Automated Real-Time Detection of Wireless Network Attacks Using Open-Source Tools

Nur Aleeya Syazwin Suhaimi and Azlinda Abdul Aziz*

***Abstract*—Wireless networks remain vulnerable to a wide range of threats, including session hijacking, denial-of-service (DoS) attempts, and replay attacks. Many existing studies rely on physical laboratory setups and specialized hardware. This make it costly and less accessible for students, educators, and small-scale researchers. To address this limitation, this study develops a fully virtualized and low-cost environment for simulating and detecting common wireless network attacks. The testbed was built on an Ubuntu environment running through Windows Subsystem for Linux (WSL). Therefore various attack scenarios were carried out using open-source tools such as Nmap, Tshark, and Scapy. A set of automated Bash and Python scripts was implemented. These automated scripts perform continuous scanning, packet capturing, and real-time alerting. The experimental results show the virtual environment can closely mimic realistic attack behaviors. The result is also reliable to identify unusual traffic patterns. Overall, the proposed framework offers a practical software-based approach for intrusion detection. Thus it provides an accessible platform for hands-on learning and experimentation in wireless network security.**

***Index Terms*—Intrusion detection, Nmap, packet sniffing, Tshark, virtual testbed, wireless network security**

I. INTRODUCTION

In today's digital landscape, wireless communication plays a central role in daily connection, information access, and online transactions [1]. The public WiFi networks and community hotspots has further increased the convenience of user to access the internet from any location. However, this convenience also introduces significant security concerns, especially when wireless networks lack strong encryption mechanisms and not have proper authentication practices [2]. One of the most common threats is packet sniffing, where attackers can capture unencrypted data transmitted across the network. Thus, potential sensitive information such as login credentials, personal details, and user activities were exposed by the attacker [3]. Wireshark and Tshark tools are widely used for legitimate network diagnostics pattern. Tshark is a tool to capture misused for malicious purposes. Similarly, Nmap is a tool often used by administrators to assess system vulnerabilities. This software may be exploited by attackers to identify open ports and uncover system weaknesses.

The need for accessible, affordable, and intelligent methods to detect unusual and harmful network behavior in real time has become increasingly urgent in cyberattacks [4]. In respond to this challenge, this study introduces a virtualization framework designed to simulate and detect wireless network attacks within a controlled environment. This framework focus on identify common threats, including session hijacking, denial of service attacks and replay attacks. The threats are often targeted in poorly secured wireless infrastructures [5]. By utilizing open source tools such as Tshark, Nmap, and automated Python monitoring scripts, the proposed system provides a practically. This approach help to enables real-time intrusion detection and supports scalable experimental applications in wireless network security.

II. LITERATURE REVIEW

Network vulnerabilities play an important role in enable attackers to infiltrate and compromise wireless systems. Weak of encryption, exposed ports, outdated software, and poorly configured firewalls are the most common weaknesses that can lead to serious security incidents [6]. For example, older encryption protocols such as Wired Equivalent Privacy (WEP) make it easier for attackers to intercept and access confidential information. Open ports may unintentionally make network services accessible to unauthorized users. While improper configuration of firewalls may fail to filter suspicious traffic and vulnerabilities. These vulnerabilities often serve as entry points for attacks such as session hijacking and denial of service attacks. Replay attacks expose weaknesses that need to be addressed to ensure data remains confidential, accurate, and available. [7].

To reduce these risks, network scanning and monitoring tools is essential to used. Nmap is widely recognize for ability to identify active hosts, detect open ports, and expose configuration issues. Nmap also provide useful insights into the overall security bearing in the network [8]. The function is to captures and examines real time network traffic and helps identify unusual packet patterns. It also can make repeated transmissions, and detect unsecured communication that may an active attack signaling [9]. As both tools are open source and lightweight, they can be easily integrated into automated scripts. Both of these open-source tools make them suitable for simulation-based studies and controlled research environments.

This manuscript is submitted on December 26, 2025, revised on June 11, 2026, accepted on July 06, 2026 and published on October 31, 2026.

Nur Aleeya Syazwin Suhaimi, Azlinda Abdul Aziz with Faculty of Computer Science and Mathematical, Universiti Teknologi MARA (UiTM) Melaka, Campus Jasin, 77300 Merlimau, Melaka MALAYSIA. (e-mail: azlindaaziz@uitm.edu.my)

*Corresponding author
Email address: azlindaaziz@uitm.edu.my

1985-5389/© 2026 The Authors. Published by UiTM Press. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Table 1 presents a comparison of active and passive packet sniffing techniques. This table shown the types of traffic patterns detected using tools such as Nmap and Tshark.

