DFRWS EU 2025 Workshop
Practical Malware Analysis and Memory Forensics for Incident Response
Faculty of Information Technology, Brno University of Technology (Czech Republic)

Practical Malware Analysis: From Memory Forensics to Threat Attribution

In this lab, you will learn how to perform practical malware analysis using a comprehensive workflow, from memory forensics to binary triage and threat attribution. The focus is on analyzing binaries extracted from memory dumps or other data sources, such as disk images or test environments, to quickly assess their functionality and potential risk in an incident response context. You will use tools such as hash calculators, string extractors, and capability detection frameworks to examine one of the binaries extracted from an ALINA-infected memory dump, previously analyzed in the lab. This dump, captured from a Windows 7 system, offers a real-world scenario to explore how classification, combined with structured analysis techniques, facilitates and accelerates a deeper understanding, detection, and attribution of malicious activity.

1 Malware Analysis Methodology

When performing malware analysis, the goal is to understand what the malicious software does, how it operates, and how it can be detected or removed. To guide this process, the following key questions must be answered:

- *What malicious activity does the sample perform on the system?*
- *How is this activity triggered or initiated (i.e., what persistence mechanisms are used)?*
- *When and how did the malware infect the system?*
- *How can the infection be removed and prevented from recurring?*

To answer these questions, we follow a two-phase methodology: **static analysis** and **dynamic analysis**.

1.1 Static Analysis Phase

The static analysis phase (also known as *cold code* or *dead code* analysis in reverse engineering) involves examining the binary without executing it. This involves analyzing the file's contents at the byte level. A solid understanding of assembly language and reverse engineering is essential at this stage, as it provides the low-level foundation (similar to how mathematics supports physics), while malware analysis builds on those principles to interpret and understand the practical behavior of real-world threats. It can be divided into three main activities:

1.1.1 Binary Hashing (MD5/SHA1/SHA256)

A cryptographic hash of the binary (e.g., using MD5, SHA-1, or SHA-256) provides a unique fingerprint of the file. These hashes can be searched online against public malware databases to identify known samples and gather contextual information. Tools such as `HashTab` on Windows, `md5` and `sha` on macOS, or the `md5sum` and `sha1sum` commands on Unix-based systems, are commonly used for this task.

1.1.2 String Analysis

By extracting human-readable text strings from the binary, we can discover useful indicators such as URLs, IP addresses, file names, or registry keys. These strings typically come from values hardcoded in the original source code. If the binary is not obfuscated or packed, these strings will be clearly visible and may indicate the malware's functionality. You can use the `strings` tool (native on Unix systems and available for Windows on the Internet).

1.1.3 Import Function Analysis

Analyzing which Windows API functions the binary imports provides insight into its capabilities. Functions related to files, registry keys, processes, or network connections can reveal how the malware interacts with the system. Tools such as `CFF Explorer` or PE viewers help explore this information. For detailed descriptions of specific API functions, the Microsoft Developer Network (MSDN)[1] is a valuable resource.

Please Note

- Static analysis can be limited by code obfuscation, packing, or encryption.
- Static analysis shows all possible execution paths, many of which might never occur in a real-world run.
- Static analysis is valuable, but often incomplete and requires dynamic analysis to obtain a more complete picture [**?**].

1.2 Dynamic Analysis Phase

The dynamic analysis phase (also known as *hot code* or *live code* analysis in reverse engineering) involves running the malware in a controlled environment to observe its behavior in real time. This phase reveals how the binary interacts with its environment and provides practical insights into its functionality.

Dynamic analysis can be divided into two major areas:

1.2.1 Interaction with the Operating System

Analysts observe whether the malware creates, modifies, or deletes files (*file system activity*), manipulates registry keys (*persistence mechanisms*), or spawns additional processes (*process activity*). Each of these actions may signal installation routines, persistence mechanisms, or propagation strategies. For instance, if new executable files are dropped, those should be analyzed as separate malware samples.

[1] See `https://learn.microsoft.com/`.

1.2.2 Interaction with the Internet

Monitoring network activity helps identify command and control infrastructure. Analysts should determine whether the malware attempts to contact specific IP addresses or domain names, what data it sends, and whether it receives instructions in return. Such behavior confirms external communication, which is common in ransomware, spyware, or botnet agents, to name a few of examples.

