

An Integrated System for Automated Threat Detection and Response Using Wazuh, Suricata and Telegram

Arfa Nadia Mohamad Nizam¹, Nur Fadzilah Othman^{1*}, Syarulnaziah Anawar², Zakiah Ayop², Sofia Najwa Ramli³

¹Department of Cyber Security, Fakulti Kecerdasan Buatan dan Keselamatan Siber, Universiti Teknikal Malaysia Melaka, Hang Tuah Jaya, 76100 Durian Tunggal, Melaka, Malaysia,

²Center for Advanced Computing Technology, Fakulti Teknologi Maklumat dan Komunikasi, Universiti Teknikal Malaysia Melaka, Malaysia,

³Faculty of Computer Science and Information Technology, University Tun Hussein Onn Malaysia, Batu Pahat Johor, Malaysia

*Corresponding Author

DOI: <https://doi.org/10.47772/IJRISS.2026.100900237>

Received: 16 September 2026; Accepted: 21 September 2026; Published: 07 October 2026

ABSTRACT

The increasing speed and diversity of cyberattacks require security monitoring systems that progress from detection to timely and controlled containment. This study developed an integrated framework that combines Suricata network intrusion detection, Wazuh event correlation and Active Response, and Telegram notification. The framework was implemented in an isolated VirtualBox laboratory using a Kali Linux attacker at 192.168.218.128 and an Ubuntu security host at 192.168.218.132. Five controlled scenarios were evaluated: port scanning, Secure Shell (SSH) brute force, Synchronize (SYN) flood, Structured Query Language (SQL) injection, and reverse shell. A scenario passed only when the correct Wazuh detection, Telegram notification, independently verified automated response, timed removal, and recovery evidence were observed. All five scenario-level tests passed, giving a functional test pass rate of 5/5 under the tested configuration. The mean detection-to-response delay was 0.727 seconds, with a maximum of 2.000 seconds, while the mean recovery time was 61.770 seconds. The principal contribution is an auditable, attack-specific closed loop that links detection, correlation, notification, containment, rollback, and recovery rather than treating alert delivery as evidence of mitigation. Because each scenario was executed once and benign control traffic was not included, the result is proof-of-concept functional validation and not general detection accuracy, reliability, precision, or recall.

Keywords: Wazuh, Suricata, Telegram, Intrusion Detection System, Active Response, Security Orchestration Automation and Response, Automated Incident Response, Functional Testing

INTRODUCTION

Modern attacks can progress quickly after initial access, making delayed manual response a significant security concern. An Intrusion Detection System (IDS) identifies suspicious activity and generates an alert, while an Intrusion Prevention System (IPS) or response mechanism attempts to stop or contain that activity [1]. Current incident-response guidance emphasizes that detection, response, and recovery should operate as connected risk-management functions so that damage can be limited and service availability restored [2]. In practice, however, network detection, centralized analysis, administrator notification, mitigation, and recovery are often implemented as separate activities. The hand-off between these activities can increase the time between detection and action.

This study addresses the operational gap by integrating Suricata, Wazuh, and Telegram. Suricata inspects

network traffic and produces Extensible Event Format JavaScript Object Notation (EVE JSON) alerts. Wazuh functions as the Security Information and Event Management (SIEM) layer by collecting and correlating these events, displaying them on its dashboard, and invoking an attack-specific Active Response. Telegram uses an Application Programming Interface (API) to provide the administrator with the attack context and response status. The framework therefore implements an end-to-end path from network observation to notification, containment, timed removal, and recovery.

The objectives were to identify five relevant cyberattacks and suitable response mechanisms, develop the integrated framework, and evaluate its functional behaviour and timing in a controlled laboratory. The evaluation verifies whether each predefined scenario produces the required detection, notification, mitigation, and recovery evidence; it does not claim production-level detection accuracy.

Research Gap and Contribution

Existing documentation demonstrates that Suricata alerts can be ingested by Wazuh [10], and Wazuh supports custom response scripts triggered by specific rule identifiers, alert levels, or groups [11]. These capabilities establish technical compatibility, but integration alone does not demonstrate that a response was appropriate, executed successfully, removed safely, or followed by service recovery. Alert-only studies and remote notification implementations may therefore leave a gap between reported detection and verified containment.

The contribution of this study is a controlled, auditable response chain for five heterogeneous attacks. First, a single correlated event drives dashboard evidence, Telegram notification, and an attack-specific response. Second, the mitigation is selected according to traffic direction and service risk: source blocking for reconnaissance and authentication abuse, rate limiting for excessive SYN traffic, port-scoped web blocking for SQL injection, and session termination with outbound restriction for a reverse shell. Third, success requires independent evidence of changed attack behaviour and automatic rollback, not merely an alert or notification. Finally, protected-address exclusions, duplicate-rule checks, input validation, response logging, and temporary timeouts are incorporated to reduce the operational risk of automated containment. The work is positioned as a laboratory proof of concept rather than a production-ready Security Orchestration, Automation, and Response (SOAR) platform.

Related Work

Network Detection and Centralized Monitoring

Suricata is an open-source network intrusion detection and prevention engine that uses configurable rules to inspect packets, flows, and application-layer activity [3]. Its multithreaded design and EVE JSON output support integration with centralized monitoring platforms. Comparative research has shown that Suricata performance depends on traffic conditions, rule sets, processor allocation, and system configuration [4], [5]. Consequently, results obtained in a small virtual laboratory should not be generalized to enterprise traffic without additional validation.