TABLE 1: ACTIVE VS. PASSIVE PACKET SNIFFING (SOURCE: METWALLY ET AL. (2022))

Type of Sniffing	Description	Example Attack	Risk Level
Active Sniffing	Intercepts and modifies data during transmission	Man-in-the-Middle (MITM)	High
Passive Sniffing	Monitors and records network traffic without altering it	Eavesdropping	Medium

A variety of packet analysis tools have been developed to support cybersecurity research and network monitoring. Wireshark provides a graphical interface for detailed inspection. While Tcpdump is the fast packet capture through the command line. Tshark combines the strengths of both approaches by providing an analytical environment that is compatible with automated workflows [10]. This flexibility allows for seamless integration with Python or Bash scripts for continuous monitoring. In this study, Tshark was selected due to its strong automation capabilities. Moreover the efficient performance on Ubuntu, and compatibility with the scripts required for continuous traffic analysis. Table 2 summarizes common network vulnerabilities alongside the types of threats associated with them. This table highlights how Nmap and Tshark can assist in weaknesses detecting.

TABLE 2:COMMON NETWORK VULNERABILITIES AND THEIR ASSOCIATED THREATS

Vulnerability	Example Threat	Description	Mitigation Method
Unencrypted Communication	Data Theft	Sensitive data transmitted in plaintext	Use HTTPS/TLS encryption
Weak Encryption (e.g., WEP)	Unauthorized Access	Outdated protocols easily compromised	Upgrade to WPA3 or strong encryption
Open Ports	DDoS, Intrusion	Unrestricted services accessible to attackers	Conduct regular port scans using Nmap
Misconfigured Firewalls	Network Breach	Incorrect rules expose internal systems	Review and harden firewall configurations
Session Management Flaws	Session Hijacking	Attackers reuse session tokens	Implement secure session handling

In network analysis tools, many previous studies examine these tools individually rather than integrating them into a unified detection system. Packet sniffing continues to be a major cybersecurity challenge in wireless environments where encryption is weak and not implemented [11]. While simulated attacks provide a safe method to study attacker behavior. Several previous studies have used packet analysis tools and network monitoring techniques to detect cyberattacks in wireless environments. Some studies focused on traffic analysis using Wireshark or Tshark, while others used Nmap for vulnerability assessment and network scanning. Other researchers have developed intrusion detection systems using virtualized environments to simulate attack scenarios. However, most studies evaluate these tools separately and require dedicated hardware or complex laboratory setups. Limited research has integrated Nmap, Tshark, and automated scripting within a single low-cost virtualized framework for real-time attack detection and alert generation.

Therefore, only limited research has explored the integration of Nmap, Tshark, and automated detection scripts within a single framework for real-time monitoring and alert generation. This gap highlights the need for an accessible, scalable, and cost-effective system capable of detecting wireless network attacks in a virtualized environment. To address this gap, the present study proposes a virtualized system that integrates Nmap and Tshark to enable real-time attack detection and alert generation.

To address this gap, the present study proposes a virtualized system that integrates Nmap and Tshark to enable real time attack and alert detection [12]. By combining vulnerability assessment with continuous network traffic monitoring, the system enhances the ability to detect anomalies quickly. Through this method it reduce the impact of potential attacks. This approach offers an accessible, low cost, and scalable framework. Thus the literature demonstrates the value of integrating vulnerability assessment with continuous monitoring with strengthen wireless network security.

III. OBJECTIVE

1) To identify network vulnerabilities by conducting automated port scanning and traffic analysis within a simulated wireless network environment.
2) To design and implement an automated detection system capable recognizing session hijacking, denial of service attacks, and replay attacks in a virtual environment
3) To evaluate the performance of an automated alert mechanism in notifying administrators of suspicious and abnormal network behavior

IV. METHODOLOGY

The methodology for this study was designed to examine wireless network security threats by simulating controlled attacks. It also implementing automated detection using Tshark and Nmap. The overall process consists of four main phases consist of the experimental environment setup, design, development and testing.

Figure 1 presents the workflow of the proposed methodology. The process begins with the setup of the virtualized experimental environment, followed by the design and development of the attack detection framework. Simulated attack scenarios were then executed and monitored using automated tools. Finally, the generated alerts and detection logs were evaluated to assess the effectiveness of the proposed framework.