Advanced dynamic analysis may also involve using a debugger to trace instruction execution, monitor CPU registers, inspect memory, and analyze return values from API calls. These techniques are typically used when behavioral analysis alone does not provide enough clarity.

Please Note

- This phase helps answer most of the key questions posed at the beginning, such as persistence, infection vectors, and potential damage of the malware.
- Unlike static analysis, dynamic analysis explores only one execution path, which depends on runtime conditions such as system environment, timing, and user interaction.
- Some malware samples include anti-analysis techniques (such as sandbox evasion, sleep delays, or virtualized environment checks) that may prevent their full behavior from being observed during dynamic analysis.

1.3 Hybrid Analysis: Combining Static and Dynamic Approaches

While static and dynamic analysis are traditionally presented as distinct phases, many modern tools and workflows adopt a hybrid analysis approach, combining the advantages of both techniques to provide a more complete understanding of malware behavior.

Static analysis offers a fast and secure way to inspect a binary's structure without executing it. It allows analysts to discover embedded strings, imported functions, and code structure, which can indicate capabilities such as persistence, networking, or encryption. However, this approach alone can miss important details when malware is packed, obfuscated, or relies on execution conditions to reveal its full behavior.

Dynamic analysis, on the other hand, observes what malware does during execution, such as modifying registry keys, communicating over the network, or creating/deleting additional files. While powerful, dynamic analysis is limited to a single execution path and can be evaded by malware that detects virtual environments or delays its actions.

Hybrid analysis tools, such as sandboxes (e.g., Any.Run, Hybrid Analysis) or frameworks that incorporate disassembly and execution traces (e.g., CAPA), address this gap. These tools correlate static artifacts (such as strings or imports) with real-time behavioral data (such as process creation or file system changes). As a result, hybrid analysis allows analysts to confirm whether statically inferred capabilities are actually used during runtime and to uncover behaviors that are only observable under certain execution conditions.

In practice, hybrid analysis improves accuracy, reduces blind spots, and improves efficiency, especially when classifying large numbers of samples or identifying subtle malware behaviors that static or dynamic methods alone might miss.

Figure 1 summarizes the high-level workflow followed throughout the malware analysis process in this lab, from memory extraction and analysis to binary extraction and reporting and threat attribution.

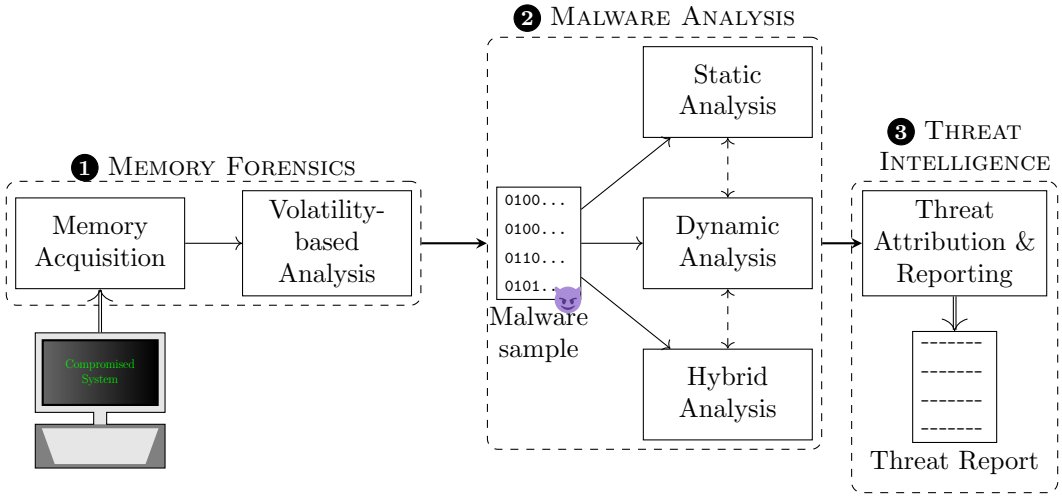

Figure 1: High-level workflow of the memory forensics, malware analysis, and threat intelligence process, combining static, dynamic, and hybrid analysis to support threat detection, attribution, and reporting.

