Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Inginerie Software și Automatică

PROGRAME MALIȚIOASE ȘI RĂSPUNS LA INCIDENTE

Îndrumar pentru lucrări practice

Elaborat: asis. univ., Cătălin Mîțu

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 clasify 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 loaded modules, memory-resident strings, handle tables, and active network sockets.

Grade 10:

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

Practical Work 3 - Incident Response (Forthcoming)

Incident response (IR) is the process by which an organization handles a data breach or cyberattack. 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. (https://www.sans.org/media/score/504-incident-response-cycle.pdf)

Preparation

The first step is to review existing security measures and policies to determine effectiveness. This involves performing a risk assessment to determine what vulnerabilities currently exist and the priority of your assets. This information is then applied to prioritize responses and reconfigure systems so that high-priority assets are protected.
This phase is also where you write new policies and procedures and refine existing ones. These procedures include a communication plan and assignment of roles and responsibilities during an incident.

Identification (The Detection)

When an incident is detected, team members need to work to identify the nature of the attack, its source, and the goals of the attacker.
During identification, any evidence collected needs to be protected and retained for later in-depth analysis. Responders should document all steps taken and evidence found in detail. This can help more effectively prosecute if an attacker is identified.
Communication plans are also typically initiated at this phase, informing security members, stakeholders, authorities, legal counsel, and eventually users of the incident and what steps need to be taken.

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 minimize the amount of damage caused.
Containment is often accomplished in sub-phases:

- Short term containment: immediate threats are isolated in place. For example, the area of your network that an attacker is currently in may be segmented off. Or, a server that is infected may be taken offline and traffic redirected to a failover.
- Long term containment: additional access controls are applied to unaffected systems. Meanwhile, clean, patched versions of systems and resources are created and prepared for the recovery phase.

Eradication

During and after containment, the full extent of an attack is made 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. In some cases, this may require taking systems off-line so assets can be replaced with clean versions in recovery.

Recovery

In this phase, teams bring updated replacement systems online. Ideally, systems can be restored without loss of data 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 also includes monitoring systems for a while after an incident to ensure that attackers don't return.

Lessons Learned

The lessons learned phase is one in which your team reviews what steps were 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.

Source: cynet.com Incident Response.

Workflow

Follow the Wazuh environment deployment steps to set up the environment.

1. **Deploy sysmon** and use the SwiftOnSecurity configuration file.

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

2. **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 `Log analysis` group.

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

Restart the Wazuh-Agent service

The logs could be seen in the `Threat Hunting` section.

Practical Work 4 - Forthcoming

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 into VirtualBox. Before running the VM for the first try, perform the configuration steps bellow:

1. 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.
2. Assign the Wazuh VM 2+ CPUs;
3. Assign at least 3.5GB of RAM;

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

4. 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 inffected state. Revert to a clean snapshot, if needed.

Before installing the Wazuh Agent to Windows, it should be in the same network. It could be done by changing the only adaptor to the newly created `NAT Network`, as was done with Wazuh VM.

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

Follow the steps in Wazuh Dashboard to install the Wazuh Agent in Windows VM.

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
```