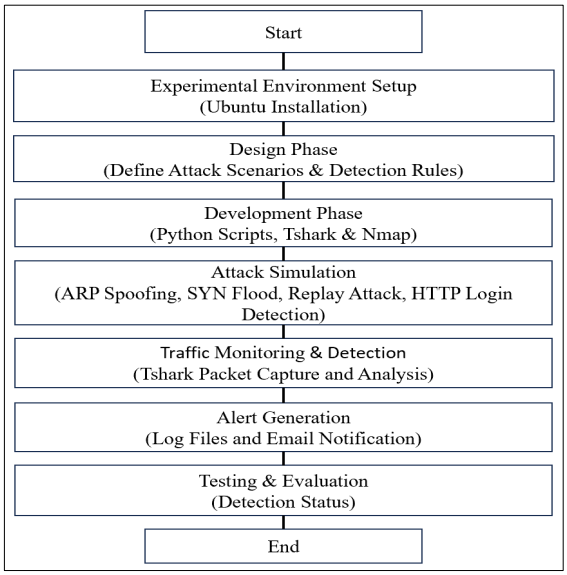


Fig. 1. The Workflow of Methodology

A. Experimental Environment Setup

The experimental environment was prepared using Windows Subsystem for Linux (WSL) running Ubuntu 22.04. This environment support Linux based network analysis and traffic manipulation. Then a virtual wireless testbed was created by configuring simulated interfaces. It is specifically *wlan0* and *eth0* port and assigning private IP addresses to represent typical network components. Then the attacker machine was assigned the address *192.168.1.50*, the victim *192.168.1.10*, and the gateway *192.168.1.1*.

Several open source tools, including Nmap, Tshark, Scapy, hping3, and Curl, was installed. The purpose for installed the open source tools is to support port scanning, packet capturing, traffic generation and the simulation of malicious activities. Virtual interfaces are created using the *ip address* and *ip link* commands. This aim is to allow reproduce wireless communication patterns where required for both attack simulation and detection.

B. Design Phase

In this phase the attack behaviours are model and the corresponding detection workflow was defined. Three major threats were selected for simulation: session hijacking through ARP spoofing, denial of service attacks using SYN flooding, and replay attacks. For each threat, the packet flow and the interactions between the attacker and victim are describe. Then the expected abnormal traffic patterns were outlined in showing how the attack could be detected.

Specific indicators was identified as detection triggers. In this study unsolicited *ARP* replies, unusually high volumes of *SYN* packets without acknowledgment. Then the repeated packet sequences are identified. After that a structured detection workflow was developed. Tshark captures real time traffic on *wlan0* port. The predefined filters through the alert mechanism are used to isolate suspicious patterns and forwards abnormal events. The design provided a clear understanding of how each attack reveals and how the detection system should respond.

C. Development Phase

In this phase, both the simulations attack and the logic detection was implement using Bash and Python scripts.

i) ARP spoofing was performed using Scapy to send forged ARP replies to impersonated the gateway at *192.168.1.1* and targeted the victim at *192.168.1.10.*

ii) Denial of service simulations was carried out using *hping3*. The purpose is to send large numbers of *SYN* packets to the victim.
hping3 -S --flood -p 80 192.168.1.10

iii) Replay attacks was generated by capturing legitimate packets with Tshark and retransmitting them using Scapy.

Next the automated Nmap scanning was used to identify active hosts and exposed services within the *192.168.1.0/24* network. Tshark was configured to perform continuous packet capturing using the command *tshark -i wlan0 -w logs.pcap*. While suspicious patterns are detection scripts to identify the analyzed live traffic. Moreover, this process enable automated attack simulation, monitoring and detection are link together with all scripts.

D. Testing Phase

In this phase, the detection framework under controlled attack conditions are evaluate. Each simulated attack was launched from the attacker machine using the ip *192.168.1.50* against the victim using ip *192.168.1.10* within the Ubuntu WSL environment. Then Tshark monitored live network traffic to confirm the accurate detection filters. The accurate detection are identified malicious behaviours for *ARP* spoofing, *SYN* flooding, and replayed packet sequences.

Next, Timestamps, packet counts and trigger events are record for scripts detection. Whenever abnormal activity was detected in system, notification alert are strictly given. The collected logs and packet traces was examined to assess the accuracy, consistency and responsiveness of the detection during all attack scenarios. The accuracy, consistency and responsiveness was detected when the examiner examinate the log collection and packet traces in testing phase.

E. Evaluation Matrics

To address Objective 3, the performance of the automated alert mechanism was evaluated using Alert Detection Rate (ADR) and Alert Generation Success Rate (AGSR). ADR measures the percentage of attacks successfully detected by the system. AGSR measures the percentage of alerts successfully generated during the attack simulations. These metrics were evaluated using the logs and alert records collected during ARP spoofing, SYN flood, replay attack, and unencrypted HTTP login tests. The detection status of each attack was checked to confirm that the system successfully detected suspicious activities and generated alerts.

V. EXPERIMENTAL SETUP AND RESULTS

A. Overview of the Experimental Environment

To evaluate the proposed detection framework, Ubuntu under Windows Subsystem for Linux (WSL) are used to controlled all experiments that carried out. The configuration provide the flexibility, low-cost and fully isolated platform. Thus it enable the safety simulation for multiple network attacks without need for physical laboratory equipment. Nmap, Tshark, Scapy, and Curl as a open source are used for this experiment. Moreover, the bash and Python script are automate attack simulations in capture network traffic, and generate real-time alerts [13].

