Reflection-Week 06

Objective

The objective of this lab was to introduce and apply static malware analysis techniques using Python. The goal was to understand how to safely examine Windows executables without executing them, and to learn how to extract meaningful indicators of compromise (IOCs). Through this exercise, I learned how to compute file hashes, extract human-readable strings, inspect Portable Executable (PE) headers, and use YARA rules to identify simple behavioural characteristics within a file.

Procedure and Results

1. **Hash Calculation (IOCs)**
   I computed the MD5, SHA1, and SHA256 hashes of Procmon.exe using Python's hashlib library.
   This demonstrated how cryptographic hashes are used as unique identifiers to verify file integrity and share IOCs in threat intelligence systems. I observed that even a single byte change in a file completely alters its hash (the avalanche effect).

2. **String Extraction**
   Using the re module, I extracted printable ASCII strings of at least four characters. The results showed file paths, DLL names, and version information—all legitimate for a Sysinternals utility. I learned that in actual malware, this step often reveals suspicious URLs, IPs, or encoded data.

3. **PE Header Inspection**
   With the pefile library, I examined the executable's PE structure, including its Entry Point, Image Base, and Imported DLLs and API functions.
   The imports listed normal Windows APIs such as kernel32.dll, user32.dll, and advapi32.dll, confirming the file was safe. I also learned that analysts often flag APIs like CreateRemoteThread or VirtualAllocEx as suspicious because they can indicate process injection or shellcode execution.

4. **YARA Rule Analysis**
   I created a basic YARA rule to detect the string "http" within the binary. This exercise showed how simple string-based signatures can automate detection and classification. The rule did not trigger for Procmon.exe, which was expected for a legitimate tool.

5. **Integrated Workflow**
   Finally, I combined all steps—hashing, string extraction, PE inspection, IOC detection, and YARA matching—into one complete static triage workflow. This process closely mirrors how security analysts assess files before moving to dynamic or sandboxed analysis.

Discussion

This lab highlighted the importance of static analysis in the early stages of malware triage. I gained an appreciation for how analysts collect evidence without executing potentially harmful files. I also understood how automation in Python can streamline repetitive analysis tasks and

help build reliable detection pipelines. Through experimentation, I learned how legitimate executables differ from malware in terms of structure, imports, and embedded artefacts.

Reflection

Overall, this lab significantly improved my understanding of static malware analysis. I learned how to apply systematic methods to extract and interpret indicators from executables. Working within a Jupyter notebook made it easy to document and visualize results step by step.
The combination of cryptographic hashing, string analysis, PE inspection, and YARA rule creation provided me with a complete view of how malware analysts perform safe triage. It also reinforced the importance of attention to detail, structured reporting, and the use of automated scripts in modern cybersecurity workflows.
I now feel more confident in using Python for forensic scripting and in understanding how static analysis contributes to threat intelligence, incident response, and SOC detection engineering.