Technical University of Moldova
Faculty of Computers, Informatics, and Microelectronics
Department of Software Engineering and Automation

MALICIOUS SOFTWARE AND INCIDENT RESPONSE

Practical work guide

Elaborated by: univ. assist., Catalin Mitu

Chisinau, 2026

Contents

Practical Work 1 - Windows executable analysis for malicious behavior identification

Malware analysis is the process of extracting information from malware through static and dynamic inspection by using different tools, techniques, and processes. It is a methodical approach to uncovering a malware's main directive by extracting as much data from malware as possible while it is at rest and in motion. The end goal is to create a solution for the malware to prevent it from spreading, detect its presence, and remediate, if possible, the malware infection.

Malware repository sources

- MalwareBazaar - malware samples sharing platform;
- theZoo - a live malware repository.

Analytical toolkit

Core

- pe-bear - multi-platform reversing tool for PE files;
- Process Monitor - advanced Windows activity monitoring tool;
- System Informer - multi-purpose tool for monitoring system resources;
- Wireshark - network analysis tool;
- Autoruns - shows what programs are configured to run during system boot or login.

Optional

- Ghidra - software reverse engineering (SRE) framework;
- ILSpy - .NET assembly browser and decompiler.

Technical Workflow

1. **Environment provisioning**:
   - Follow the Analysis Environment Configuration guide to set up the environment;
   - Install the core analysis tools (pe-bear, system informer, autoruns and procmon on windows; wireshark on linux) and learn how to use them against benign programs.
2. **State management**:
   - Take a cold snapshot named `Clean state. Tools installed`;
   - From the windows VM chose the program you want to analyze, be it malware or benign, and download it.

> ⚠
> Taking a snapshot before analyzing the malware is mandatory. We should have a clean base to revert to after the malware was run on the system.

3. **Static triage**:

Before performing the static analysis, read about what a PE file is and what binary embedded strings could be found in a file.
Using PE-Bear you can obtain:

- Its hashes and run the check in VirusTotal;
- Binary embedded strings;
- Imported WinAPI functions.

Using the values above one can predict the behavior of the sample.
Note: If the program is CLI (Common Language Infrastructure), one can try to use ILSpy to reconstruct its original code, as the bytecode is easier to revert than machine code.

4. **Runtime detonation**:

- Initialize monitoring tools: System Informer, ProcMon and Wireshark (Listen the *Internal Network* interface);
- Execute the sample. Meanwhile, analyze the actions from a simple user perspective, like: opening some apps, deleting itself from the original folder, adding new files/shortcuts on desktop;
- Process suspend. After some time, suspend its process tree using System Informer. Stop and save the captured events by ProcMon and the capture in Wireshark.

Suspending the main process tree does not mean the malware was 100% stopped, the machine may still be dangerous, as the malware could have injected code in to other processes or use other common persistence methods.

There are many malware analysis articles online, it's fine to learn from them, but if the final report has results from tools any other than the ones recommended here, it may be considered as cheating!

Evaluation Criteria

Mandatory Documentation Policy. The grade is based exclusively on the submitted report. All **relevant** technical findings, screenshots, and analysis must be documented in detail. Live demonstrations from virtual machines will not be accepted as a substitute for written evidence.

Grade 5. Environment setup and initial static analysis:

- Analysis environment is set;
- Program type is determined: native PE with machine code or a .NET one. If .NET, ILspy is used to analyze the actual code;
- Found strings are categorized;
- Based on the set of imported WinAPI functions and strings, infer the probable behavior of the program (e.g. file system interaction, registry usage, process creation, network communication). The WinAPI functions serve as evidence, not as the main subject of description.

Grade 6. Basic dynamic analysis with ProcMon:

- Recreate the process tree using ProcMon's filters (built in *Process Tree* feature might be at hand, but not as the main evidence). Use the filter function to prove the analyzed program spawns or not child processes. If yes, filter recursively all the initiated child processes. Document the full execution path and file system locations for all spawned processes.

Grade 7. Common persistence discovery:

- Use Autoruns to find if the analyzed program registered itself in a common auto-run places. Describe the used method. (Use the built in autoruns feature to save current state to a file and compare);
- Check for newly created users;
- Use ProcMon to check for shortcuts hijacking (.lnk files).

Grade 8. Network traffic analysis:

- Identify Domain Name System resolutions. List the suspicious domains and the resolved IP addresses;
- Identify the geographical location or ISP of the remote servers using IP reputation tools (whois, VirusTotal);
- Document all destination IP addresses, ports, and protocols and classify as *small heartbeats* or *large data transfer*;
- For unencrypted traffic, extract and explain the content;
- For encrypted traffic, extract the certificate details;

To match the wireshark results with the analyzed program, one can use ProcMon network events.

Grade 9. Comprehensive behavioral analysis:

- Conduct a full-spectrum behavioral analysis by correlating system-wide artifacts, including: filesystem modifications, registry integrity changes, process-level telemetry via ProcMon, with live runtime telemetry from System Informer, specifically targeting memory-resident strings and active network sockets.

Grade 10:

- The student explains the underlying logic in its own words.

Practical Work 2 - Practical adversarial tactics: a MITRE ATT&CK-based project

Introduction to Malicious Logic

In the context of offensive security, malicious software (malware) is defined as any code or application specifically engineered to compromise the Confidentiality, Integrity, or Availability (CIA) of an information system. Unlike benign software, malware is characterized by its adversarial intent - the execution of unauthorized operations to fulfill a specific objective without the consent of the system owner.