☞ Summary (TL;DR)

- Malware analysis typically consists of two main phases: static analysis and dynamic analysis.
- Static analysis inspects the binary without executing it, focusing on aspects such as:
  - Hashes (MD5, SHA-1, SHA-256).
  - Embedded strings.
  - Imported functions and libraries.
- Dynamic analysis involves executing the malware in a controlled environment to observe actual behavior, including:
  - File system modifications.
  - Registry changes.
  - Process activity.
  - Network connections.
- Static analysis can reveal all possible execution paths, but may be limited by packaging or obfuscation.
- Dynamic analysis reveals actual runtime behavior, but only for the specific execution path taken.
- Hybrid analysis combines both approaches, allowing analysts to correlate static information with observed behavior. This combined approach increases accuracy, reduces blind spots, and facilitates better classification, detection, and attribution.

2 Threat Attribution and Reporting

After analyzing malware behavior using static and dynamic techniques, the next step is often to attribute the sample to a known malware family, threat actor, or campaign. Attribution can help contextualize the malware within broader threat intelligence initiatives, inform mitigation strategies, and contribute to knowledge sharing within the cybersecurity community.

Attribution is often based on comparing artifacts or behaviors observed during the analysis with known indicators associated with existing malware families. For instance, the use of specific mutexes, hard-coded domains, encryption algorithms, or unique file-naming conventions can serve as signatures that point to a specific campaign or actor. Comparing hashes, strings, or metadata from extracted files with databases such as VirusTotal (`https://www.virustotal.com/`), MalwareBazaar (`https://bazaar.abuse.ch/`), or Hybrid Analysis (`https://www.hybrid-analysis.com/`) can quickly identify whether the sample has been previously observed.

In addition to attribution, we should document the analysis results in a clear and structured manner. A good malware analysis report should include (i) basic sample properties (hashes, file size, compilation time), (ii) summary of its behavior (e.g., persistence, network communication, file and process activity), (iii) any indicators of compromise (IoC) identified, (iv) screenshots or tool excerpts (e.g., strings, API calls, registry changes), (v) MITRE ATT&CK mappings (if applicable), and (vi) recommendations for detection, containment, and eradication.

This reporting step not only helps internal response teams but also contributes to the broader cybersecurity community when shared responsibly. Well-structured reports are especially useful in incident response, threat hunting, and response team exercises. A standardized reporting structure improves consistency across cases, accelerates collaboration, and ensures that malware analysis results contribute meaningfully to tactical and strategic defenses.

The report should begin with an overview of the sample, including the file name, cryptographic hashes (MD5, SHA-1, SHA-256), file size, and the sample source (e.g., extracted from memory, a sandbox, or a disk image). This metadata uniquely identifies the binary and is necessary for future reference, correlation, and deduplication.

Next, provide a behavioral summary describing what the sample does during its execution. This includes observed file system changes, registry modifications, process startups, and network connections. If applicable, indicate whether the sample attempts to establish command and control communications or shows signs of data exfiltration or destructive behavior.

The report should also include a section on IoCs. These can include file paths, mutexes, registry keys, domains, IP addresses, and YARA signatures extracted from the sample. IoCs are especially valuable for threat detection and hunting in enterprise environments.

Where applicable, add a section on MITRE ATT&CK technique mappings, linking observed behaviors to the tactics and techniques defined in the ATT&CK framework. This helps standardize threat intelligence and facilitates integration with SIEM and SOAR platforms.

Include screenshots or tool excerpts to support key findings, especially if the behavior is subtle or context-dependent. Images can also help non-technical stakeholders understand critical aspects of the analysis.

Finally, conclude the report with recommendations. These can include proposed detection logic (e.g., YARA rules), indicators to monitor, mitigation measures (e.g., registry key removal, endpoint isolation), and suggestions for preventing similar infections in the future. If the malware shows links to a known family or actor, indicate this attribution and include links to threat intelligence sources.

☞ Summary (TL;DR)

- After analyzing a sample's behavior, attribution connects it to known malware families, threat actors, or campaigns using behavioral signatures and public threat intelligence databases.
- Common attribution indicators include mutex names, hardcoded domains, unique filename patterns, encryption schemes, and other persistent artifacts.
- Comparing extracted hashes, strings, and metadata with platforms such as VirusTotal, MalwareBazaar, or Hybrid Analysis can confirm known samples or identify related ones.
- Clear and structured reports are essential. A good analysis report should include sample metadata, behavioral findings, indicators of compromise, ATT&CK mappings (see § 3), and remediation recommendations.
- Well-documented reports support internal teams (e.g., detection engineering, IR, threat hunting) and, when shared responsibly, contribute to the broader cybersecurity community.