Wazuh provides centralized log collection, event correlation, dashboard visualization, and Active Response capabilities [6], [9]. Official integration guidance shows how Wazuh can collect Suricata EVE JSON records from a monitored endpoint [10]. In the proposed framework, Suricata supplies network-level evidence while Wazuh applies project-specific correlation logic and controls response execution. This combination extends network detection with centralized decision-making and an auditable response trail.

Notification and Automation and SOAR

Telegram bots provide an API-based method for delivering incident information to an administrator without requiring continuous dashboard access [7]. Notification alone, however, does not prove that mitigation occurred. A response must be verified through firewall state, packet counters, session behaviour, application availability, or response logs. The present study therefore requires independent mitigation evidence before a scenario can be marked as passed.

SOAR systems coordinate security tools, automate repeatable actions, and execute response playbooks across incident phases. Kinyua and Awuah [12] describe SOAR as a mechanism for integrating otherwise separate tools and automating workflows that continue beyond SIEM alert generation into containment, eradication, and recovery. NIST similarly places Detect, Respond, and Recover within a connected incident-response model [2]. The current framework implements a small, rule-driven subset of SOAR principles: Suricata supplies the event, Wazuh performs correlation and orchestration, Telegram communicates status, and Active Response executes and later removes a temporary control. It does not provide enterprise case management, multi-tenant orchestration, approval workflows, threat-intelligence enrichment, or adaptive playbooks.

Standard IDS measures such as detection rate, precision, false-positive rate, and accuracy require labelled malicious and benign observations represented by true positives, true negatives, false positives, and false negatives [8]. The current experiment instead evaluates a small set of predefined system functions. For this reason, the reported percentage is a functional test pass rate and not a classification-accuracy measure.

Table 1. Comparison of relevant approaches and the proposed contribution

Approach	Primary role	Typical limitation	Role in this study
Suricata	Rule-based network detection	Coverage depends on rules, traffic, and configuration	Produces network security events
Wazuh	Central correlation and Active Response	Response safety requires careful tuning and rollback	Correlates events and invokes controls
Telegram	Remote incident notification	Does not independently prove mitigation	Reports attack and response status
SOAR	Orchestrated playbooks across security tools	Enterprise implementations require governance and broad integrations	Provides the conceptual closed loop model
Proposed integration	Detection, correlation, notification, containment, rollback, and recovery	Validated only as a controlled proof of concept	Verifies attack specific outcomes and recovery evidence

As summarized in Table 1, the novelty is not the isolated use of any single tool. It is the verified closed-loop workflow and attack-specific response selection built from those tools.

METHODOLOGY

An experimental methodology was used to implement and functionally test the framework. The work consisted of laboratory setup, component integration, attack-specific response development, controlled attack execution, evidence collection, and result analysis. The design prioritised traceability: each scenario had to provide a consistent sequence of timestamps and artefacts from detection through recovery.

Experimental Setup

The framework was deployed in VirtualBox on a Windows 11 host with an isolated virtual network. Kali Linux at 192.168.218.128 acted as the attacker. Ubuntu Server at 192.168.218.132 hosted the monitored services, Suricata, Wazuh components, the Telegram integration, Damn Vulnerable Web Application (DVWA), and the response scripts. Isolation reduced risk to external systems and allowed the same baseline configuration to be restored between scenarios. Table 2 summarizes the experimental testbed.

Table 2. Experimental testbed configuration

Component	Configuration	Function
Attacker	Kali Linux, 192.168.218.128	Runs Nmap, Hydra, Hping3, SQL input, and Netcat

Victim and monitor	Ubuntu Server, 192.168.218.132	Hosts security components and protected services
Detection	Suricata	Inspects traffic and writes EVE JSON alerts
Correlation and response	Wazuh	Correlates events and triggers response scripts
Notification	Telegram Bot API	Delivers attack and response status

Integrated System Workflow

Figure 1 illustrates the complete workflow. Suricata first inspects network traffic and records a matching alert in eve.json. Wazuh reads the JSON event, extracts the relevant source, destination, protocol, port, and signature fields, and applies a project-specific rule. The same correlated event is displayed on the dashboard, sent to Telegram, and passed to the relevant Active Response script. The script validates the target, applies the selected control, records the outcome, and removes the temporary rule after the configured timeout.

