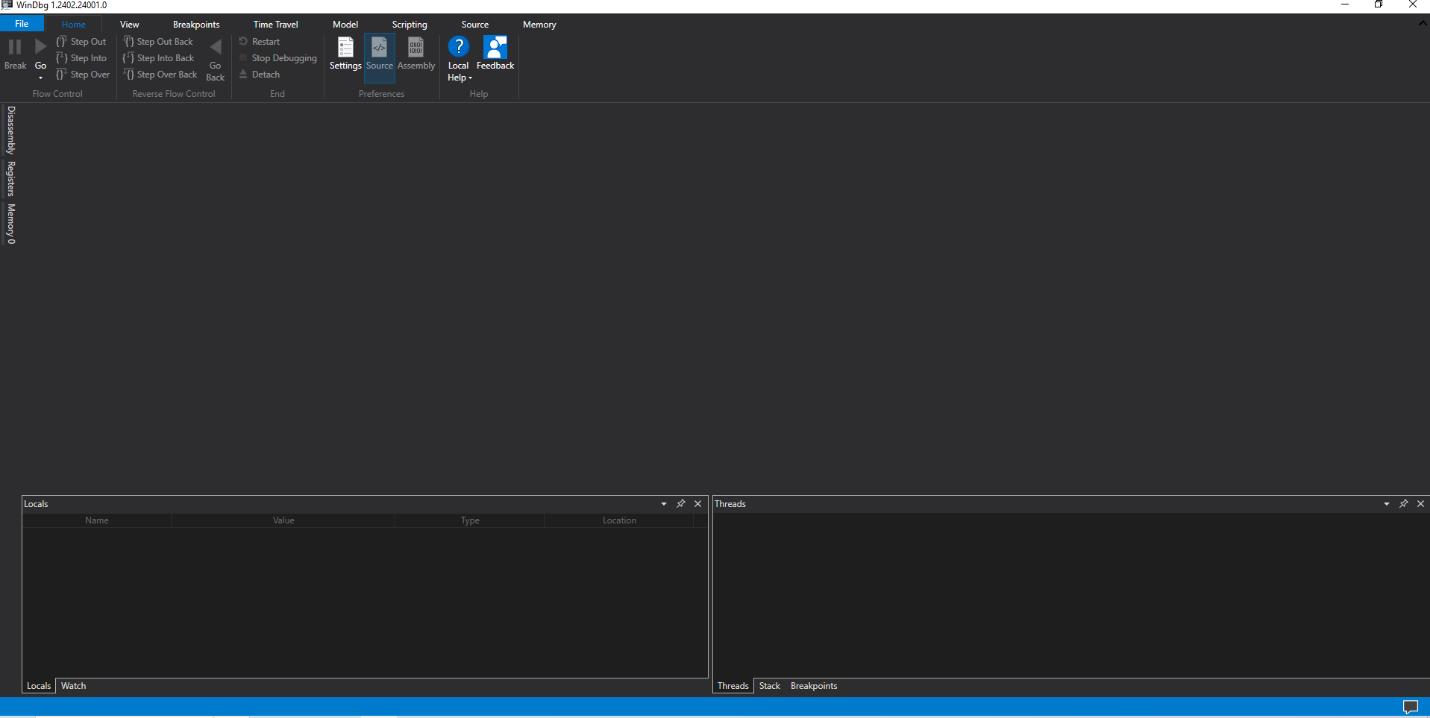
Installation:

From Online

A screenshot of a computer program

Description automatically generatedFrom Microsoft Store:

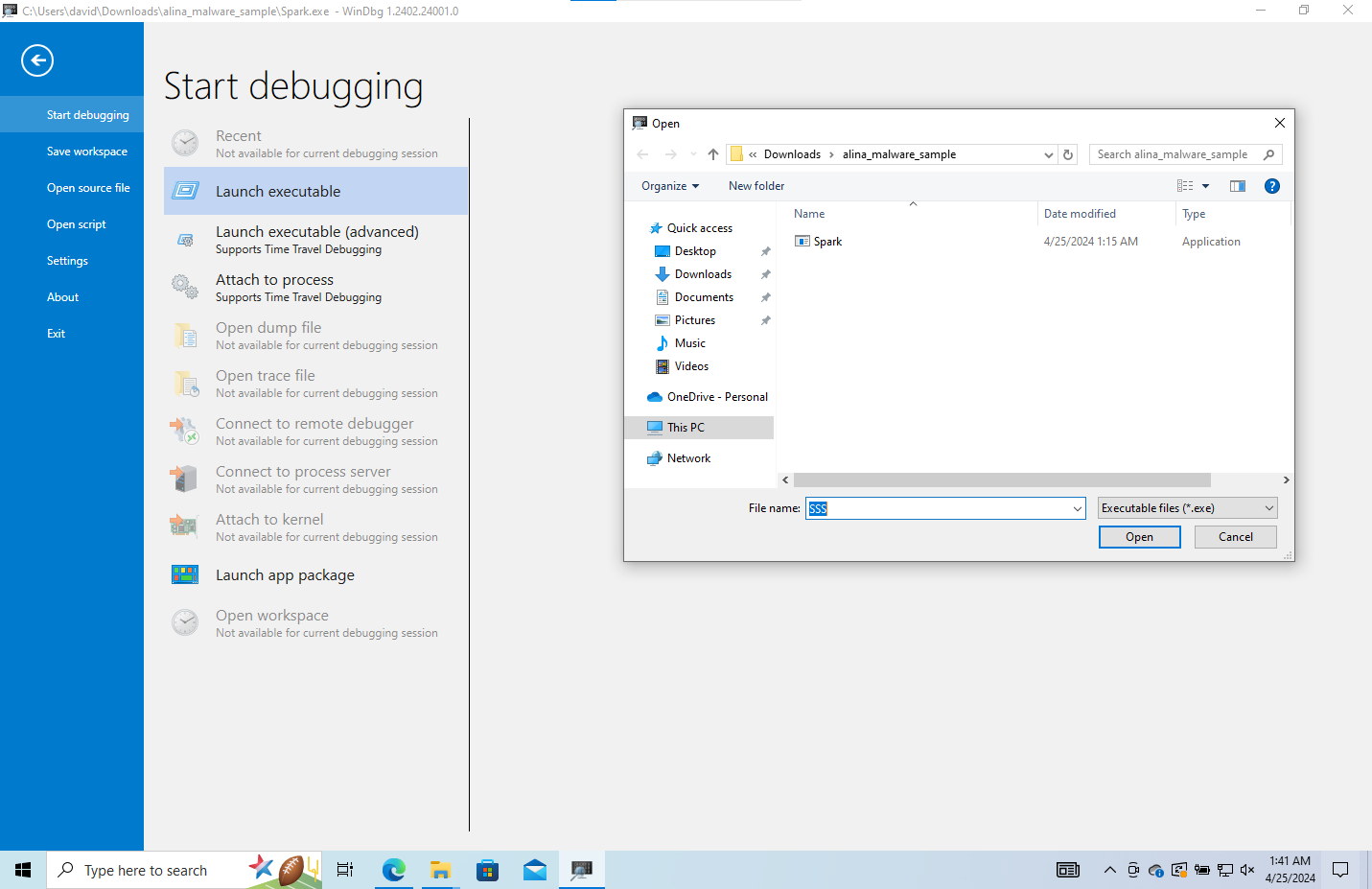




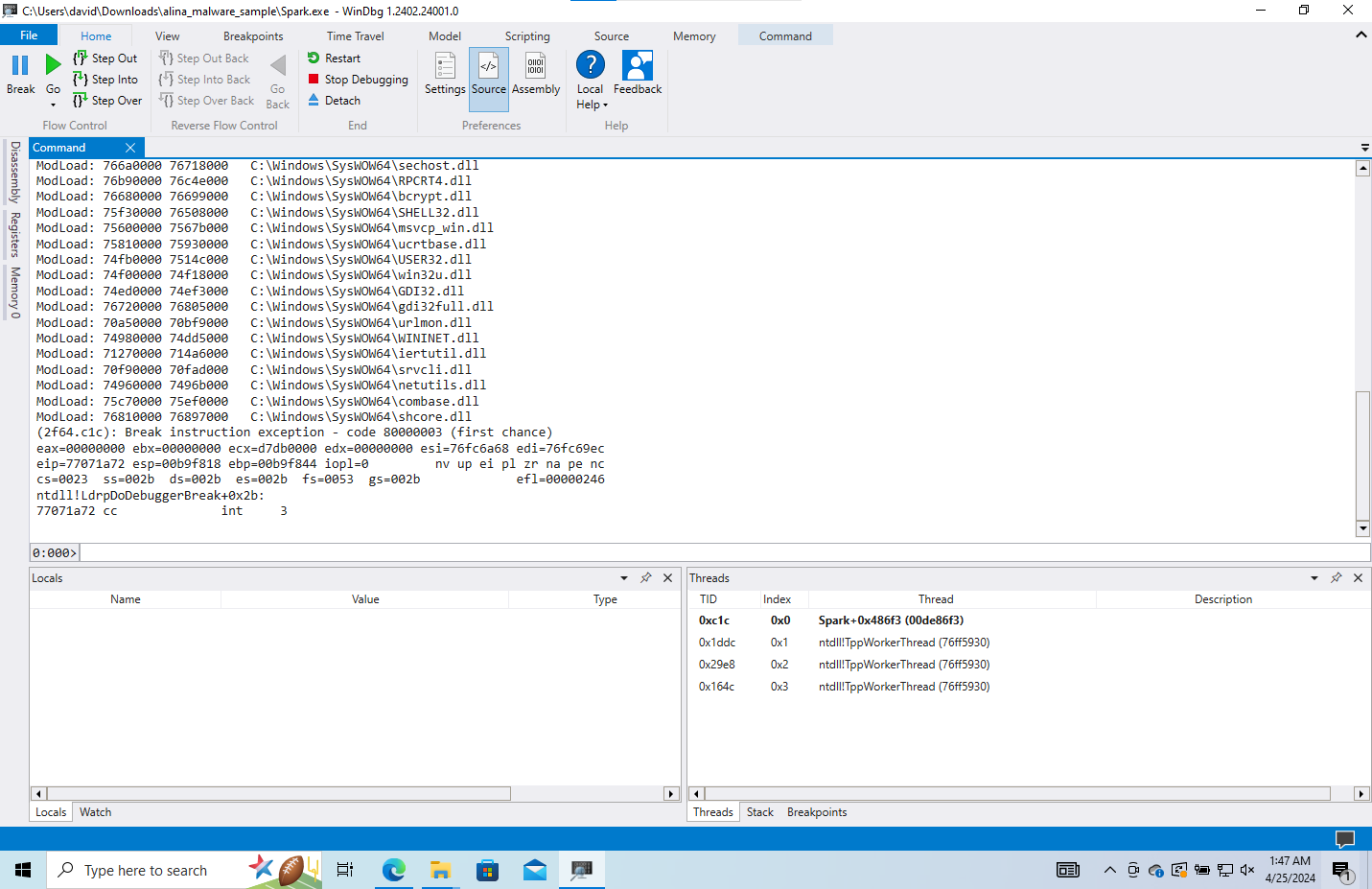
The WinDbg

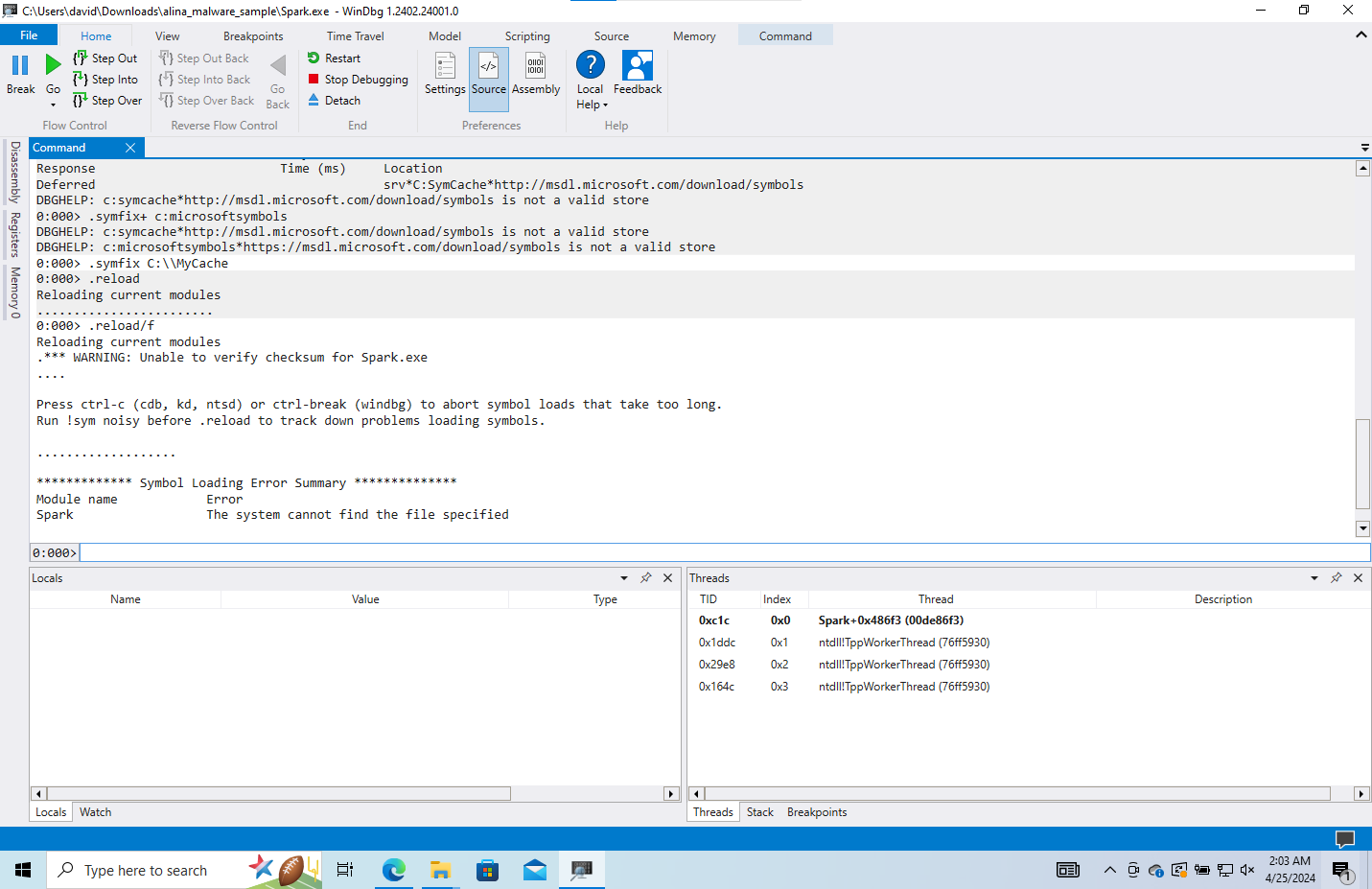
So overall, The Debugging of the Alina Malware (aka Spark.exe) was not a complete success, as an important asset known as symbols was unable to load into it which would’ve helped with the execution of the malware of the debugger, however much we wanted it to be working.

In the screenshot below is the insertion of the malware:



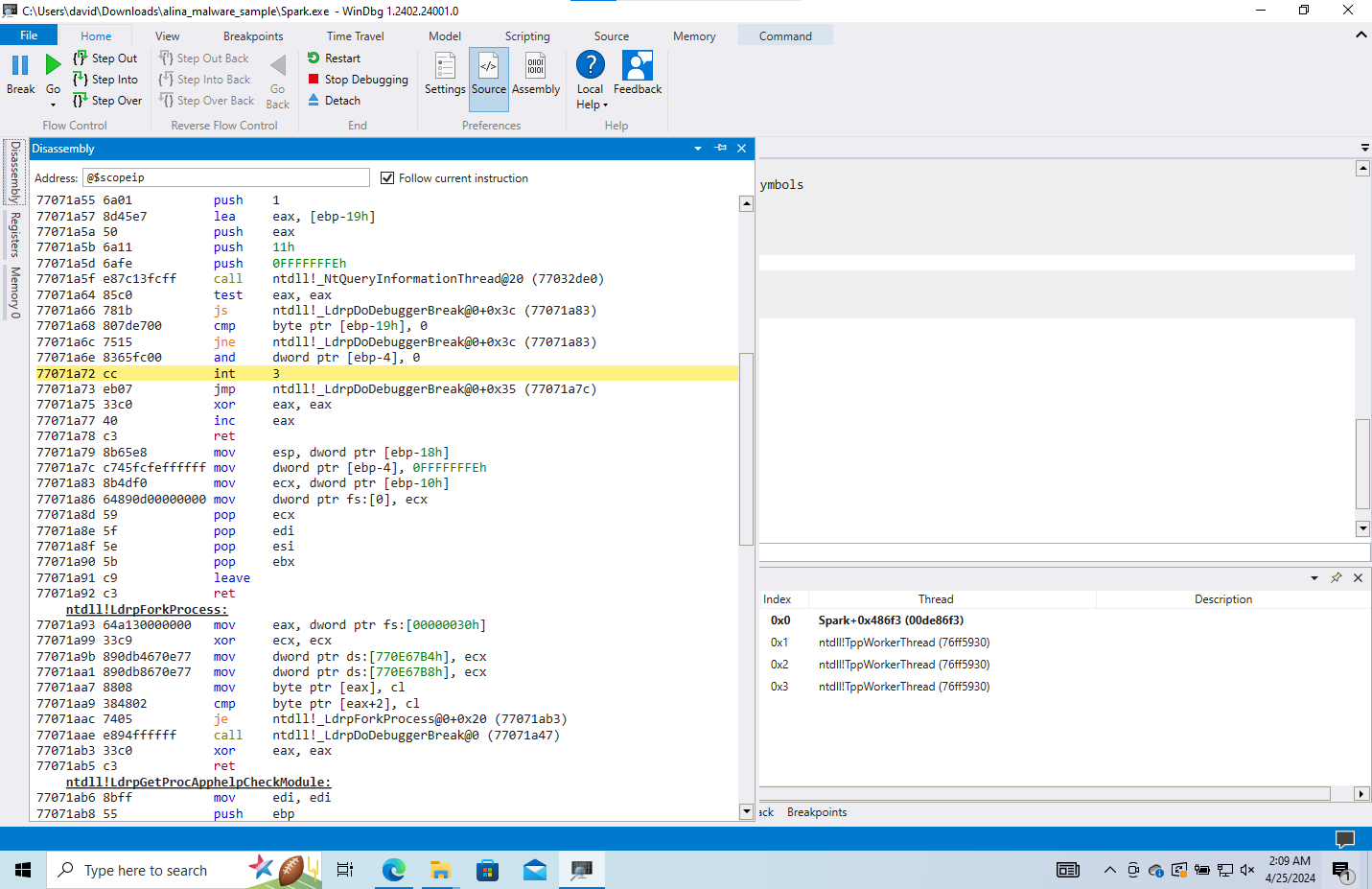
Initial set up of the malware within the debugger:



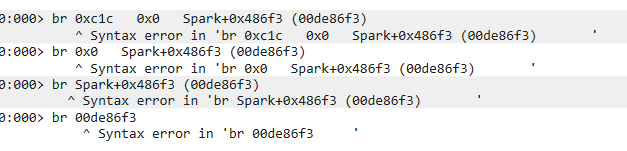
The following screenshot showcases the commands done in order to load in symbols and its result (Spoilers: it failed):  
(note: the file specified is literally running in the debugger right now, so there was literally no reason for it not to see anything)

Now, it is still technically possible to continue without the symbols however the issue is these symbols makes it [the debugging code] significantly easier to understand and is key to being able to do important things such as set proper breakpoints and find the API calls that we need to find to see what causes the malware to run its process.

To further emphasize this point, the following screenshot shows the disassembly of the code:



As you can see, we literally know only one thing, an integer. Everything else is supposed to be process and integers (which you can see somewhat clearly in Ghidra), but without the symbols, they are unintelligible.

This is just to show that even when given the thread, this isn’t valid because this might actually not be a significant address at all.

So, in short, we could not debug it simply because it was literally not possible and my theory has to do with the fact that it is a malware and when symbols are loaded in, it doesn’t work with the executable because it is using API + thread calls that are foreign to these symbols as an executable (as they have to be capable of symbols from its development if it is to work)

Source:

Main file used: <https://cdn.ttgtmedia.com/rms/pdf/excerpt_ghidra_bookshelf.pdf>

[\_\_security\_init\_cookie | Microsoft Learn](https://learn.microsoft.com/en-us/cpp/c-runtime-library/reference/security-init-cookie?view=msvc-170)

Virus Total: <https://www.virustotal.com/>

Ghidra – used to get binary and in-depth decompiling

WinDbg- used for debugging and in-depth research of code

Buffer Overflow Attack with Example - GeeksforGeeks

Assembly Language- <https://godbolt.org/>

Malware code using a listener- <https://cocomelonc.github.io/tutorial/2021/09/18/malware-injection-1.html>

WinDbg how to use - <https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/getting-started-with-windbg>

212 Project- <https://drive.google.com/file/d/1ZU_4qqZunyZ5OK_uHgXP47bR4l2SWbhU/view?usp=sharing>

[Lectures (ligerlabs.org)](https://ligerlabs.org/lectures.html)