3 MITRE ATT&CK Behavioral Indicators and Mapping

As malware analysis increasingly integrates with threat intelligence and detection engineering, it is important to frame findings using structured knowledge bases such as the MITRE ATT&CK framework (`https://attack.mitre.org/`). MITRE ATT&CK is a globally recognized taxonomy that describes attackers' actual behavior (in terms of adversary tactics, techniques, and procedures) at the different stages of an intrusion.

Specifically, the MITRE ATT&CK framework organizes adversary behavior into a series of tactics, each representing a stage of the *attack lifecycle*. These stages describe the progression of an attack from the initial breach to the achievement of the adversary's objectives. The primary tactics of the ATT&CK business matrix, which reflect the typical stages of an intrusion, are:

1. *Initial access*: How the adversary gains access to a system or network (e.g., phishing, vulnerability exploitation).
2. *Execution*: How the adversary executes malicious code on a system (e.g., via PowerShell, scripts, or binaries).
3. *Persistence*: Techniques used to maintain access across reboots or user sessions (e.g., registry execution keys, scheduled tasks).
4. *Privilege escalation*: Gaining higher-level permissions or access (e.g., exploiting system vulnerabilities or bypassing User Account Control).
5. *Defense evasion*: Avoiding detection and analysis by disabling security tools, obfuscating code, or deleting logs.
6. *Credential access*: Capturing credentials through keylogging, credential dumping, or brute-force attacks.
7. *Discovery*: Identifying information about the system, network, or environment (e.g., enumerating users, services, or network topology).

8. *Lateral movement*: Moving between systems within the network to achieve additional objectives (e.g., via remote desktop or SMB).
9. *Collection*: Collecting data of interest from compromised systems (e.g., screenshots, documents, or keystrokes).
10. *Command and Control*: Establishing communication with external servers to issue commands or exfiltrate data.
11. *Exfiltration*: Stealing and transferring data outside the compromised environment (e.g., via HTTP, cloud storage, or FTP).
12. *Impact*: Manipulating, disrupting, or destroying data or systems to achieve the adversary's ultimate goal (e.g., ransomware, data deletion, defacement).

MITRE has expanded the attack lifecycle with pre-attack tactics to capture adversary behavior before initial access. These include *reconnaissance*, the first stage of the extended attack lifecycle, in which adversaries gather information about potential targets (such as vulnerabilities, exposed services, public infrastructure, employee data, or other weaknesses that can be exploited later), and *resource development*, where adversaries acquire and prepare the infrastructure and tools needed for later stages of the attack. This may involve registering domains for phishing or command and control purposes, developing or acquiring malware, exploits, or access credentials, or setting up compromised or rented cloud services, VPS servers, or fake identities, among other resources.

In the MITRE ATT&CK framework, a *tactic* represents a general objective or goal that an adversary attempts to achieve during an intrusion. Tactics are, in essence, the "why" of an attacker's actions, such as gaining initial access, maintaining persistence, executing code, or exfiltrating data. Each tactic corresponds to a phase of the attack lifecycle and serves as a broad category under which specific actions (techniques) are grouped.

A *technique* describes the "how," that is, the specific method an adversary uses to carry out a given tactic. Techniques provide more detailed insight into the attacker's actual behavior, such as using a malicious macro to gain initial access, creating a registry execution key for persistence, or using PowerShell to execute code. Techniques can vary widely in complexity and visibility, and many have sub-techniques that capture more granular variants of the same core behavior.

A tactic is the concrete implementation of a technique by a specific threat actor or malware family in a real-world attack. *Procedures* provide the most detailed level of information, describing exactly how a technique was executed: what tools or commands were used, what parameters were set, or how it was combined with others. While tactics and techniques are general and reusable in many cases, procedures capture the unique ways adversaries adapt those methods to specific situations.

Together, tactics, techniques, and procedures (commonly referred to as TTPs) form the backbone of behavioral threat intelligence and help analysts understand, detect, and respond to adversarial activity more effectively.

During static and dynamic analysis, specific behavioral indicators can be observed, such as registry modifications, process injections, or file encryption, that correspond to the techniques defined in MITRE ATT&CK. Linking observations to these techniques provides a better understanding of malware targets and helps standardize how threat intelligence is documented and shared.