Malware serves as a primary tool for threat actors to establish a foothold within a target environment, facilitating operations ranging from long-term espionage to immediate financial extortion.

Adversarial Objectives and Strategic Utility

Modern malware development is not merely an exercise in disruption; it is a goal-oriented process designed to provide utility or value to the operator. These objectives are systematically categorized within the MITRE ATT&CK Matrix, which maps the lifecycle of an intrusion from Initial Access to Impact.

Common functional objectives include:

- **Data Exfiltration**: Identifying and extracting sensitive intellectual property, corporate secrets, or personally identifiable information;
- **System Manipulation**: Disrupting critical infrastructure or operational processes;
- **Resource Hijacking**: Leveraging the victim's hardware for unauthorized activities such as cryptocurrency mining, proxy relaying, or participation in a *botnet* for *Distributed Denial of Service* (DDoS) campaigns;
- **Cryptographic Extortion**: Rendering data inaccessible via encryption to demand financial ransom (Ransomware);
- **Information Gathering**: Establishing long-term surveillance through keylogging, screen scraping, or audio/video monitoring.

Economic Drivers of Malware Development

The "value" provided by malware is often realized through underground economies. Threat actors monetize their efforts through several distinct vectors:

- **Direct Monetization**: Exploiting financial credentials or demanding crypto-currency ransoms;
- **Information Brokerage**: Selling exfiltrated datasets or "initial access" credentials on dark web marketplaces;
- **Infrastructural Abuse**: Renting out "zombie" networks (Botnets) to other adversaries for illicit operations.

Technical Workflow: Adversarial Implementation

The Objective

Students are required to develop a functional piece of malicious logic. This program must go beyond simple "trollware" or annoyance-based scripts; it must demonstrate tactical utility and provide a clear "value proposition" for a hypothetical attacker. The implementation must align with specific MITRE ATT&CK Tactics,

ensuring that each feature developed can be mapped back to a recognized adversarial technique. The project must prioritize the Impact and Exfiltration stages of an attack, proving that the software achieves a tangible result for the creator.

Development Requirements

To ensure a realistic approach to malware development and system-level interaction, students must adhere to the following architectural requirements:

- **Core Adversarial Agent**: The primary executable must be developed using **C++** for its superior resource control and obfuscation potential. This language is well-supported, extensively documented, and provides a highly accessible implementation path for this task;
- **Auxiliary Scripting**: Scripting languages such as PowerShell, Bash, or Batch are permitted exclusively for supplementary tasks;
- **Infrastructure & C2 Server**: Higher-level interpreted languages are allowed to develop the Command and Control (C2) server.

Command and Control (C2) & Exfiltration

The C2 infrastructure acts as the centralized "brain" for the agent, facilitating bidirectional communication for remote tasking and exfiltration.

- **Command and Control (TA0011)**: Implement a Beaconing mechanism where the agent checks in to receive instructions. Use Application Layer Protocols (e.g., HTTP/S) to blend with legitimate traffic.
- **Exfiltration (TA0010)**: Develop a method to move stolen data from the target to your server.

Persistence mechanisms

Persistence consists of techniques that adversaries use to keep access to systems across restarts, changed credentials, and other interruptions that could cut off their access. Techniques used for persistence include any access, action, or configuration changes that let them maintain their foothold on systems, such as replacing or hijacking legitimate code or adding startup code.

Refer to the linked MITRE ATT&CK pages for technical implementation details:

- Boot or Logon Autostart Execution (T1547):
  - Registry Run Keys (T1547.001);
  - Shortcut Modification (T1547.009);
- Boot or Logon Initialization Scripts: Logon Script (T1037.001);
- Create or Modify System Process: Windows Service (T1543.003);
- Event Triggered Execution: File Association (T1546.001);
- Scheduled Task/Job (T1053.005);

Defence Evasion: Binary obfuscation

Defense Evasion focuses on techniques that make static analysis, like the work performed in Lab 1, difficult or impossible. By obfuscating static artifacts (strings, headers, and function names), an adversary forces the analyst to move to riskier dynamic analysis to understand the program's intent.

Evading Static Analysis In Lab 1, you used tools like PE-Bear to inspect the Import Address Table (IAT) and binary strings. To prevent an analyst from seeing your program's capabilities (e.g., networking or file manipulation), you must implement:

- **Dynamic API Resolution (T1027.007)**: Instead of having your WinAPI functions listed in the IAT, use LoadLibrary and GetProcAddress. This allows you to call functions without them appearing in the file header;
- **String Obfuscation (T1027)**: Encrypt or encode sensitive string. They should only exist in cleartext in memory during runtime. You are encouraged to use established libraries like this C++ string obfuscator.

Dynamic Anti-Analysis & Environment Awareness

Dynamic anti-analysis techniques bridge Discovery (TA0007) and Defense Evasion (TA0005). While binary obfuscation protects the file at rest, these techniques allow the agent to sense its surroundings at runtime and decide whether to execute its malicious payload or remain "dormant" to avoid detection by security researchers.