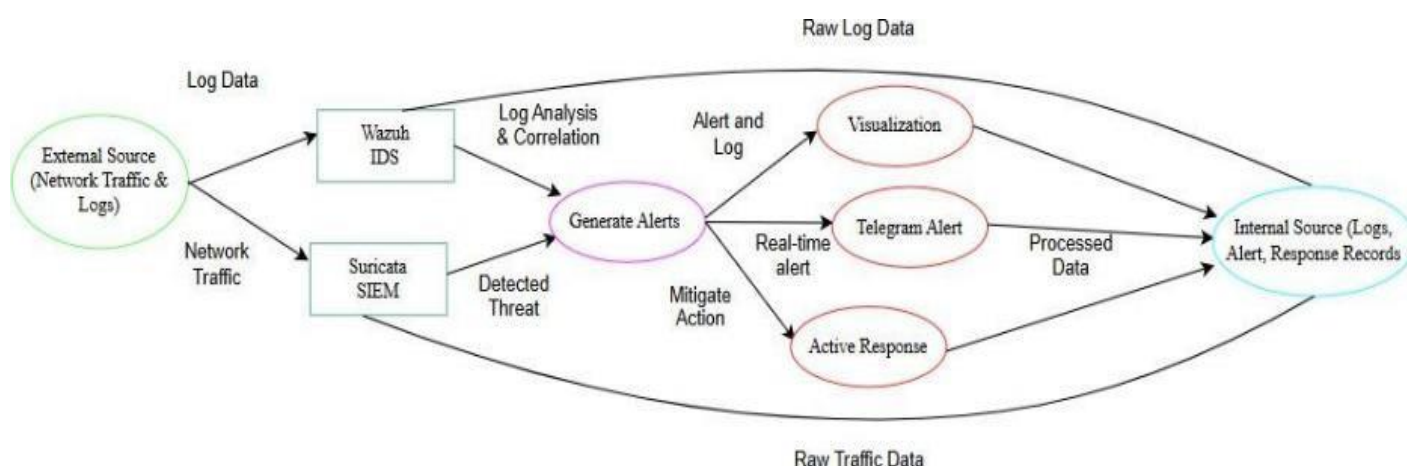


Figure 1. Integrated detection, notification, response, and recovery workflow.

Attack Scenarios and Response Design

Table 3 maps each authorised laboratory test to the corresponding response. The controls were deliberately scoped and temporary. Permanent or overly broad blocking could disrupt legitimate users, particularly when addresses are shared or a detection rule produces a false positive.

Table 3. Attack scenarios and implemented responses

Scenario	Test Tool or Target	Verified Response
Port scanning	Nmap	Temporary source-IP block
SSH brute force	Hydra on TCP 22	Temporary source-IP block
SYN flood	Hping3 on TCP 443	Source- and port-scoped SYN rate limiting
SQL injection	DVWA web request	Temporary source block on TCP 80 and 443
Reverse shell	Netcat on TCP 4444	Session interruption and outbound block

Reproducible Configuration and Data Flow

The implementation used a documented chain of configuration files and logs. Suricata was configured in /etc/suricata/suricata.yaml and project signatures were maintained in /var/lib/suricata/rules/custom.rules. Matching events were written to /var/log/suricata/eve.json.

Wazuh log collection read the JSON stream and stored matched alerts in /var/ossec/logs/alerts/alerts.json. Project-specific correlation rules were maintained in /var/ossec/etc/rules/local_rules.xml. The Telegram

integration was implemented as `/var/ossec/integrations/custom-telegram.py`, while the attack-specific handler was deployed as `/var/ossec/active-response/bin/fyp-response.py`. Exact bot tokens and chat identifiers are intentionally excluded because they are authentication secrets.

For traceability, the project-specific Wazuh detection rules were 100100 for port scanning, 100101 for SYN flood, 100102 for SSH brute force, 100103 for SQL injection, and 100104 for reverse shell. The custom response-success and removal events shown in the dashboard used 100110/100120 for port scanning, 100111/100121 for SYN flood, 100112/100122 for SQL injection, and 100113/100123 for reverse shell; the SSH firewall-drop lifecycle used the associated Wazuh Active Response events. The Suricata signature definitions, complete Wazuh XML, and complete Python scripts should be supplied with the final submission as supplementary configuration files so that thresholds, regular expressions, and credentials can be separated safely from the article body.

The Active Response handler received the correlated Wazuh alert as JSON through standard input, consistent with the documented Wazuh custom-script mechanism [11]. It extracted the rule context and validated the source or destination address before invoking the relevant firewall action. The handler logged the add and delete operations so that response execution and timed removal could be correlated with the original alert.

Table 4. Detection, response and verification mapping

Scenario	Detection trigger	Response scope	Independent verification
Port scanning	Suricata scan signature and Wazuh correlation	INPUT restriction for attacker source IP; 60-second timeout	Firewall state, changed scan result, removal log
SSH brute force	Repeated failed SSH activity correlated by Wazuh	INPUT restriction for attacker source on TCP 22; 60-second timeout	SSH timeout during control and successful access after removal
SYN flood	Excess TCP SYN activity targeting port 443	Source- and port-scoped SYN rate limit; 60-second timeout	DROP and ACCEPT counter increments plus removal event
SQL injection	Suricata web-attack signature in DVWA request	Source block on TCP 80/443; 60-second timeout	Web request timeout followed by restored access
Reverse shell	Suspicious callback/session on TCP 4444	Session termination and victim OUTPUT restriction; 60-second timeout	Reconnection timeout followed by successful postremoval test

Automated-Response Safety Controls

Automated containment can interrupt legitimate activity when a signature is too broad or an address is incorrectly classified. The response workflow therefore applied five safeguards: validation of the extracted Internet Protocol (IP) address and port; exclusion of protected or management addresses; duplicate-rule checks before insertion; attack-specific rather than global service controls; and automatic removal after approximately 60 seconds. Every add and delete action was logged. These controls reduce risk but do not eliminate it. A production deployment should add tested allow lists, change approval for high-impact actions, health checks, and an emergency rollback path.

Functionality Testing

Figure 2 summarizes the controlled test sequence used for every scenario. Each attack was executed independently after the environment returned to its documented baseline. A scenario received Pass status only when the correct Wazuh alert was recorded, a readable Telegram message was delivered, the intended response was independently verified, the temporary control was removed, and the baseline service state was restored.