For instance, if the malware creates an execution key in the registry to ensure its execution at startup, this can be assigned to "*T1547.001 – Registry Run Keys / Startup Folder*"[2], which

[2]See `https://attack.mitre.org/techniques/T1547/001/`.

is a subtechnique of "Boot or Logon Autostart Execution" that belongs to *Persistence* and *Privilege Escalation* tactics. Similarly, if the malware communicates with external domains for command and control, this may correspond to "*T1071 – Application Layer Protocol*"[3], specifically "*T1071.001 – Web Protocols*, while if it deletes snapshots to prevent recovery, it is assigned to "*T1490: Inhibit System Recovery*"[4].

Mapping observed behaviors to the MITRE ATT&CK framework allows analysts to contextualize malware activity within a standardized model of known adversary behavior. By assigning specific techniques used by a sample to ATT&CK, analysts can compare malware behavior to that of known threat actors or campaigns. This improves attribution accuracy and supports threat hunting initiatives. Additionally, behavior-based detection rules can be optimized by focusing on tactics and techniques rather than static signatures, increasing resilience against evasion and obfuscation. Using ATT&CK heatmaps helps visualize which parts of the attack lifecycle are covered by current defenses and where visibility gaps exist. Additionally, aligning findings with ATT&CK facilitates more seamless integration into SIEM, SOAR platforms, and detection engineering workflows, facilitating operational response and strategic decision-making.

☞ Summary (TL;DR)

- The MITRE ATT&CK framework provides a standardized way to describe and interpret adversary behavior using tactics, techniques, and procedures.
- Tactics represent adversary objectives (e.g., persistence, execution), techniques describe how those objectives are achieved, and procedures demonstrate actual implementations by specific actors.
- Mapping observed malware behavior (e.g., registry changes, C&C communication, snapshot deletion) to ATT&CK techniques provides structure and clarity during analysis. This process facilitates threat attribution, improves detection logic using behavior-based rules, and optimizes threat hunting capabilities.
- ATT&CK heatmaps help visualize defensive coverage throughout the attack lifecycle, and integration with SIEM, SOAR, and IR tools enables faster operational response.

4 Hands-On: ALINA Memory Extraction and Analysis

As the final part of this lab, you will apply the concepts and techniques learn throughout the course to a real-world scenario. Specifically, you'll analyze a memory dump of a system infected with the ALINA point-of-sale malware, a malware family known for extracting credit card data from system memory and exfiltrating it to a command and control server [**?**]. Your goal is to **identify and extract the ALINA malware binary from the memory dump and perform (basic) static analysis to gain preliminary insights into its behavior**. The memory dump is available on the workshop website[5].

To guide your analysis, you are assigned the following tasks:

Task 1.- Use Volatility 3 to identify suspicious processes in the memoy dump and extract the binary associated with the ALINA malware. Apply the skills developed in the previous lab session.

[3] See `https://attack.mitre.org/techniques/T1071/`.
[4] See `https://attack.mitre.org/techniques/T1490/`.
[5] See `https://webdiis.unizar.es/~ricardo/dfrws-eu-25-workshop`.

Task 2.- Perform basic static analysis using command-line tools or your preferred analysis environment, based on the techniques described in Section 1.

To structure your investigation and reinforce key concepts, the following guiding questions are provided. Each question is designed to be answered using basic static techniques, such as process metadata inspection, hash calculation, string extraction, and import function analysis, and will help you build an initial behavioral profile of the ALINA malware sample. Use your preferred tools to substantiate your findings and document your observations as if you were preparing an initial threat report.

Question 1.- *What is the name of the suspicious process where ALINA resides?*

Question 2.- *What are its PID and PPID?*

Question 3.- *At what time was the process created?*

Question 4.- *How was the suspicious process launched on the compromised system?*

Question 5.- *What Volatility command did you use to extract the process binary?*

Question 6.- *What is the size of the extracted file?*

Question 7.- *What is the MD5, SHA1, or SHA256 hash of the extracted binary?*

Question 8.- *Does the hash match any known ALINA samples on VirusTotal or other public threat intelligence repositories?*

Question 9.- *List any suspicious or revealing strings found in the binary.*

Question 10.- *Do you notice any references to HTTP URLs, domains, registry keys, or process names in the strings?*

Question 11.- *Does the malware create any mutex objects to prevent reinfection or coordinate execution?*

Question 12.- *What Windows API functions does the binary import?*

Question 13.- *Are there any functions related to networking (e.g.,* `WinHttpSendRequest`, `InternetOpen`*), memory reading (e.g.,* `ReadProcessMemory`*), or persistence (e.g.,* `RegSetValueA`*)?*

Question 14.- *What is the binary's build timestamp (if available)?*

Question 15.- *Does the PE header contain a legitimate company or product name?*

5 Additional Reading and Resources

To deepen your understanding of malware analysis and memory forensics, we recommend the following resources. These materials cover a variety of topics, from fundamental theory and practical techniques to real-world case studies and threat intelligence integration.