Environmental Discovery Techniques Adversaries use environmental "tells" to identify if they are being analyzed in a laboratory or sandbox. You are required to implement one or more of the following techniques to identify if your agent is being analyzed:

- Virtualization/Sandbox Evasion (T1497.001);
  - Reference: Consult the VMAware Flag Table for specific indicators.
- Process Discovery (T1057): Enumerate active processes to find analysis tools like *procmon.exe* or *wireshark.exe*;

Documentation & submission policy

Technical documentation & mapping Every implemented tactic and procedure must be documented comprehensively. Each entry must include:

- MITRE ATT&CK Mapping: A direct link or reference to the specific MITRE ATT&CK technique/subtechnique being addressed;
- Evidence of Execution: Visual "Before and After" proof (e.g., system state changes, logs, or terminal output) to verify successful implementation.

Code submission

- **Report integration**: Relevant code snippets that implement the specific procedure should be included directly in the report. Please exclude auxiliary or boilerplate code to maintain focus on the core logic;
- **Full source code**: The complete project source code must be submitted as a supplementary archive alongside the formal report;
- **Build instructions**: All code submissions must include a README file detailing the environment requirements and step-by-step instructions for building and running the project.

Grading & evidence Evaluation is based exclusively on the submitted written report. To ensure full credit, all relevant technical findings, screenshots, and analytical insights must be documented in detail within the document.

Grade 5:

- The core malicious actions are implemented.

Grades 6-9 (A point for each feature implemented):

- **Command and Control (C2) & Exfiltration**: Implementation of a functional C2 channel to exfiltrate data to a remote server;
- **Persistence mechanisms**: The malware demonstrates the ability to survive system reboots;
- **Static defence evasion**: Critical strings, API function calls, and linked libraries (DLLs) must not be visible as plain text within the binary data;
- **Dynamic anti-analysis**: The implementation includes environment-aware protections, such as detecting virtual machines (VMs) or the presence of common analysis tools.

Grade 10:

- The student explains the underlying logic in its own words.

Practical Work 3 - Incident Response

Incident response (IR) is the process by which an organization handles a data breach or cyber attack. It is an effort to quickly identify an attack, minimize its effects, contain damage, and remediate the cause to reduce the risk of future incidents. NIST, SANS, and other leading security institutes offer several approaches to building a structured incident response process.

In this practical work, we will focus on the SANS PICERL model, which outlines six core steps for managing security incidents: Preparation, Identification, Containment, Eradication, Recovery, and Lessons Learned.

Workflow

Preparation Follow the Wazuh environment deployment steps to establish baseline security and ensure you have full visibility to detect incoming threats.

Identification (The Detection) Follow Simulating malicious activity steps to generate detectable telemetry.

To understand what happened in the system the following pages from Wazuh Dashboard will be used:

1. **Threat Intelligence > Threat Hunting**. This page displays **alerts** (events marked as level 3+) in an organized manner.
2. **Explore > Discover**. Displays the logs in a more raw format based on the selected index pattern. The default one, `wazuh-alerts-*` will display the same data as Threat Hunting one. To view the unclassified logs or with rule level under 3, one can select `wazuh-archive-*` (note: it's not a default pattern and should be configured manually).

In this practical work the focus will go from identifying the alert using **Threat Hunting** to creating a clearer picture of what happened around that event using the **Discover** page.

> Here you can find a quick reference to the Dashboard Query Language used in Wazuh.

Containment After an incident is identified, containment methods are determined and enacted. The goal is to advance to this stage as quickly as possible to limit the scope of the incident and preserve evidence for later analysis. It can include actions like:

- Isolate affected systems: Disconnect from the network;
- Modify access controls: Remove or limit user accounts;
- Secure file systems: Make disks read-only; unplug external drives;
- Monitor and log activities: Increase logging and set up intrusion detection;
- Preserve evidence: Create forensic images and document actions taken;
- Communicate with stakeholders: Inform key personnel and engage the incident response team;
- Assess impact: Analyze the extent of the breach;
- Plan remediation steps: Develop recovery strategies and preventive measures.

Eradication During and after containment, the full extent of an attack becomes visible. Once teams are aware of all affected systems and resources, they can begin ejecting attackers and eliminating malware from systems. This phase continues until all traces of the attack are removed.

Recovery In this phase, teams bring updated replacement systems online. Ideally, systems can be restored without data loss, but this isn't always possible. In the latter case, teams must determine when the last clean copy of data was created and restore from it. The recovery phase typically extends for a while, as it includes monitoring systems for some time after an incident to ensure attackers don't return.

Lessons Learned This phase is when your team reviews the steps taken during a response. Members should address what went well, what didn't, and make suggestions for future improvements. Any incomplete documentation should also be wrapped up in this phase.

Evaluation Criteria

> ⚠ **Mandatory Documentation Policy**. The grade is based exclusively on the submitted report. All **relevant** technical findings, screenshots, and analyses must be documented in detail. Live demonstrations from virtual machines will not be accepted as a substitute for written evidence.

Grade 5. Environment preparation:

- Windows VM is managed by Wazuh via an agent;
- Sysmon logs are ingested by Wazuh Agent and sent to the server;
- All events, including non-alert ones, are available to be analyzed from the Wazuh dashboard.

Grade 6. Identification & timeline reconstruction:
Don't just find the alert, find the origin. Use the "Discover" tab to create a mini-timeline of the attack. Provide:

- the specific Wazuh alert rule ID;
- timestamp of the first execution;
- user;
- PIDs, PPIDs, process image paths;
- reasons why this is a malicious behavior and not a normal one.