Controlled Test Sequence for Each Attack Scenario

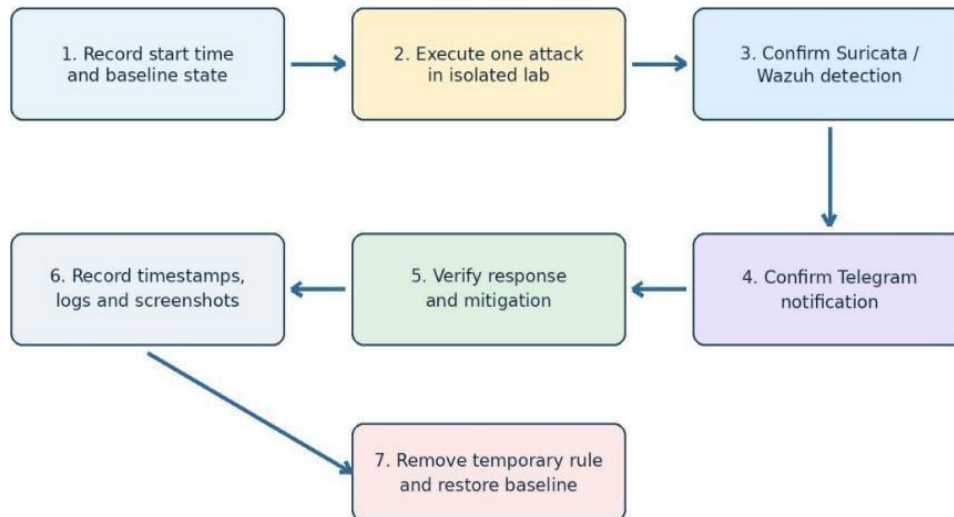


Figure 2. Controlled test sequence used for each attack scenario.

Functional Test Pass Rate (FTPR) = $(P / T) \times 100\%$

In the equation, P is the number of passed functional test cases and T is the total number of executed functional test cases. This calculation measures completion of predefined functions under the tested configuration. It does not measure production detection accuracy or the probability that unknown attacks will be detected.

Timing Methodology

Two intervals were derived from Wazuh and response-log timestamps. Detection-to-response delay was calculated as $t_{\text{response}} - t_{\text{detection}}$, where $t_{\text{detection}}$ is the timestamp of the correlated Wazuh detection and t_{response} is the timestamp confirming successful control application. Recovery time was calculated as $t_{\text{removal/recovery}} - t_{\text{response}}$, where the later timestamp represents the recorded removal of the temporary rule and confirmation that the baseline service was available again. Events were compared within the same laboratory host chronology to avoid cross-host clock-offset error. Millisecond values were retained where available.

Only one documented execution of each attack was available. Therefore, the reported mean is a descriptive average across five different scenarios, not a statistical estimate from repeated trials. Standard deviation, confidence intervals, run-to-run variability, reliability, false-positive rate, false-negative rate, precision, and recall were not calculated. These require repeated observations and benign or mixed-traffic controls.

RESULTS AND ANALYSIS

Functional Test Results

All five attack scenarios produced the required detection, notification, verified mitigation, timed removal, and recovery evidence. Accordingly, five of five scenario-level functional tests passed.

Table 5. Scenario-level functionality testing results

Scenario	Detection and Notification	Mitigation and Recovery	Status
Port scan	Correct Wazuh event and Telegram message	Scanning restricted; access restored after removal	Pass
SSH brute force	Correct Wazuh event and Telegram message	SSH timed out during block; authorized access restored	Pass
SYN flood	Correct Wazuh event and	Excess SYN traffic limited; service	Pass

	Telegram message	accessibility restored	
SQL injection	Correct Wazuh event and Telegram message	Web access blocked at network level; access restored	Pass
Reverse shell	Correct Wazuh event and Telegram message	Session interrupted; reconnection blocked, then restored	Pass

$$\text{FTPR} = (5 \text{ passed test cases} / 5 \text{ executed test cases}) \times 100\% = 100\%$$

The result means that every predefined scenario completed its expected functional path in the controlled laboratory. It does not mean that the framework has 100% detection accuracy against general network traffic or unseen attacks.

Port-Scanning Evidence

Figure 3 shows the Wazuh event sequence that connects port-scan detection to temporary blocking and later rule removal. Figure 4 shows the corresponding Telegram message. The response was not marked as successful from the message alone; the firewall state and changed scan outcome were also verified.

