

CS 499/ISA 564: Lab 4 – Malware Analysis and RE

Lab Submission Guidelines

Submissions must be in Microsoft Word or PDF format. Be sure to clearly label what question you're answering. Where possible, screenshots should be embedded directly in the document. Screenshots should be cropped to only include what is necessary to answer the question. If you need to save them as a separate file, save them in compressed format (gif, jpg, png) and name them after the question they pertain to. If your submission contains multiple files, archive them (zip, 7z, tar.gz) and submit the file via Blackboard.

Lab Requirements

- 32-bit Windows 7 VM
- Files in the Lab 4 Resources folder

Windows 7 VM Setup

- Copy the Tools and AnalysisFiles folders to an easily accessible location (like the Desktop)
- Install IDA Free
- Run the win.reg file which will disable System Restore as well as the Security Center, Windows Defender, and Windows Firewall services. Reboot your VM
- Configure networking as Host-Only
- Take a snapshot of your VM after finalizing its configuration

Lab 4: Task 1 – Static File Analysis

For this exercise we will be using Lab4.1.exe. The purpose of this exercise is to learn what we can about the file through Internet research and examination of its metadata and properties.

Use hashcalc to get the MD5, SHA-1, and SHA-256 hashes of the binary. Search for any of the hashes on VirusTotal.

Question 1.1 – What was the most recent submission date and detection ratio? Did the AV engines reach any kind of useful consensus for the file (malware name/family)? What other information does VT provide and what value does it have for the purpose of malware analysis (check details and behavior tabs)?

Perform a Google search for the hashes or any information you think is useful from VT.

Question 1.2 – Summarize the search results. Did you learn anything useful, if so what? Provide at least one reference to corroborate your claim. If you found nothing useful, provide a screenshot of your Google search.

Use bintext to search the binary for strings. You can use the Filter tab to change the conditions for string matching. Use this to filter out unhelpful results. If you know what kind of string you're searching for (a URL for example), you can use Stage 3 to require certain characters.

Question 1.3 – Provide a list of useful or informative strings from the binary. Pay close attention to URLs, registry keys, and file names.

Use CFFExplorer (cffe.exe) to examine the import table of the binary (it will be under Import Directory).

Question 1.4 – What functions does this binary import? What do the IAT results suggest about the binary? (Hint: It is absolutely impossible for a functioning Windows binary to only have one imported function). There is a provided tool you can use to confirm your suspicions. Provide a screenshot of that tool's output.

Extra Credit (5 points) – CFF Explorer has many other features besides just providing the IAT. Explore the tool and see what else it can do. Provide a short write-up detailing what features you found useful and how they can be used to improve your analysis of the binary.

Lab 4: Task 2 – Dynamic Behavioral Analysis

For this exercise we will continue to use Lab4.1.exe. The purpose of this exercise is to learn what the program does when it is allowed to execute in a controlled and monitored environment.

Perform the following steps:

1. If you haven't done so already, take a snapshot of your VM
2. Run autoruns and save a pre-execution snapshot with File → Save
3. Run procmon. It will automatically start capturing system events, let it run
4. Run regshot and do the 1st shot. You don't need to save the shot as long as you keep regshot running
5. Execute the malware binary
6. Stop the procmon capture with CTRL+E and leave it running
7. In regshot do the 2nd shot and then compare
8. Run autoruns again and compare the results to the pre-execution snapshot with File → Compare
9. Once you have answered the Task 2 questions, revert your VM to your saved snapshot to prepare for the next exercise. You may wish to consider saving the tool output to files and copying them to your host machine in case you need them to revise your answers later

Question 2.1 – Provide a screenshot of the comparison that autoruns generated. What persistence mechanism (if any) did the malware employ and how does it work?

Question 2.2 – What changes to the Windows Registry did the malware make (if any)? Provide copy/paste output from regshot's comparison file to corroborate your claims. What effect do the Registry changes have on the system? Of what benefit were the Registry changes to the malware author? What can you infer about their intent from the changes?

Question 2.3 – Use procmon's filters to display only pertinent system changes that the malware made and provide a screenshot of procmon's output. Summarize the results. Of what benefit were the system changes to the malware author? What can you infer about their intent from the malware's behavior?

Question 2.4 – How did the results from your static file analysis inform your investigations into the malware's running behavior? What results from the analysis helped you decide what behavior to look for (if any)? Did the malware do something unexpected, something that you couldn't predict from static file analysis, if so, what?

Lab 4: Task 3 – Static Code Analysis

For this exercise we will be using Lab4.2.exe. The purpose of this exercise is to analyze binary code using a disassembler to determine its possible execution paths and to gain an understanding of its functionality.

The ultimate goal of this exercise is to discern enough information about the binary to inform future dynamic code analysis and debugging. To do so you must learn what conditions the binary requires to successfully execute and indeed what successful execution looks like.

The first step in this process is to identify the main function. IDA Free is rather deficient in its ability to detect main so I'll just tell you that it's sub_401460. Within the program code is a function that will trigger a self-delete for the binary as well as a few different checks of the operating environment and command-line arguments. Use IDA to analyze the program code to answer the following questions.

Question 3.1 – Provide the function name or address that does the self-deletion and explain how it works. Your explanation must include a description of what the Windows API calls within the function do.

Question 3.2 – Provide the names or addresses of the functions that call the deletion function. For each one, explain what check it performs and how you can either make the check pass legitimately or modify the program's instructions or CPU registers to make it pass. Provide exact memory addresses or precisely explain where the changes need to be made.

Question 3.3 – Summarize the conditions necessary to make this program execute successfully and describe what successful execution entails.

Extra Credit (5 points) – Provide the installation password asked for in one of the check functions.

Lab 4: Task 4 – Dynamic Code Analysis

For this exercise we will continue to use Lab4.2.exe. The purpose of this exercise is to learn how to analyze and modify a program's execution flow using OllyDbg.

Left to its own devices, this program will call the Windows API function ShellExecuteA and delete itself. Your objective is to find the execution path that leads to this function being called, and to patch the binary such that it doesn't happen while still allowing the program to execute and do everything else that it's programmed to do. There are multiple ways to accomplish this objective; you may use any that you wish. Consult the Lecture 6 slides to understand how patching works and use the IDA-Olly-Shortcuts image file in the Readings folder for useful OllyDbg commands.

Question 4.1 – Provide a screenshot showing successful execution of the program (successful in this case meaning that it does what it's supposed to do and doesn't delete itself). Provide screenshots of the patches you made to the running binary.

Extra Credit (5 points) – It is possible to make this program execute successfully by changing just a single byte. Provide either a screenshot of your single-byte patch or the address of the byte change with its old and new values. Note that the deletion routine will still happen, but that's ok. Also note that solving it this way does not also give you credit for question 4.1 so you must still provide a separate solution for that question.

Extra Credit (5 points) – Why is it that even though the deletion routine happens BEFORE the installation routine, the single-byte patch still works? Hint: Use procmon to delve into how the deletion routine works.

Grading

- **Task 1 – 20 points (5 points per question)**
- **Task 2 – 20 points (5 points per question)**
- **Task 3 – 30 points (10 points per question)**
- **Task 4 – 30 points (30 points per question)**
- **Extra Credit – 20 points**