Grade 7. Containment & evidence preservation:

- Network isolation: The VM should talk only with the Wazuh server;
- Process freeze: Suspend the malicious processes, if running;
- Preservation: Identify and move the malicious binaries to password-protected archives (password: infected).

Grade 8. Forensic discovery & eradication:

- Use Wazuh logs to find every footprint: affected registry keys or files, network connections, scheduled tasks, created users, etc;
- Revert the system to pre "infection" state (delete files, users, registry keys/values, etc.).

Grade 9. Hardening & recovery:

- System Enforcing:

  Implement simple methods that would prevent this specific Atomic test from running again (try to run the test again to see if it does the same), like:

  - Work users with limited privileges;

  - Group Policy Objects configurations;
  - AppLocker restrictions (tip: check Application Identity service).

  Why did the attack succeed initially? How does the "Hardening Tweak" stop the attacker?

- Return the machine to normal operations.

Grade 10:

- The student can explain the "Big Picture" in his own words.

Practical Work 4 - Detection Engineering & Active Threat Hunting

This project shifts the focus from reactive response to proactive defense. Students will learn to create detection logic for the threats they previously built.

Detection Engineering - The systematic process of creating, testing, and tuning detection rules based on adversary TTPs. This bridges the gap between offensive techniques and defensive monitoring.

Workflow

> ⚠
> All detections will be built for the adversarial program developed during practical work 2.

Automated Analysis

The goal here is to raise an alert in wazuh dashboard when the malware is written to `Downloads` folder.

The process is as follows:

1. The file is written in the monitored folder;
2. Wazuh agent sends event of file write/modify to Wazuh server;
3. Wazuh server sends back a task to run a YARA scan over that file;
4. Wazuh agent runs the scan script. The result is sent back to Wazuh server;
5. Wazuh server decides to raise an alert.

1. YARA Rule definition

Create a YARA rule for detecting the program written during practical work 2.

One can look for:

- `LoadLibrary` and `GetProcAddress` imported and less that 3-4 imports in total;
- Unobfuscated strings, like: C2 hostnames/IPs, registry paths, file system paths;
- Untreated PDB file destination.

2. Real-time file monitoring

Follow the documentation steps to set up Real-time monitoring over Downloads folder to log every file change.

The events could be monitored here: `Endpoint Security` > `File Integrity Monitoring` > `Events`. `rule.id: 554` for file added and `rule.id: 550` for file changed.

3. Wazuh Active Response

In order to trigger an action as a response for an event, one should set up a custom Active Response.

Relevant documentation pages:

- Custom active response script;
- File integrity monitoring and YARA.

1. Copy the yara executables to `C:\Program Files (x86)\ossec-agent\active-response\bin\yara\` and the rule to `bin\yara\rules\`.

2. Create `yara.py` and `yara.bat` scripts inside `active-response\bin\` folder and place the following content inside the batch script:

```
@echo off
<path-to-python.exe> "%~dp0yara.py" >> C:\Users\Public\debug.txt 2>&1
exit /b
```

3. Update the wazuh-manager config file `/var/ossec/etc/ossec.conf`.

Add a new command under Active Response section:

```
<command>
  <name>win-yara</name>
  <executable>yara.bat</executable>
</command>
```

Add an active response for add (554) and update (550) events.

```
<active-response>
  <command>win-yara</command>
  <location>local</location>
  <rules_id>550,554</rules_id>
</active-response>
```

4. Local YARA scan script

When wazuh-agent runs the active response script, it provides the event details in form of JSON to STDIN.

It should read the line (from STDIN, which contains the JSON), extract the relevant data, in our case the path to the created/modified file (`parameters.alert.syscheck.path`), and scan the file against our custom yara rule. The **positive** result should be written to the default active response log located at `C:\Program Files (x86)\ossec-agent\active-response\active-response.log`. The log format should be: `wazuh-yara-pmri: <yara-match-result>`, where `<yara-match-result>` is the yara scan output when a file matched a rule, which is by default in the format `rule-name file-name`.

> If yara found no match, the log should **not** be written to the active response log file.

5. Interpreting the active script results

In order to interpret those logs and raise and alert, wazuh-manager should use a custom decoder.

The following content will be placed inside `/var/ossec/etc/decoders/local_decoder.xml`.

```
<decoder name="yara_decoder">
  <prematch>^wazuh-yara-pmri:</prematch>
</decoder>

<decoder name="yara_decoder_extractor">
  <parent>yara_decoder</parent>
  <regex>^wazuh-yara-pmri: (\w+) (\.+)$</regex>
```

```
  <order>yara_rule,yara_scanned_file</order>
</decoder>
```

6. Raising an alert on a YARA match

Alerts in Wazuh are generated based on a matching rule with a level 3+.

We need to create a rule to assign a level to the matched Wazuh log.

Add the following rule to `/var/ossec/etc/rules/local_rules.xml` file.

```
<group name="yara,">
  <rule id="108000" level="10">
    <decoded_as>yara_decoder</decoded_as>
    <description>File "$(yara_scanned_file)" is a positive match.
    Yara rule "$(yara_rule)".</description>
  </rule>
</group>
```

In order to test the decoder and rule pipeline, one can use the built in testing tool: `wazuh-logtest`

Automated Remediation

This section outlines the transition from threat detection to immediate containment.