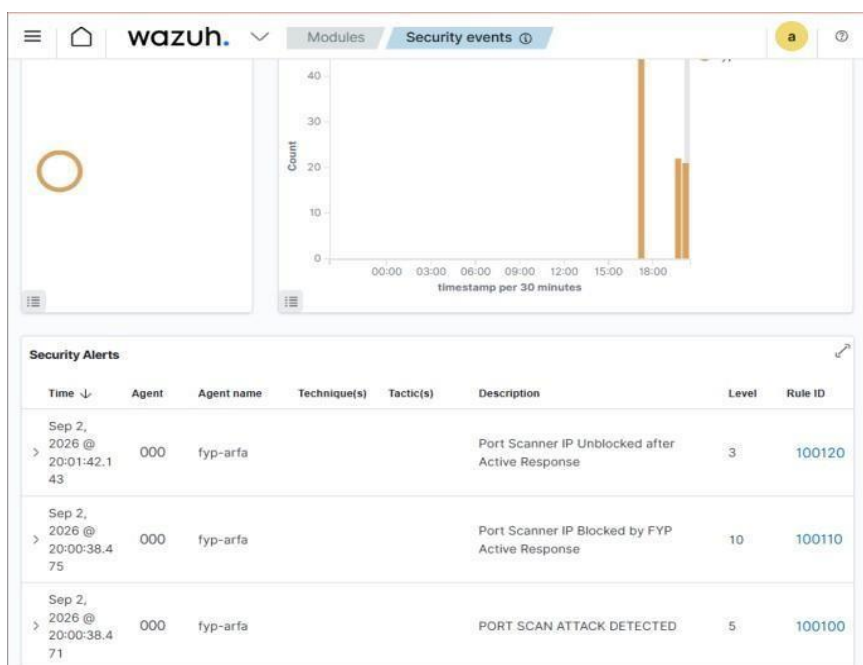


Figure 3. Wazuh event sequence showing port-scan detection, temporary blocking, and rule removal.



Figure 4. Telegram notification containing port-scan details and automated-response status.

SSH Brute-Force Evidence

Figure 5 presents the Telegram notification identifying the SSH brute-force source and temporary source-IP block. The independent SSH attempt in Figure 6 timed out while the firewall rule was active. Authorised access was verified again after automatic removal, confirming containment and recovery.

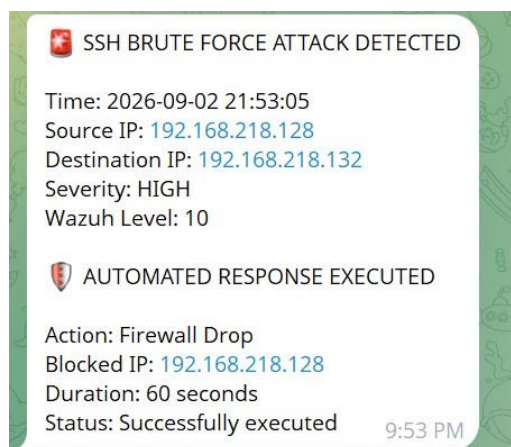


Figure 5. Telegram notification reporting the SSH brute-force source and temporary sourceIP block.

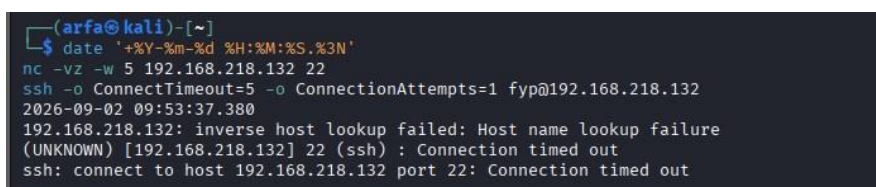


Figure 6. SSH connection timeout observed while the attacker address was blocked.

SYN-Flood Evidence

Figure 7 shows the Wazuh sequence for SYN-flood detection, rate limiting, and response removal. Figure 8 records the Telegram notification. Unlike a complete service block, the response installed a rate-limiting rule for the attack source and target port. Firewall counters showed that excessive SYN packets were dropped while a smaller number remained accepted, supporting the intended availability-preserving response.

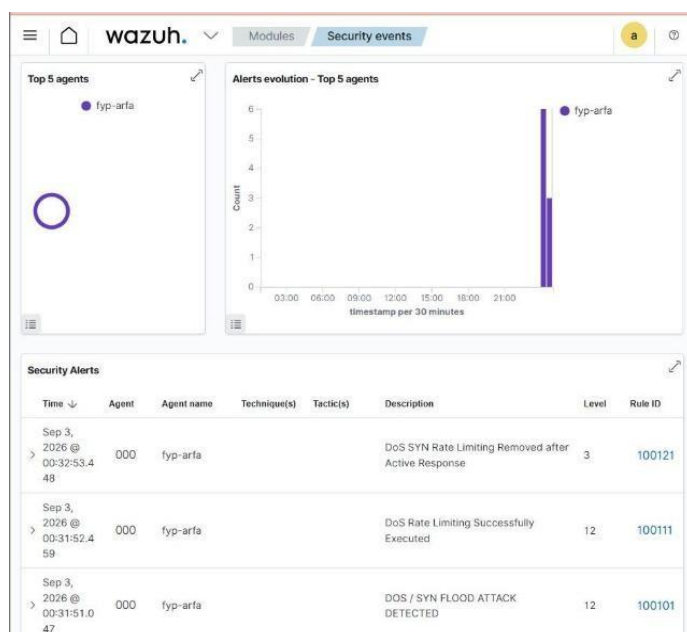


Figure 7. Wazuh sequence for SYN-flood detection, rate limiting, and response removal.