B. Develop and Configure Attack Scripts

```
# 2. Launch ARP spoofing attack
echo "Launching ARP spoofing attack..."
sudo arpspoof -i $INTERFACE -t $VICTIM_IP $GATEWAY_IP > $LOG_DIR/arp_spoof.log 2>&1 &
ARP_PID=$!

# 3. Launch SYN flood (brief test)
echo "Testing SYN flood detection..."
sudo hping3 -S -p 80 --flood $VICTIM_IP -c 100 > $LOG_DIR/syn_flood.log 2>&1 &


# 4. Generate fake login attempt
echo "Generating test login..."
curl -X POST http://$VICTIM_IP/login -d "username=test&password=test" > $LOG_DIR/login_attemp
t.log 2>&1 &
```

Fig. 2. A bash Script Attacks on Sniffing

In this phase, the detection of evaluation using multiple simulated attack scenarios is developed and configured. The purpose is to assess the performance in real-time monitoring, logging and alert generate. Figure 2 illustrates the overall setup and workflow of the attack simulations. The results, along with interpretations, are presented below.

1) ARP Spoofing Detection

In the first scenario, an ARP spoofing attack was performed using the *arp spoof* utility. Falsified ARP replies caused the victim device to associate the attacker's machine with the network gateway and to allow interception of all traffic[14]. The detection system successfully identified the abnormal ARP responses in real time. Events were logged in arp_spoof.log, and the process ID was stored in *$ARP_PID* for controlled management.

Discussion: The unsolicited ARP replies mean the system identification was demonstrates effectiveness in detecting session hijacking. Continuous monitoring are needed to ensure the attacks can be mitigated before significant data is compromised.

2) SYN Flood Detection

In using *hping3***,** TCP SYN burst packets are sending to targeting port 80 on the victim system. Then it simulated by using SYN flood attack.

The *--flood* option maximized transmission speed, while *-c 100* ensured a safety. The framework captured abnormal traffic patterns, logged them in *syn_flood.log*, and triggered immediate alerts.

Discussion: The detection system accurately responded to high-volume traffic anomalie. The indicator capability to identify denial-of-service attacks. Moreover the system can provide early warnings without affecting normal network operations.

3) Replay Attack Simulation

Replay attacks were generated by retransmitting previous packets captured using Scapy. The system monitored for repeated identical sequences of packets. Then flagged them as anomalies and recorded detailed logs. Alerts were triggered whenever repeated traffic patterns exceeded predefined.

Discussion: This result shows the framework was detect indirect attacks attempt where there need to exploit legitimate communication patterns. Real-time detection of replayed packets is crucial for maintaining in weak of encryption.

4) Unencrypted HTTP login Detection

Finally, an unencrypted HTTP login attempt was simulated with a *curl POST* request, copying a user transmitting credentials over an unsecured channel. The system successfully identified the unencrypted traffic, logged it in *login_attempt.log*, and generated an alert.

Discussion: Detecting unsecured credential transmission highlights the framework's relevance for practical cybersecurity scenarios. This capability is particularly important in educational settings, where learners can safely study the risks of transmitting sensitive data without encryption.

The result of the overall observation in the scenario, the system consistently captured malicious actions. Then the system generated real-time alerts, and produced detailed logs suitable for forensic analysis. Automated scripts allowed continuous monitoring without manual intervention. The combination of Nmap scanning, Tshark packet capture, provided a flexible and scalable detection framework.

Discussion: The experimental findings confirm that the proposed virtualized framework delivers effective, reliable, and reproducible results. Its ability to detect multiple types of attacks in a safe, isolated environment makes it suitable for research and small-scale institutional deployment. By combining automated monitoring with detailed logging, the system provides both immediate operational alerts and understandings long-term analytical.

B.1 Evaluation of Alert Mechanism

Table 3 shows the result of the evaluation matrix of alert mechanism.

TABLE 3. OPEN EVALUATION OF ALERT MECHANISM

ATTACK TYPE	ALERT GENERATED	DETECTION STATUS
ARP Spoofing	**Yes**	**Detected**
SYN Flood	**Yes**	**Detected**
Replay Attack	**Yes**	**Detected**
HTTP Login	**Yes**	**Detected**

The experimental results show that the alert mechanism successfully detected all simulated attacks.