This list is not exhaustive, but it provides a solid foundation for further learning. Utilizing these resources will help you develop practical skills, stay up-to-date on new techniques, and connect with the broader digital forensics and malware investigation community. Enjoy the reading!

- "Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software", by Michael Sikorski and Andrew Honig. No Starch Press; 1st edition (February 1, 2012). ISBN: 978-1593272906
- "The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory", by Michael Hale Ligh, Andrew Case, Jamie Levy, and AAron Walters. John Wiley & Sons Inc (October 3, 2014). ISBN: 978-1118825099
- "Malware Analyst's Cookbook: Tools and Techniques for Fighting Malicious Code", by Michael Hale Ligh, Steven Adair, Blake Hartstein, and Matthew Richard. Wiley; 1st edition (November 2, 2010). ISBN: 978-0470613030
- "Malware Unicorn's Reverse Engineering 101", `https://malwareunicorn.org/workshops/re101.html`
- "OpenSecurityTraining.info", `https://opensecuritytraining.info`
- "Malpedia", `https://malpedia.caad.fkie.fraunhofer.de`

6 Solutions to Questions

Solution (turn to read)

Q1.- `ALINA_CJLXYJ.exe` (`windows.pslist` plugin).

Q2.- PID is 1828, PPID is 628 (`windows.pslist` plugin).

Q3.- 2019-09-21T12:07:04+00:00 (`windows.pslist` plugin).

Q4.- The binary file is located on the path `C:\Users\Usuario\AppData\Roaming\ALINA_CJLXYJ.exe` and it was launched with the argument `ALINA=C:\Users\Usuario\Desktop\ALINA_mod.exe` (`windows.pstree` plugin).

Q5.- `vol3 -f /shared/alinaiG.elf windows.pslist --pid 1828 --dump`

Q6.- 151552 bytes (148K)

Q7.-
- MD5: `269b76fea3686455220b8f85c5ca71d1`
- SHA1: `0faf587d6b6f26d2193f071764006d360fb60b62`
- SHA256: `dd71cc083b0f557c2ee2989450d06f5a6a6fe77f0d6e08eb3e292e50f8686c31`

Q8.- Yes, it is found on VirusTotal.

Q9.- There are some regex strings that match Track 1 and Track 2 magnetic strip card formats, according to ISO/IEC 7813. Other strings are related to third-party libraries embedded into the binary.

Q10.- There are references to IP addresses (`192.168.56.102`), HTTP POST verbs, registry keys (`Software\Microsoft\Windows\CurrentVersion\Run`), and a bunch of process names (`pidgin.exe`, `skype.exe`, `thunderbird.exe`, `devenv.exe`, `steam.exe`, `wininit.exe`, `crss.exe`, `smss.exe`, `svchost.exe`, `iexplore.exe`, `firefox.exe`, `chrome.exe`, `explorer.exe`).

Q11.- Yes, it creates a mutex with name `huahsdOsa90`, using `CreateMutexA` WinAPI. If the mutex already exists or cannot be created, the malware exits and stops infection.

Q12.- It statically imports 85 functions from `KERNEL32.dll` (e.g., `GetVolumeInformationA`, `Sleep`, `ReadProcessMemory`, etc.), 7 functions from `ADVAPI32.dll` (e.g., `AdjustTokenPrivileges`, `RegOpenKeyExA`, `LookupPrivilegeValueA`), 1 from `SHELL32.dll` (`SHGetSpecialFolderPathA`), 5 from `WININET.dll` (e.g., `HttpOpenRequestA`, `InternetConnectA`, `InternetCloseHandle`, etc.), and 1 from `SHLWAPI.dll` (`UrlCanonicalizeA`).

Q13.- Yes, there are functions related to networking (all functions from `WININET.dll`), memory reading (from `KERNEL32.dll`), and persistence (from `ADVAPI32.dll`).

Q14.- 2012-10-12 18:34:44.

Q15.- No, there is no metadata associated in the PE header.