For running a quarantine script one should follow the exact same steps done previously to run the detection script.

1. Prepare the quarantine scripts (`quarantine.cmd` and `quarantine.py`) inside active-response bin folder (`C:\Program Files (x86)\ossec-agent\active-response\bin`);

The script should:

- Copy the malicious file inside `C:\ProgramData\PMRI-Quarantine` in a password (which is: "infected") protected archive.
- In case of success, log the result to `active-response.log` in the format: `wazuh-quarantine-pmri: "<the-original-file-path>" "<the-quarantine-path>"`;

This time, the file path to quarantine will be found at `parameters alert data yara_scanned_file` JSON path.

2. Define the script inside a wazuh command;

3. Define the active response action to trigger the command/script when a YARA match alert is raised (rule id: `108000`);

4. Create a custom decoder for the script response;

5. Create an alert based on the active response successful execution log.

> To test the remediation process, one can write the YARA match log directly to the `active-response.log` file or use the agent_control tool.

Behavior-Based Monitoring

To ensure granular telemetry, update Sysmon configuration with this one (update with `.\sysmon.exe -c sysmonconfig.xml`).

Execute the malware. If anti-VM logic is detected, apply minimal patches to ignore the check and trigger the core activity, but without compromising the VM detection process. The objective is to maximize telemetry ingestion in Wazuh.

Navigate to the Wazuh Discover tab. Set an absolute timeframe and apply filters to select the malicious artifacts (start with data.win.eventdata.image). Identify two-three unique events, like: process creation, network connection, image loaded, file create, registry event (IDs 12-14) or Service installed;

Use the templates described in the SIGMA section to create a SIGMA rule for each identified event, and then link them using Sigma Correlation to create a behavioral detection chain.

Examples:

- Program writes itself to registry Run key (Event ID 12-14);
- Program copied its exe to another folder (ID 11), launched a child from it (ID 1), stops itself and then the child deletes the parent from the original location (ID 26);
- Program accessed a specific server multiple times in a small period of time (ID 3);
- A file is constantly created and deleted, optionally accessing a server between the events (might indicate the captured data is stored in a file, send to the server, then the file is deleted);
- Constant file creation and deletion in a small amount of time. File creation has the same extension (e.g.: .locked, .encrypted) and deleted files are, for example: .png, .pdf, .docx, etc.

Evaluation Criteria

Grades 5-9:

- The YARA rule identifies only the malware written in practical work 2;
- Automatic YARA scan is set up and scans the files inside Downloads folder. When the dropped file is the one from lab 2, it raises an alert on wazuh dashboard;
- The file matched by YARA rule is automatically quarantined;
- SIGMA rules are created for each relevant TTP identified;
- SIGMA rule correlation is created.

Grade 10: The student can explain the whole process in its own words.

Technical Reference

Portable Executable file type

- Portable Executable - clean and easy explication of the PE file type;
- 101 Editor (Optional) - hex viewer and editor with built in PE structure detection and navigation, making it much easier to learn and explore the PE file type.

Binary embedded strings

The most popular strings found in binaries fall into these categories:

1. The standard MS-DOS stub string
`!This program cannot be run in DOS mode.`
Almost every Windows executable (malicious or benign) contains this string. It is part of the MS-DOS stub. If you see this string repeatedly or in unexpected places (like inside another file), it might indicate dropping (malware carrying another piece of malware inside itself);

2. Dynamic linking & import functions
Any program must interact with the operating system to do some meaningful work. They use functions defined in dynamic libraries provided by the OS, like: `kernel32.dll` (handles memory management and process creation), `user32.dll` (user interface functions), `libc.so` (the GNU C Library), `libSystem.dylib` (bundles other macOS necessary libraries). Seeing these names as strings reveals the program's capabilities. Common function names being:

- ReadFile / WriteFile - Reads or writes data from the specified file or input/output (I/O) device;
- LoadLibrary - Dynamically loads the specified module into the address space of the calling process;
- GetProcAddress - Retrieves the address of an exported function or variable from the specified dynamic-link library.

More info could be found on official windows documentation. To open the documentation for a specific function, struct or constant, one can perform a simple search with a search engine.
Another fast way is downloading the PDF and perform a find by name to get the url of the respective symbol.

3. Network indicators

- User Agents: `Mozilla/5.0...`
- IP Format Strings: `%d.%d.%d.%d` (A format string used by the code to construct IP addresses dynamically).
- Protocols: `http://`, `https://`, `ftp://`.
- Specific Domains: `some-name.com`.

4. Registry keys & file paths

- Registry Keys: `Software\Microsoft\Windows\CurrentVersion\RunOnce`, `Software\Microsoft\Windows\CurrentVersion\Run` (The most common "autorun" location);
- File paths: `C:\Windows\System32\`, `\AppData\Local\Temp\`;
- File names: `svchost.exe`, `explorer.exe`.

5. Scripting and obfuscation artifacts

- Command Line Tools: `cmd.exe /c` (Run a command and terminate), `powershell.exe -nop -w hidden -enc` (Run PowerShell silently with an encoded command);
- Obfuscation Alphabets: `ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/` (The standard Base64 string).