C. *Nmap Automated Scanning Results*

```
# 2. Port Scan Detection
echo -e "\n[PORT SCAN RESULTS]" >> $ALERT_FILE
sudo nmap -sS $VICTIM_IP | grep "open" >> $ALERT_FILE
```

Fig. 3. Port Scan Detection

Figure 3 shows the process of performing a *TCP SYN* scan using Nmap with the command sudo nmap *-sS $VICTIM_IP*. The process of scanning helps to identify which ports on the target machine are open by sending SYN packets. It also observes how the system responds. By using the **sudo** command it can provide the necessary privileges for Nmap to access low-level network functions and produce accurate results.

After the scan is completed, the output is filtered with grep open to display only the entries that indicate open ports. These filtered results are added to a log file defined by *$ALERT_FILE* command. It is also called alerts.log, under the header *[PORT SCAN RESULTS]*. This log serves as a central record of all detected security events for later review.

In this study, the command is integrated into the automated detection system to continuously monitor active services that could expose potential vulnerabilities [15]. The collected information can also be used to trigger email alerts, help the system provide real-time threat detection and reporting.

D. *Attack Detection Bash Script Result*

```
    # 1. ARP Spoofing Detection
    echo -e "\n[ARP SPOOFING CHECK]" >> $ALERT_FILE
    sudo timeout 30 tshark -i $INTERFACE -Y "arp.opcode == 2" -a duration:30 -w $LOG_DIR/arp.p
cap 2>/dev/null
    ARP_COUNT=$(tshark -r $LOG_DIR/arp.pcap 2>/dev/null | wc -l)
    [ $ARP_COUNT -gt 20 ] && echo "ALERT: ARP spoofing detected ($ARP_COUNT replies)" >> $ALER
T_FILE || echo "No ARP spoofing detected" >> $ALERT_FILE

    # 2. Port Scan Detection
    echo -e "\n[PORT SCAN RESULTS]" >> $ALERT_FILE
    sudo nmap -sS $VICTIM_IP | grep "open" >> $ALERT_FILE
    [ -s $LOG_DIR/logins.pcap ] && echo "ALERT: Unencrypted logins detected" >> $ALERT_FILE ||
echo "No session hijacking detected" >> $ALERT_FILE

    # 4. DDoS Detection
    echo -e "\n[DDoS ATTEMPT CHECK]" >> $ALERT_FILE
    sudo timeout 30 tshark -i $INTERFACE -Y "tcp.flags.syn==1 && tcp.flags.ack==0" -w $LOG_DI
R/syn.pcap 2>/dev/null
    SYN_COUNT=$(tshark -r $LOG_DIR/syn.pcap 2>/dev/null | wc -l)
    [ $SYN_COUNT -gt 50 ] && echo "ALERT: SYN Flood detected ($SYN_COUNT packets)" >> $ALERT_F
ILE || echo "No DDoS detected" >> $ALERT_FILE

    # 5. Replay Attack Detection
    echo -e "\n[REPLAY ATTACK CHECK]" >> $ALERT_FILE
    sudo timeout 30 tshark -i $INTERFACE -w $LOG_DIR/general.pcap 2>/dev/null
    REPLAY_COUNT=$(tshark -r $LOG_DIR/general.pcap -Y "tcp.analysis.retransmission" 2>/dev/nul
l | wc -l)
    [ $REPLAY_COUNT -gt 10 ] && echo "ALERT: Replay attack detected ($REPLAY_COUNT retransmiss
ions)" >> $ALERT_FILE || echo "No replay attacks detected" >> $ALERT_FILE
```

Fig. 4. A Bash Script for Detection

Figure 4 shows the Attack Detection Bash Script design. The purpose is to automate real-time threat detection by analyzing network traffic for suspicious patterns. The script are monitors in using Tshark and specific to capture the attack signatures. Then the script is also integrated with an email alert system to notify administrators immediately upon detecting anomalies and enabling a rapid response [16]. The Bash detection script successfully monitored network traffic and identified predefined attack signatures during the experiments. The system generated alert notifications whenever suspicious activities were detected, demonstrating the capability of the framework to perform automated real-time attack detection.

The efficient design for the script development provides a powerful tool for identify and mitigate network threats in real-time live network [17]. Moreover, it can be easily adapted to new attack vectors and detection rules.

E. *Analyse Logs Alerting System Result*

```
# 7. Check results
if grep -q "ALERT" $LOG_DIR/monitoring.log; then
    echo "Attacks detected! Sending alert email."
    send_alert "$(cat $LOG_DIR/monitoring.log)"
else
    echo "No attacks detected. Checking why..."
    send_alert "TEST FAILED: No attacks detected\n\n$(cat $LOG_DIR/monitoring.log)"
fi
```

Fig. 5. Configuration of Alerting System

Figure 5 shows the configuration of the alert system result. The script is responsible for examining the monitoring logs to determine if any attacks have been detected. The script uses grep command to search for the keyword *ALERT* within the *monitoring.log* file located in the designated log directory. When this keyword is found, it indicates the intrusion detection system. The suspicious activity is prompted by this script. The purpose is to send an email alert containing the relevant log details [18]. If no alert is present, the script interprets this as no attacks being detected and sends a separate notification email, including the full *monitoring.log* file for further review. This approach is to ensure the detection process is valid through the email update.

