West Chester University

CSC 471

Dr. Si Chen

Spring 2024 Lab 3

Submitted by

Sean Berlin

4/9/24

1. Introduction:

The purpose of this lab is to develop a heuristic malware detection system utilizing static analysis techniques. Through analyzing Portable Executable (PE) files, students aim to identify potential malware samples based on specific heuristic rules. The primary motivation behind this experiment is to gain a deeper understanding of Import Address Table (IAT) and Export Address Table (EAT) concepts and acquire practical experience in utilizing static analysis methods for malware detection.

2. Analysis and Results:

Screenshot of Modified enum export.py

```
root@8624d065b4c9:/workdir # cat enum_export.py
import pefile
import sys

import os #for interacting with file systems
import collections #to count hashable objects


#validate argument as a correct folder path
if len(sys.argv) != 2:
        print("Please provide the folder path as an argument")
        sys.exit(1)

#extract the folder path from the command line arguments
folder_path = sys.argv[1]


#transverse each file in the specified folder recursively using os.walk
for root, dirs, files in os.walk(folder_path):
        for file in files:
                filePath = os.path.join(root, file)
                try:
                        #attempt to open file as PE
                        pe = pefile.PE(filePath)
                except pefile.PEFormatError:
                        #handle case when file is not PE
                        print(f"File {filePath} is not in PE format")
                        continue
                except OSError as e:
                        #handle errors reading file
                        print(f"Failed to read {filePath}: {e}")
                        continue

                if hasattr(pe, 'DIRECTORY_ENTRY_EXPORT'):
                        addresses = [exp.address for exp in pe.DIRECTORY_ENTRY_EXPORT.symbols if exp.address is not None]
                        #sort addresses to continually compare
                        addresses.sort()

                        # Check for three or more export functions have the same memory address
                        addressTotal = collections.Counter(addresses)
                        sameAddrViol = any(count >= 3 for count in addressTotal.values())

                        # Check for three or more export functions have the same memory offset
                        offsets = [addresses[i+1] - addresses[i] for i in range(len(addresses) - 1)]
                        offsetTotal = collections.Counter(offsets)
                        memoryDiffViol = any(count >= 2 for count in offsetTotal.values())

                        violationList = []
                        if sameAddrViol:
                                violationList.append("Rule 1")
                        if memoryDiffViol:
                                violationList.append("Rule 2")

                        if violationList:
                                print(f"Potential malware detected in {filePath}. Violates {' and '.join(violationList)}")
                        else:
                                print(f"NO potential malware detected in {filePath}.")
                else:
                        print(f"No export functions found in {filePath}.")
```

Figure 1

Screenshot of output running all code in malware.zip (malware_lab_1)

```
root@8624d065b4c9:/workdir # python3 enum_export.py malware_lab_1
Potential malware detected in malware_lab_1/6a764e4e6db461781d080034aab85aff. Violates Rule 2
Potential malware detected in malware_lab_1/e0bed0b33e7b6183f654f0944b607618. Violates Rule 2
Potential malware detected in malware_lab_1/0fd6e3fb1cd5ec397ff3cdbaac39d80c. Violates Rule 2
Potential malware detected in malware_lab_1/1c1131112db91382b9d8b46115045097. Violates Rule 2
Potential malware detected in malware_lab_1/16d6b0e2c77da2776a88dd88c7cfc672. Violates Rule 1 and Rule 2
Potential malware detected in malware_lab_1/cc3c6c77e118a83ca0513c25c208832c. Violates Rule 2
```

Figure 2

Analysis:

Generated by ChatGPT- The lab experiment focuses on developing a heuristic malware detection system through static analysis of Portable Executable (PE) files. Implemented within the provided Python script, enum_export.py, the heuristic rules aim to identify potential malware by scrutinizing Export Address Tables (EATs) for suspicious patterns. The code for the enum_export.py program is shown in *Figure 1*. By validating command-line arguments and utilizing the os module to traverse through files in the specified folder, the script opens each file as a PE file using the pefile module for analysis. It applies two heuristic rules: Rule 1 flags potential malware if three or more export functions share the same memory address, while Rule 2 identifies potential malware if three or more export functions share the same memory offset. Results from the analysis indicate potential malware detections along with the violated rules, as shown in *Figure 3,* enabling assessment of the heuristic analysis's effectiveness in identifying malicious behavior within PE files.

3. Discussion and Conclusion:

Generated by ChatGPT- In conclusion, the lab aimed to develop a heuristic malware detection system utilizing static analysis techniques applied to Portable Executable (PE) files. Through the implementation of heuristic rules within the provided Python script, enum_export.py, students endeavored to identify potential malware samples based on specific patterns observed within Export Address Tables (EATs). Overall, the lab largely satisfied its stated purpose by providing students with practical experience in malware analysis and heuristic rule-based detection. Throughout the experimentation process, we observed that the heuristic rules implemented in enum_export.py effectively identified potential malware samples based on the criteria of shared memory addresses and memory offsets within the EATs. This aligns with the theoretical understanding of malware behavior, which often exhibits repetitive patterns in

memory allocation and function call structures. The output generated by the script accurately highlighted potential malware detections along with the violated rules, allowing for a comprehensive assessment of the heuristic analysis's effectiveness. Additionally, the lab provided valuable insights into the complexities and challenges of malware detection. While the implemented heuristic rules proved effective in identifying certain types of malicious behavior, it is important to acknowledge that malware authors continually evolve their tactics to evade detection. Thus, static analysis techniques, while valuable, may not always suffice as standalone solutions for malware detection. Future iterations of this lab could explore complementary approaches, such as dynamic analysis and machine learning-based detection methods, to enhance detection capabilities further. In summary, the lab succeeded in providing students with a practical understanding of heuristic malware detection techniques and static analysis methods applied to PE files. By actively engaging in the experimentation process and analyzing the results, students gained valuable insights into the complexities of malware analysis and the importance of employing multiple detection strategies to combat evolving threats effectively.