Process Monitor

Process Monitor is an advanced monitoring tool for Windows that shows real-time file system, Registry and process/thread activity. It combines the features of two legacy Sysinternals utilities, Filemon and Regmon, and adds an extensive list of enhancements including rich and non-destructive filtering, comprehensive event properties such as session IDs and user names, reliable process information, full thread stacks with integrated symbol support for each operation, simultaneous logging to a file, and much more.

Upon start, ProcMon starts instantly logging the system activity, by default including Registry, File, Network and Process & Thread activity. To toggle each of them, one should just click their respective button in the tool bar.

There is a compact explication of the most important option from tool bar:

- Open - opens an already saved capture in a PML (Process Monitor Lor) file;
- Save - saves the captured events into an PML file;
- Capture - toggle capturing the events;
- Autoscroll - Follow the latest events;
- Clear - clears all the logs;
- Filter - the most used feature, as the name implies, allows filtering the events by: Operation, Path, Category, etc;
- Highlight - simmilar with filter, but it does not remove the filtered out events, just highlights the selected ones in the list view;
- Process Tree - creates a process tree based on the events, just like System Informer does;
- Event properties / Double clicking on the event itself - displays extended detailes about the event;
- Find - general event find;
- Jump to Object - if the event is a registry or file one, it opens the path in regedit and explorer, respectively.

Wireshark

Wireshark is a network packet analyzer, it presents captured packet data in as much detail as possible.
The following are some of the many features Wireshark provides:

- Available for UNIX and Windows;
- Capture live packet data from a network interface;
- Open files containing packet data captured with tcpdump/WinDump, Wireshark, and many other packet capture programs;
- Display packets with very detailed protocol information;
- Export some or all packets in a number of capture file formats;
- Filter packets on many criteria;
- Search for packets on many criteria.

For more information, reffer to Wireshark User's Guide.

Filter documentation could be found by running `man wireshark-filters` in terminal or on the following web page: https://www.wireshark.org/docs/man-pages/wireshark-filter.html .

MITRE ATT&CK

MITRE ATT&CK (MITRE Adversarial Tactics, Techniques, and Common Knowledge) is a globally-accessible knowledge base of adversary tactics and techniques based on real-world observations. The ATT&CK knowledge base is used as a foundation for the development of specific threat models and methodologies in the private sector, in government, and in the cybersecurity product and service community.

ATT&CK describes behaviors across the adversary lifecycle, commonly known as tactics, techniques, and procedures (TTPs). In ATT&CK, these behaviors correspond to four increasingly granular levels:

1. **Tactics** represent the "why" of an ATT&CK technique or sub-technique. They are the adversary's technical goals, the reason for performing an action, and what they are trying to achieve. Each tactic contains an array of techniques that defenders have observed being used in the wild by threat actors.
2. **Techniques** represent "how" an adversary achieves a tactical goal by performing an action. Techniques may also represent what an adversary gains by performing an action. A technique is a specific behavior to achieve a goal and is often a single step in a string of activities intended to complete the adversary's overall mission.
3. **Sub-techniques** provide more granular descriptions of techniques;
4. **Procedures** represent "what" an adversary did and are instances of how an adversary has used a technique or sub-technique.

ATT&CK is organized in three "technology domains" - the ecosystem within which an adversary operates. The ATT&CK domains have matrices that reflect associated platforms (or systems) within each technology domain:

- MITRE ATT&CK - Enterprise:
  - Operating systems: Windows, Linux, and MacOS;
  - Cloud: Azure AD, Office 365, Google Workspace, Software-as-a-Service (SaaS), Infrastructure-as-a-Service (IaaS);
  - Network: Network infrastructure devices;
  - Containers: Container technologies;
  - PRE: Covering preparatory techniques, deprecating the previous PRE-ATT&CK domain;
- MITRE ATT&CK - Mobile: Provides a model of adversarial tactics and techniques to operate within the Android and iOS platforms. ATT&CK for Mobile also contains a separate matrix of network-based effects, which are techniques that an adversary can employ without access to the mobile device itself.
- MITRE ATT&CK - Industrial Control Systems (ICS): Focuses on tactics and techniques of adversaries whose primary goal is disrupting an industrial control process, including supervisory control and data acquisition (SCADA) systems, and other control system configurations.

Source: CISA Best Practices for MITRE ATT&CK Mapping.

Analysis Environment Configuration

The lab environment consists of two virtual machines. The primary machine is a Windows system where the malware is executed and observed. A secondary Linux-based machine acts as a network gateway, capturing and analyzing all traffic originating from the Windows system.

The following steps focus on Oracle VirtualBox virtualization software;


Don't install guest additions on the analysis machine.

1. Create the gateway machine

1. Download latest Debian net installer (amd64) ISO;
2. Configure the machine with at least 2GB of RAM and 15GB of disk. Make sure to Check `Skip Unattended Installation`. In network settings, enable only the first two adapters:
    1. One adaptor should be attached to `NAT`.
    2. The second one to `Internal Network`
        - Name: `pmri-internal-network`;
        - Adapter Type: leave the default one, *82540EM*;
        - Promiscuous Mode: *Deny*;
        - Check *Cable Connected*.
3. Follow the installation steps.
   Note: When asked to chose the desktop environment, XFCE is a decent and lightweight choice, instead of classic Gnome;
4. After the machine is fully installed, take a cold snapshot named `Clean installation`.