The analysis of the monitoring logs showed that the system successfully detected the simulated attack activities. Alert notifications were generated for ARP spoofing, SYN flood, replay attack, and unencrypted HTTP login attempts. The generated logs confirmed that suspicious network activities were identified and recorded for further investigation. These results demonstrate the effectiveness of the alerting mechanism in supporting real-time network monitoring.

F. Open Ports Alert on Email Messages

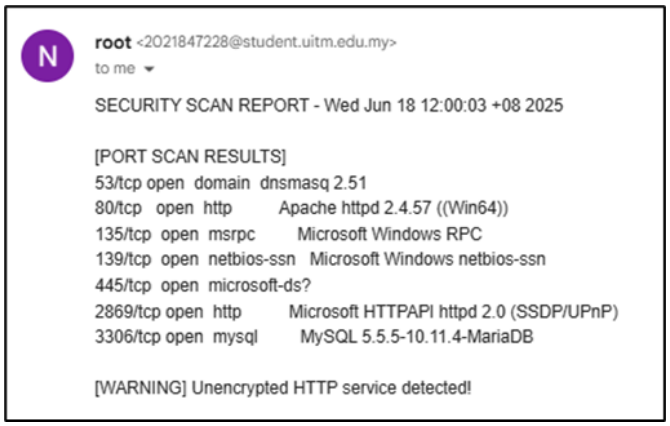


Fig. 6. Nmap Scan Output

Figure 6 shows the security scan for identifying the open ports, which potentially expose the network to vulnerabilities based on the running services on the network. The result shows the port found to be open consists of ports 53 (DNS), 80 (HTTP), 135 (MSRPC), 139 (NetBIOS), 445 (Microsoft-DS), 2869 (HTTP), and 3306 (MySQL). The particular concern port 80, where unencrypted HTTP traffic may allow sensitive information to be intercepted. Port 139 (NetBIOS) and Port 445 (Microsoft-DS) are also irritating because they have historically been exploited in high-profile attacks like EternalBlue.

Additionally, the service port 3306 MySQL detects an outdated Apache version and could potential for weaknesses for gain unauthorized access and data theft [19]. These findings highlight the need for immediate action to update vulnerable services**.** enable the encryption and restrict any unnecessary open ports to reduce the overall risk.

VI. Security Analysis of Open Ports

Based on the Nmap scanning results, Table 4 show the several open ports were identified and categorized the associated security risks. These findings highlight the potential vulnerabilities within the system and serve as a basis way for implement an appropriate mitigation strategies.

TABLE 4. Open Ports and Security Risk Detection

PORT	SERVICE	RISKS LEVEL	DESCRIPTION
53/tcp	DNS	MEDIUM	Potential DNS poisoning
80/tcp	HTTP (Apache)	HIGH	Unencrypted web traffic
135/tcp	MSRPC	HIGH	Windows RPC vulnerabilities
139/tcp	NETBIOS/SSN	CRITICAL	SMBv1 vulnerabilities
445/tcp	MICROSOFT-DS(SMB)	CRITICAL	Eternal Blue exploit potential
2869/tcp	HTTP	MEDIUM	UPnP NAT injection risks

A. High Risk Ports

High-risk ports are a major security concern because they are frequent targets for attackers and often linked to serious vulnerabilities. Port 135/tcp (MSRPC) is commonly exploited through DCOM-based attacks to gain unauthorized access to Windows systems. In addition, ports 139/tcp (NetBIOS) and 445/tcp (Microsoft-DS) are associated with SMB-related vulnerabilities and have been exploited in attacks such as EternalBlue. These ports present significant threats and require prompt attention to reduce potential security risks.

Thus, these ports may not be as immediately exploitable as critical-risk ports, they still present significant threats that require prompt attention. Effective mitigation strategies like enforcing strong authentication, applying strict limits to network access and ensuring all services are up to date with the latest security patches need to be implemented [20]. By using this step, organizations can substantially reduce the risk of successful attacks against these exposed services.

B. Medium Risk Ports

Medium-risk ports not provide attackers with immediate control over systems, but they can serve as entry points for more advanced attacks. Port 53/tcp (DNS) may be vulnerable to DNS spoofing and potentially redirect users to malicious sites. Similarly, port 2869/tcp (UPnP) can expose internal network details and allow external actors to identify hidden devices and services. Whereas, ports 8080/tcp may host misconfigured web services that could be exploited if not properly secured. Although these ports do not pose an immediate threat, they expand the network's attack surface. Recommended security measures include enabling secure communication protocols when using DNS-over-HTTPS. By using this process it was possible to disable unnecessary services and continuously monitor network activity for unusual behavior. Implementing these steps helps prevent attackers from using medium-risk ports as part of larger intrusion attempts.