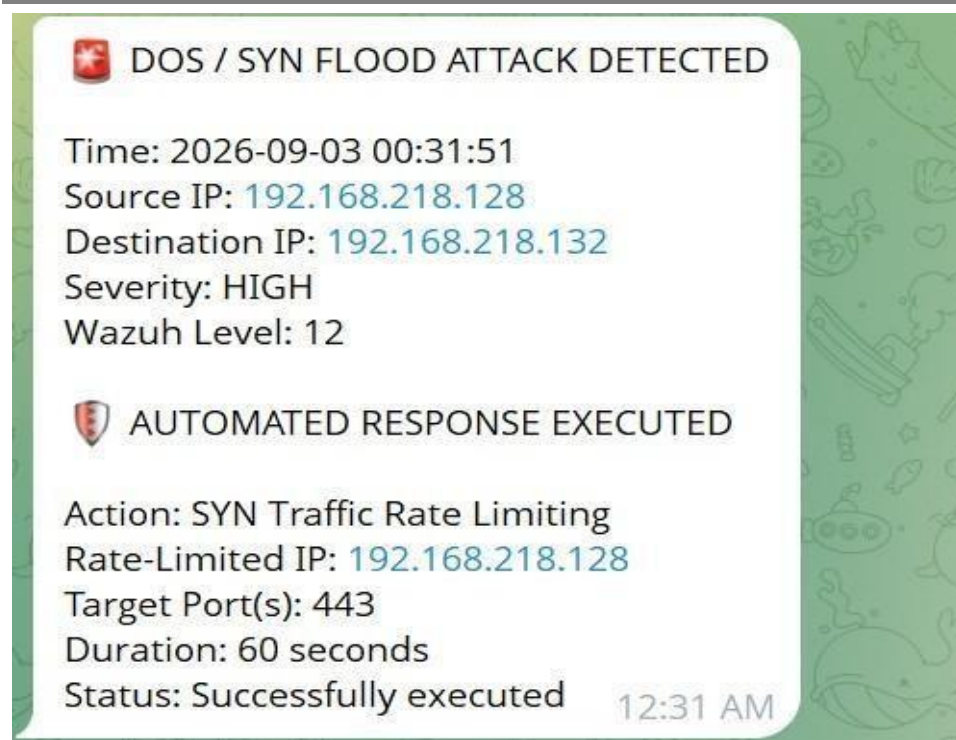


Figure 8. Telegram notification confirming SYN traffic rate limiting on TCP port 443.

SQL-Injection Evidence

Figure 9 shows SQL-injection detection, temporary blocking, and response removal in Wazuh, while Figure 10 shows the corresponding Telegram notification. The failed web request in Figure 11 independently verifies the temporary network restriction. This evidence demonstrates source- and port-scoped access containment only; it does not establish SQL query sanitisation or Web Application Firewall (WAF) protection.



Figure 9. Wazuh sequence for SQL-injection detection, temporary blocking, and response removal.

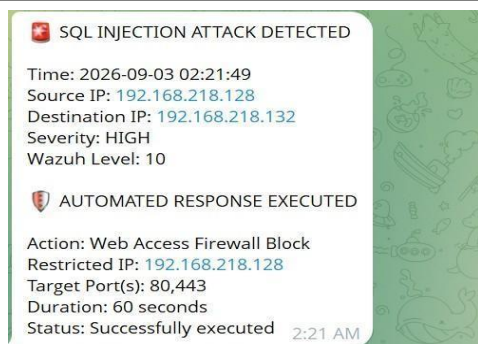


Figure 10. Telegram notification for SQL-injection detection and temporary web-access blocking.

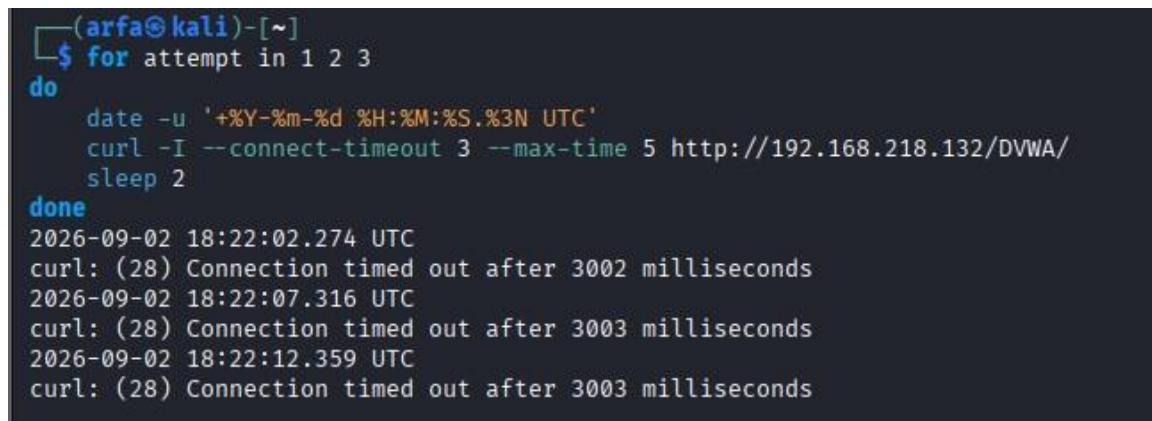


Figure 11. Web request timeout while the temporary network-level restriction was active.

Reverse-Shell Evidence

Figure 12 shows the Wazuh detection, containment, and removal events, and Figure 13 shows the Telegram notification. The handler terminated the observed channel and restricted outbound communication from the victim to the attacker listener on TCP port 4444. The timeout in Figure 14 verifies that immediate reconnection was prevented. A later connection test succeeded after timed removal, confirming recovery.