2. Create the main windows machine

1. Download the ISO file from the official website;
2. Create and configure a new virtual machine in VirtualBox. Make sure to Check `Skip Unattended Installation`, provide at least 4GB of ram and 64GB of disk storage (othervise, the installation will fail). In network settings, enable only the first adapter and attach to `NAT`;
3. Run and follow the installation process. When selecting the `Time and currency format`, chose `English (World)`. This is an easy hack to avoid some bloatware.
4. Select Windows 11 Pro;
5. When asked to configure the disks, just create a partition using the available free space and the installation tool will take care of the other required partitions;
6. After the installation finished and you are provided to chose between a *personal usage* or *set up for work or school*, chose the later option, select `Sign-in Options` and `Domain join instead`. This way we avoid using an online account;
7. After the machine is fully installed, take a cold snapshot named `Clean installation`.

3. Configure the machines to forward traffic from the Windows system through the gateway VM to external networks

1. Change the network adapter settings on windows machine from `NAT` to the `Internal Network` named `pmri-internal-network`, just like in the linux machine;

2. From the Windows VM settings set the manual *IPv4 address* as `192.168.88.2 255.255.255.0`, with *gateway* as `192.168.88.1` and *preffered DNS* as `8.8.8.8`;

3. Configure the gateway machine to forward the traffic from internal network NIC to the NAT one:
   If a command is not present, just install it with `sudo apt update && sudo apt install <package-name>`

    1. Check the names of the interfaces, which one is as NAT and which one is as Internal Network, using `nmcli` and comparing the returned MAC addresses with the ones from VirtualBox VM settings.

2. Configure the internal network inside the gateway machine:

```
sudo nmcli connection add \
  type ethernet \
  ifname <internal-network-interface-name> \
  con-name pmri-internal-connection \
  ipv4.method manual \
  ipv4.addresses 192.168.88.1/24 \
  ipv6.method disabled
```

enable kernel forwarding

```
sudo sysctl --write net.ipv4.ip_forward=1
```

make forwarding persistent

```
sudo sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
```

allow forwarding from internal to NAT network and back

```
sudo iptables --append FORWARD \
  --in-interface <internal-network-interface-name> \
  --out-interface <nat-interface-name> \
  --jump ACCEPT

sudo iptables --append FORWARD \
  --in-interface <nat-interface-name> \
  --out-interface <internal-network-interface-name> \
  --match state \
  --state ESTABLISHED,RELATED \
  --jump ACCEPT
```

make available packet forwarding to external networks

```
sudo iptables --table nat \
  --append POSTROUTING \
  --out-interface <nat-interface> \
  --jump MASQUERADE
```

install `iptables-persistent` to make the firewall changes persistent.
(To save the rules after the installation, run `sudo netfilter-persistent save`)

If any interface is not connected, use `sudo nmcli device connect <interface-name>`;

If after OS restart the NAT interface in not connected. Try to assign the interface name to that connection with: `sudo nmcli connection modify <connection-uuid> connection.interface-name <nat-interface-name>`

Wazuh Environment Deployment

Download the ova file and import it into VirtualBox. Before running the VM for the first time, perform the configuration steps below:

1. Wazuh Server configuration

- Create a new `NAT Network` (Tools > Network > NAT Networks) in VirtualBox with the name `pmri-nat-network`, IPv4 Prefix `192.168.88.0/24`, and DHCP enabled. This will allow the VMs to access each other and also the internet.
- Assign the Wazuh VM 2+ CPUs and at least 3.5GB of RAM;

  > If you are low on RAM and can't afford to run both VMs simultaneously, give at least 2.6GB to the Wazuh machine, set up the swap file for the difference, shut down the VM and allocate the size you can afford.

- Network settings: leave the default bridged adaptor and create the second one as `NAT Network` attached to the newly created `pmri-nat-network` network;
- Run the Wazuh VM, follow the steps to log in and check the assigned IP address with `ip a`. It should be part of `192.168.88.0/24` network;
- From the host OS, access the Wazuh Dashboard in a browser using the bridged adaptor address and credentials: `admin` & `admin` (The updated credentials could be found here).

> Before proceeding with the Windows VM configuration, make sure it is not in an infected state. Revert to a clean snapshot, if needed.

2. Decrease windows logs verbosity for User Account Management

Wazuh queries users and their group memberships to enrich security events with context (who did what, with what privileges). Windows logs that look up as Event ID 4798, which creates noise. It can create unwanted noise for this task, thus we will disable this policy using ~~a hack~~ this command:

```
auditpol /set /subcategory:"User Account Management" /success:disable
```

3. Wazuh-Agent installation

Follow the steps in the Wazuh Dashboard to install the Wazuh Agent in the Windows VM. Make sure to assign the newly created `NAT Network`.

> The Windows VM should be already configured with `192.168.88.2` IPv4 address. If you encounter connectivity issues, check the network settings.

4. Setting up advanced logging

Sysmon is a Windows system service and device driver that monitors and logs system activity to the Windows event log. It provides detailed information about process creations, network connections, and changes to file creation time.

- Deploy sysmon and use the SwiftOnSecurity configuration file.

```
.\Sysmon64.exe -accepteula -i sysmonconfig-export.xml
```

- Configure Wazuh agent to ingest the sysmon logs: Open a text editor as admin and open the agent configuration file: `C:\Program Files (x86)\ossec-agent\ossec.conf`.
  Add the configuration below at the end of the `Log analysis` group.

```
<localfile>
  <location>Microsoft-Windows-Sysmon/Operational</location>
  <log_format>eventchannel</log_format>
</localfile>
```

- Restart the Wazuh-Agent service.