C. Low Risk Port

Low-risk ports are generally considered secure when properly configured, but they still require monitoring to maintain overall network integrity. SSH on port 22/tcp is secure when key-based authentication is enforced instead of passwords. Similarly, port 443/tcp HTTPS provides encrypted communication, although outdated TLS versions and weak cipher suites can still pose vulnerabilities. While low-risk ports are not the main targets for attackers. Therefore, it is important to keep all software up to date, maintain detailed activity logs, and limit unnecessary exposure the services to the public internet. Consistent monitoring helps to ensure low-risk ports do not become potential entry points for attackers.

VII. Detection Test Alert on Email Messages

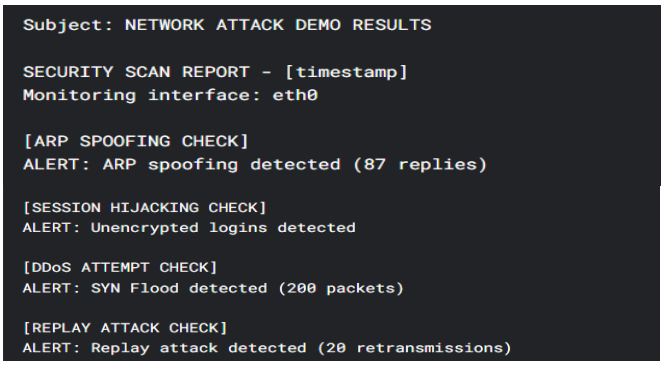


Fig. 7. Detection Alerts on Email Message

Figure 7 shows the script developed to simulate multiple network attacks and evaluate the effectiveness of security monitoring systems. The script executes *ARP spoofing* to redirect traffic to unintended destinations. SYN flood attacks emulate a Distributed Denial of Service (DDoS) by generating large volumes of connection requests and credential capture over unencrypted channels such as HTTP and FTP [21]. Additionally, TCP replay performs attacks by retransmit previous packets capture to test the system's resilience through replayed traffic.

Nmap scans also conducted to identify open ports without triggering immediate alerts. Each attack is carefully configured to generate standard security notifications. For instance, repeated ARP replies during spoofing, 200 SYN packets in flood attacks, and detection of unencrypted login attempts. The attacks are executed concurrently, allowing the monitoring system's response to multiple simultaneous threats to be assessed.

Network activity is monitored in real time using Tshark, a command-line tool based on Wireshark, with custom filters applied for each attack type. More than 20 ARP and 50 SYN packet responses are tested using intelligent rules to flag unusual activity. All network traffic is recorded in PCAP files for detailed post-test analysis.

Upon completion, the script generates an automated report via email. The script also summarizes the detected attacks and records the corresponding detection events in the monitoring logs. The process is fully automated, encompassing tool installation, attack execution, threat monitoring, cleanup, and report generation. This self-contained design ensures a controlled testing environment, making it ideal for evaluating security systems in realistic scenarios.

VIII. Conclusion

This study demonstrates the practical benefits of a fully virtualized framework for wireless network security. Moreover, it enables real-time detection of attacks such as session hijacking, denial-of-service (DoS), and replay attacks. By combining open-source tools, including Nmap and Tshark, with automated Bash and Python scripts, the framework provides a cost-effective and scalable environment. The results indicate that simulated attacks within this controlled setup can effectively replicate realistic network behaviors, allowing for accurate monitoring, anomaly detection, and timely alert generation. This approach offers a safe and repeatable platform for hands-on learning and small-scale institutional deployment without the need for expensive physical infrastructure. Future work could focus on integrating advanced threat intelligence. Moreover can applying machine learning for predictive detection and expanding the virtual testbed to accommodate more complex attack scenarios and larger network topologies.

Acknowledgment

The authors sincerely thank to all individuals and organizations who contributed to this study. We greatly appreciate for the support, guidance, and resources provided, which were instrumental in facilitating the research process. Their insights and assistance were invaluable in shaping the outcomes of this work.

References

[1] T. Agashov, "Wireless mobile communication systems: current status, application fields and challenges," Problems of Information Society, vol. 15, no. 1, pp. 86–97, Feb. 2024, doi: 10.25045/jpis.v15.i1.09.

[2] O. Mostafa, "Internet of Things_ Security, Issues, T..," vol. 19, Feb. 2024, doi: doi:10.12720/jmc.19.2.78-89.