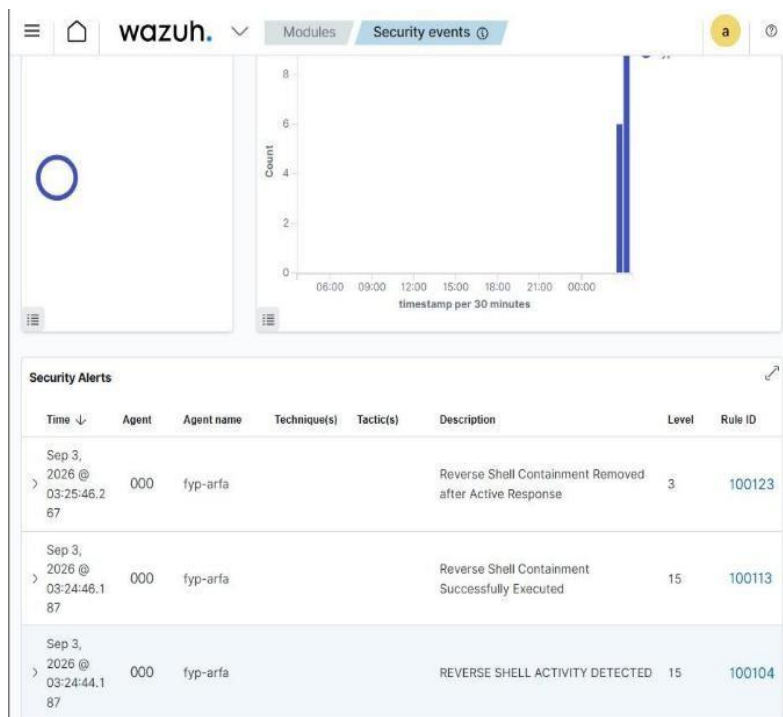


Figure 12. Wazuh events for reverse-shell detection, successful containment, and response removal.



Figure 13. Telegram notification for reverse-shell detection and outbound containment.

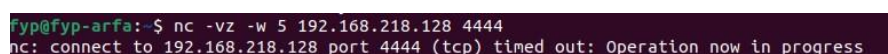


Figure 14. Reverse-shell reconnection timeout while the outbound restriction was active.

Response and Recovery Timing

Table 6 presents the detection-to-response and recovery intervals obtained using the definitions in the Timing Methodology subsection.

Table 6. Detection-to-response and recovery timing

Attack	Response Delay	Recovery Time	Result
Port scanning	0.004 s	63.668 s	Pass
SSH brute force	0.105 s	62.109 s	Pass
SYN flood	1.412 s	60.989 s	Pass
SQL injection	0.113 s	62.002 s	Pass
Reverse shell	2.000 s	60.080 s	Pass
Average	0.727 s	61.770 s	-

The temporary IP blocks for port scanning and SSH brute force were applied within 0.004 and 0.105 seconds. The SYN-flood rate-limiting control required 1.412 seconds, while the SQLinjection network restriction required 0.113 seconds. Reverse-shell containment recorded the longest delay at 2.000 seconds because it required interruption of an active channel and restriction of the relevant outbound connection. All observed response delays remained below the project-specific 10-second functional acceptance criterion.

Recovery was intentionally configured around a 60-second timeout. Observed recovery times ranged from 60.080 to 63.668 seconds. The additional interval reflects scheduling, command execution, service checking, and logging overhead between the nominal timeout and recorded recovery evidence. Because each value came from one execution, the differences must not be interpreted as statistically stable performance rankings.

Attack-Specific Analysis

For port scanning and SSH brute force, the framework temporarily blocked only the attacking source address rather than disabling the protected services globally. This preserved access for other hosts and allowed the original service state to return after the timeout.

For the SYN flood, rate limiting was selected instead of a complete service block. During the measurement interval, the firewall DROP counter increased by 184 packets while the ACCEPT counter increased by seven

packets. Thus, 184 of 191 observed counter increments, or approximately 96.3%, were dropped while some traffic remained accepted. This counter-based observation describes that interval only and is not a general packet-loss statistic.

For SQL injection, the implemented response was a temporary network-level restriction for the attacker address on TCP ports 80 and 443. The evidence confirms temporary web-access containment but does not demonstrate SQL input sanitisation or a dedicated WAF. For reverse shell, the response logic considered connection direction: the Ubuntu host initiated the callback to the Kali listener, so the system interrupted the session and restricted outbound communication to the attacker destination.

DISCUSSION

Interpretation of the Findings

The findings confirm end-to-end functional integration. A network event was detected by Suricata, correlated by Wazuh, presented through the dashboard and Telegram, and linked to an attack-specific response. Independent evidence then demonstrated a changed attack outcome, while removal evidence showed that the temporary control did not remain indefinitely. This detect–respond–recover sequence is consistent with the connected incidentresponse functions described by NIST [2].

The use of attack-specific controls was important. A generic block action would not adequately represent the different traffic directions and operational risks of the five scenarios. Source-IP blocking was suitable for port scanning and SSH brute force, rate limiting preserved partial service availability during the SYN flood, a network-level web restriction contained the SQLinjection source without claiming application-layer remediation, and outbound blocking addressed the reverse-shell callback direction.

Safety and Operational Risk

Automation shortens the interval between detection and action, but a false positive can turn the response mechanism into a source of service disruption. The most serious risks in this framework are blocking a legitimate shared address, applying an overly broad port rule, interrupting an authorised session, and failing to remove a temporary control. Protected-address exclusions, scoped rules, duplicate checks, execution logs, and timed rollback reduced these risks during the laboratory tests. Production use would require formal allowlist governance, staged or human-approved actions for high-impact cases, monitoring of service health after enforcement, rate limits on repeated playbook activation, and tested emergency rollback procedures.