Possible alerts could be seen in the `Threat Hunting` section in the dashboard.

5. Set up log archiving

By default, Wazuh indexes (saves) only alerts, which are events marked as level 3+. The events with level 0-2 are discarded. Simple, unclassified events are the ones that create the whole picture of an incident. Saving only the alerts is like having a photo of a thieft, while the whole video is lost.

In order to enable saving all events in Wazuh, follow these steps from the documentation.

To be able to display them in the dashboard, create their index pattern like here.

Restart the VM with: `sudo systemctl reboot`.

Simulating malicious activity

In order to simulate some malicious actions, we will use the Atomic Red Team library and their tool Invoke-AtomicRedTeam for an easier and unified execution of tests.

> Before proceeding with the installation, disable Microsoft Defender.

Follow the steps to install the execution framework and the atomics folder.

> If the installation fails using the default PowerShell 5.1, retry with version 7.

The "installation" is not persistent. To reenable the Invoke-AtomicTest command, execute:

```
Import-Module C:\AtomicRedTeam\invoke-atomicredteam\
```

Execute the code below to run one of the predefined tests in a deterministic random way.

```
$args = ("T1136.001", "4"), ("T1543.003", "2"),
  ("T1218.010", "3"), ("T1053.005", "7")
$i = [math]::Abs($env:COMPUTERNAME.GetHashCode() % 4)
Invoke-AtomicTest $args[$i][0] -TestNumbers $args[$i][1]
```

Swap file

```
SWAPFILE_PATH=~/.swapfile
SWAP_SIZE_GB=3
sudo fallocate -l ${SWAP_SIZE_GB}G $SWAPFILE_PATH
sudo chmod 0600 $SWAPFILE_PATH
sudo mkswap $SWAPFILE_PATH
sudo swapon $SWAPFILE_PATH
# check if enabled
free -h
```

Optionally, make it permanent. Write the following line to `/etc/fstab`

```
/home/wazuh-user/.swapfile    none    swap    sw    0    0
```

YARA

YARA is a tool aimed at (but not limited to) helping malware researchers to identify and classify malware samples. With YARA one can create descriptions of malware families (or whatever one wants to describe) based on textual or binary patterns. Each description, a.k.a. rule, consists of a set of strings and a boolean expression which determine its logic.

More information regarding how to write and use a yara rule can be found in official documentation.

One can get a precompiled binary from the github repo here.

Modules

Modules are the method YARA provides for extending its features. They allow defining data structures and functions which can be used in rules to express more complex conditions.

A useful module in our practical work is PE Module, which allows creating more fine-grained rules for PE files by using attributes and features of the PE file format. Here are some of the functions relevant for our practical work: `is_pe`, `imports`, `is_signed`, `pdb_path`, `import_details.number_of_functions`.

Rules examples

- Yara-Rules;
- Neo23x0.

Example command to test the rule against all built in windows files.

```
.\yara64.exe --recursive <own-yara-rule-file>.yara C:\Windows
```

Rule template:

```
import "pe"

rule <rule_name>
{
  meta:
    description = "Description"
    author = "Student Name"

  strings:
    $string_name = "value"

  condition:
    $string_name

}
```

SIGMA

Sigma is a generic and open signature format that allows describing relevant log events in a straightforward manner. The rule format is flexible, easy to write and applicable to any type of log file.

SIGMA Rule documentation.

To convert sigma rules, one can use the website https://sigconverter.io/.

Sigma Taxonomy - defines the field names and log sources that are allowed to be used in SIGMA rules shared on the official SigmaHQ repository.

Single SIGMA rule template:

```
title: <title>
name: <snake_case_name_to_use_as_reference_in_correlation_rule>
description: <description>
author: <student-name>
tags:
  - attack.<tactic-name> # example attack.exfiltration
  - attack.<technique-id> # example attack.T1041

# https://sigmahq.io/docs/basics/rules.html#logsources
logsource:
```

```
  product: windows
  # service: or category: . check sigma taxonomy above.

# https://sigmahq.io/docs/basics/rules.html#detection
detection:
  selection:
    # use the preferred detection method:
    # - keywords in raw log
    #   (https://sigmahq.io/docs/basics/rules.html#detection-keyword)
    # - one value under a specific fields
    #   (https://sigmahq.io/docs/basics/rules.html#detection-and)
    # - one of the multiple values in a specific field
    #   (https://sigmahq.io/docs/basics/rules.html#detection-or)

  # https://sigmahq.io/docs/meta/
  #   filters.html#design-of-inclusion-vs-exclusion
  filter:
    <specify-the-exclusions>

  condition: selection and not filter

falsepositives:
  - <Describe-possible-benign-triggers>
```

SIGMA Correlation rule template:

```
title: <title>
description: <description>
author: <student-name>

# https://sigmahq.io/docs/meta/correlations
correlation:
  # https://sigmahq.io/docs/meta/correlations#types-of-correlations
  type: temporal

  rules:
    - <rule_name_1>
    - <rule_name_2>
    # ...
    # - <rule_name_n>
  group-by:
    - <FieldName1>
    # ...
    # - <FieldNameN>
  timespan: 0m # 10s / 1m / 5m / etc.
```