[3] Duaa Sameer, Hussain, and Zaid, "Software Engineering_ Emerging Trends and Practices in System Development," pp. 202–220, Aug. 2025, doi: https://doi.org/10.1007/978-3-032-03406-9_13.

[4] M. Abdullah, M. M. Nawaz, B. Saleem, M. Zahra, E. binte Ashfaq, and Z. Muhammad, "Evolution Cybercrime—Key Trends, Cybersecurity Threats, and Mitigation Strategies from Historical Data," Sep. 01, 2025, Multidisciplinary Digital Publishing Institute (MDPI). doi: 10.3390/analytics4030025.

[5] L. Serena, G. D'Angelo, S. Ferretti, and M. Marzolla, "Simulation in Cybersecurity: Understanding Techniques, Applications, and Goals," Aug. 2025, [Online]. Available: http://arxiv.org/abs/2508.06106

[6] Ö. Aslan, S. S. Aktuğ, M. Ozkan-Okay, A. A. Yilmaz, and E. Akin, "A Comprehensive Review of Cyber Security Vulnerabilities, Threats, Attacks, and Solutions," Mar. 01, 2023, MDPI. doi: 10.3390/electronics12061333.

[7] D. S. Bhatti et al., "A Survey on Wireless Wearable Body Area Networks: A Perspective of Technology and Economy," Oct. 01, 2022, MDPI. doi: 10.3390/s22207722.

[8] Z. Al-Khazaali, A. Al-Ghabban, H. Al-Musawi, A. Sabah, and N. Al Mahdi, "Characteristics of Port Scan Traffic: A Case Study Using Nmap," Journal of Engineering and Sustainable Development, vol. 29, no. 1, pp. 26–35, Jan. 2025, doi: 10.31272/jeasd.2638.

[9] K. S. Lim, S. Y. Ooi, M. S. Sayeed, Y. J. Chew, and N. M. Ahmad, "Securing the Internet of Things: Systematic Insights into Architectures, Threats, and Defenses," Oct. 01, 2025, Multidisciplinary Digital Publishing Institute (MDPI). doi: 10.3390/electronics14203972.

[10] N. Misha, N. Binti, and M. Razali, "Development of Web-based Network Traffic Analysis and Monitoring System," 2025. [Online]. Available: http://127.0.0.1.500

[11] I. Laktionov, G. Diachenko, D. Moroz, and I. Getman, "A Comprehensive Review of Cybersecurity Threats to Wireless Infocommunications in the Quantum-Age Cryptography," IoT, vol. 6, no. 4, p. 61, Oct. 2025, doi: 10.3390/iot6040061.

[12] K. Hussain, "COMPUTER SCIENCE BULLETIN PRIVACY AND SECURITY IN SMART HOME DEVICES: CHALLENGES, SOLUTIONS, AND FUTURE DIRECTIONS," vol. 8, no. 1, 2025.

[13] A. R. Bagabaldo and J. Hackl, "Digital Twins for Intelligent Intersections: A Literature Review," Oct. 2025, [Online]. Available: http://arxiv.org/abs/2510.05374

[14] T. Bilot et al., Sometimes Simpler is Better: A Comprehensive Analysis of State-of-the-Art Provenance-Based Intrusion Detection Systems. 2025.

[15] J. Polónio, J. Moura, and R. N. Marinheiro, "Towards Automatic Detection and Mitigation of High-Risk Cybersecurity Vulnerabilities at

Networked Systems," IEEE Access, 2025, doi: 10.1109/ACCESS.2025.3622497.
[16] Y. Ali, T. Manzoor, H. Yang, A. Ali, and Y. Xia, "Towards Efficient and Secure Cloud Control Systems: Advances, Challenges, and Future Directions," Sep. 2025, [Online]. Available: http://arxiv.org/abs/2509.09299
[17] S. Singh, "Cyber Security Threats and Mitigation Strategies in the Digital Era," Jagannath University Journal of Research and Review, no. 02, 2025.
[18] N. Mukabbir, "Hybrid AI Frameworks for Real-Time Intrusion Detection and Threat Mitigation," Oct. 2025.
[19] Nataraj N and Nataraj R V, "AN EXPERIMENTAL STUDY ON NETWORK SECURITY GROUP POLICY FOR VIRTUAL INSTANCES," Oct. 2019. [Online]. Available: https://en.wikipedia.org/wiki/Cloud_computing_security
[20] K. Mohsin, "Cybercrime and Privacy in the Digital Age: Legal Frameworks, Emerging Challenges, and Future Trends," Aug. 2024.
[21] Rajest Tanti, "Study-of-Phishing-Attack-and-their-Prevention-Techniques," International Journal for Multidisciplinary Research (IJFMR), Sep. 2024.