Validity and Limitations

The experimental evidence is intentionally narrow. Each attack scenario was executed once, one attacker and one monitored host were used, and the isolated network did not include representative benign background traffic, shared-address behaviour, concurrent attacks, or enterprise-scale load. The test therefore establishes only that the configured functional paths worked in the documented environment.

The design cannot establish reliability across repeated runs or compute true positives, true negatives, false positives, and false negatives. Precision, recall, detection accuracy, falsepositive rate, false-negative rate, standard deviation, confidence intervals, and timing variability are consequently unavailable. Signature-based rules may also miss unknown or modified attacks, while overly broad signatures may affect legitimate traffic. These constraints prevent generalisation of the 5/5 functional result to production environments.

Future Evaluation

A stronger evaluation should execute multiple trials per attack, randomise test order, introduce benign and mixed traffic, use multiple source addresses and monitored hosts, and record confusion-matrix outcomes. Timing should be measured from synchronised clocks and reported with the number of trials, mean, median, standard deviation, confidence intervals, and outliers. SQL-injection containment should be extended with

ModSecurity and the Open Worldwide Application Security Project Core Rule Set, while future orchestration should include approval modes, case management, playbook versioning, health checks, and policybased rollback.

CONCLUSION

This study developed and evaluated an integrated system for automated threat detection and response using Suricata, Wazuh, and Telegram. Five controlled attack scenarios were detected, reported, mitigated, and recovered using attack-specific temporary controls. All five predefined scenario-level tests completed successfully, producing a 5/5 functional pass result within the controlled laboratory. The descriptive average response delay was 0.727 seconds and the average recovery time was 61.770 seconds.

The main contribution is an auditable closed loop that links detection, correlation, administrator notification, verified containment, timed rollback, and recovery. The result demonstrates proof-of-concept feasibility; it does not establish general IDS accuracy, reliability, or production readiness. Repeated attack trials, benign controls, multi-source traffic, statistical analysis, and stronger response governance are required before broader deployment claims can be made.

ACKNOWLEDGEMENT

The authors acknowledge Universiti Teknikal Malaysia Melaka and the Faculty of Artificial Intelligence and Cyber Security for the facilities and supervision supporting this project.

REFERENCES

1. K. Scarfone and P. Mell, Guide to Intrusion Detection and Prevention Systems (IDPS), NIST Special Publication 800-94, National Institute of Standards and Technology, 2007.
<https://doi.org/10.6028/NIST.SP.800-94>
2. A. Nelson, S. Rekhi, M. Souppaya, and K. Scarfone, Incident Response Recommendations and Considerations for Cybersecurity Risk Management: A CSF 2.0 Community Profile, NIST Special Publication 800-61 Revision 3, National Institute of Standards and Technology, 2025.
<https://doi.org/10.6028/NIST.SP.800-61r3>
3. Open Information Security Foundation, Suricata Documentation: Rules and EVE JSON Output, 2025.
<https://docs.suricata.io/>
4. E. Albin and N. C. Rowe, "A realistic experimental comparison of the Suricata and Snort intrusion-detection systems," in Proc. 26th International Conference on Advanced Information Networking and Applications Workshops, 2012, pp. 122–127. <https://doi.org/10.1109/WAINA.2012.29>
5. D. Fadhilah and M. I. Marzuki, "Performance analysis of IDS Snort and IDS Suricata with many-core processor in virtual machines against DoS/DDoS attacks," in Proc. 2nd International Conference on Broadband Communications, Wireless Sensors and Powering, 2020.
<https://doi.org/10.1109/BCWSP50066.2020.9249449>
6. Wazuh, Wazuh Documentation: Log Data Collection and Active Response, 2025.
<https://documentation.wazuh.com/>
7. Telegram, Telegram Bot API, 2025. <https://core.telegram.org/bots/api>
8. A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," Cybersecurity, vol. 2, article 20, 2019.
<https://doi.org/10.1186/s42400-019-0038-7>
9. T. Suryantoro, B. D. P. Purnomosidi, and W. Andriyani, "The analysis of attacks against port 80 webserver with SIEM Wazuh using detection and OSCAR methods," in Proc. 5th International Seminar on Research of Information Technology and Intelligent Systems, 2022.
<https://doi.org/10.1109/ISRITI56927.2022.10052950>
10. Wazuh, "Network IDS integration: Integrating Suricata with Wazuh," Proof of Concept Guide, 2025.
<https://documentation.wazuh.com/current/proof-of-concept-guide/integratenetwork-ids-suricata.html>
11. Wazuh, "Custom Active Response scripts," Active Response Documentation, 2025.

<https://documentation.wazuh.com/current/user-manual/capabilities/activeresponse/custom-active-response-scripts.html>

12. J. Kinyua and L. Awuah, "AI/ML in security orchestration, automation and response: Future research directions," *Intelligent Automation & Soft Computing*, vol. 28, no. 2, 2021.

<https://doi.org/10.32604/iasc.2021.016240>

13. National Institute of Standards and Technology, "Security orchestration, automation, and response," CSRC Glossary, 2025.

https://csrc.nist.gov/glossary/term/security_orchestration_automation_and_response