פרויקט מס' BS-CS-25-210

Threat hunting - using YARA signature to detect cyber attacks in the organization

ציד איומים - שימוש בחתימות YARA לאיתור תקיפות סייבר בארגון

פרויקט מחקרי

דוח סופי

הוכן לשם השלמת הדרישות לקבלת

תואר ראשון במדעי הטבע .B. Sc

מאת

שחר דוד	**207696287**
רועי צור	**209439702**

בהנחיית

ד״ר גיא תל-צור

הוגש למחלקה למדעי המחשב

המכללה האקדמית להנדסה סמי שמעון

באר-שבע

תאריך עברי	תאריך לועזי
י״ג סיון התשפ״ה	09/06/2025

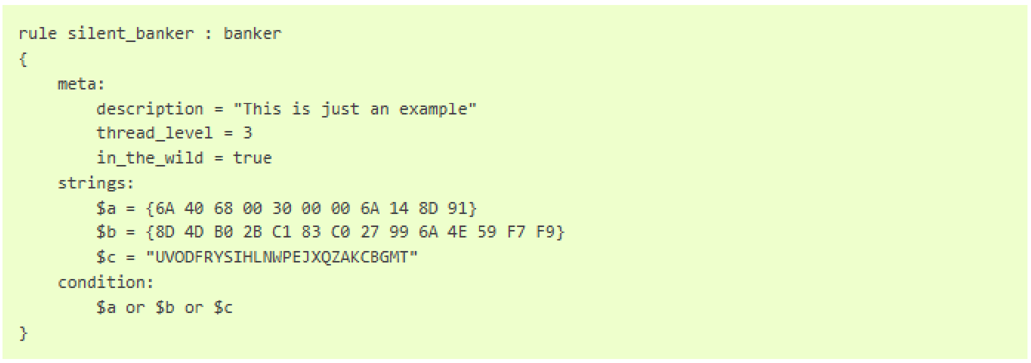
Table Of Content:

Abstract

Email systems remain a primary vector for cyber threats, posing significant security challenges for enterprise networks. These threats, which range from phishing scams to advanced persistent threats, exploit the ubiquity and essential nature of email communications to bypass traditional security measures. There is a growing consensus on the necessity for an Intrusion Detection System (IDS) and Itrusion Prevention System (IPS) that can perform real-time analysis and detection of such threats as they traverse network boundaries. Furthermore, the integration of YARA rules into IDS frameworks emerges as a crucial enhancement. YARA rules enable precise and flexible definition of malware signatures and suspicious patterns, thus bolstering the IDS capability to detect and mitigate sophisticated cyber threats effectively. There is a critical need for sophisticated detection frameworks that integrate IDS and YARA rules, aiming to enhance the defense against the evolving landscape of email-based cyber threats within enterprise environments.

1. Introduction

This project is dedicated to enhance the cybersecurity of organizational networks by developing a sophisticated detection system that combines email-specific YARA [1] signatures with the Suricata [2] Intrusion Detection System. By integrating specialized YARA signatures, the system will monitor email traffic in real-time, enabling immediate alerts and swift responses to potential cyber threats. Utilizing existing YARA rules [See Figure 1] from the "Awesome YARA" [3] database-tailored to identify specific malware patterns associated with email threats-the system will continuously update and apply these rules to effectively detect new and evolving malware. Suricata will analyze these traffic patterns with its robust rule-based detection engine, ensuring comprehensive coverage and minimizing the window of opportunity for malware to impact the network. This dual approach not only enhances network security but also provides a proactive method to combat email-based malware threats.



```
rule silent_banker : banker
{
    meta:
        description = "This is just an example"
        thread_level = 3
        in_the_wild = true
    strings:
        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
        $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"
    condition:
        $a or $b or $c
}
```

Figure 1: General YARA Rule Structure for Malware Analysis
This figure illustrates the typical structure of a YARA rule used in malware detection. It highlights key components such as the rule name, metadata, strings section for pattern matching, and condition statements that define when the rule triggers. Understanding this structure is essential for creating effective rules that identify malicious files or behaviors during automated analysis.

References:
1. YARA Documentation. Retrieved from https://yara.readthedocs.io/en/latest/
2. Suricata Documentation. Retrieved from https://docs.suricata.io/en/latest/what-is-suricata.html
3. InQuest. Awesome YARA – A curated list of YARA rules, tools, and resources. Retrieved from https://github.com/InQuest/awesome-yara

2. Literature Review

2.1 Background

2.1.1 E-Mail Cyber threats

Malicious emails pose substantial threats to businesses. Whether it is a malware attachment or a URL leading to malware, exploitation or phishing, attackers have been employing emails as an effective way to gain a foothold inside organizations of all kinds. [1]

2.1.2 Intrusion Detection System

To detect any malicious network communication, there are many security products: firewalls, intrusion detection systems, email security products, and so forth. Even host-based anti-malware solutions listen to network communication to/from the host to monitor for the presence of malware on the host. All these different security products serve different needs, and a good **defense in depth** solution needs a combination of products to protect our systems. Of all these network monitoring products, intrusion detection systems (IDS) and intrusion prevention systems (IPS) are one of the oldest and one of the most deployed solutions. There are various IDS/IPS in the market, all the way from commercial paid products to free and open source ones like Suricata and Snort.[2]

2.1.3 Suricata

Suricata is a high performance Network IDS, IPS and Network Security Monitoring engine. It is open source and owned by a community-run non-profit foundation, the Open Information Security Foundation (OISF). Suricata is developed by the OISF. [3]

2.1.4 YARA Rules

YARA is a tool aimed at helping malware researchers to identify and classify malware samples. With YARA one can create descriptions of malware families (or whatever you want to describe) based on textual or binary patterns. Each description, a.k.a. rule, consists of a set of strings and a boolean expression which determine its logic. [4]

2.2 Related Work

2.2.1 Detecting Malicious Files with YARA Rules as They Traverse the Network

The primary goal of the research is to detect malicious files traversing the network using YARA rules, the Zeek framework, and a custom Python script developed by the researcher. The focus is on implementing network-based detection using open-source tools to complement endpoint-based YARA detection, increasing overall malware detection coverage.
The research employs YARA rules, the Zeek framework, and a custom Python script to detect malicious files on the network by extracting files from network traffic and scanning them for known malware patterns. Zeek is configured to extract files for analysis, while the Python script automates scanning and generates alerts with enriched contextual data. The results demonstrate effective detection of malicious Office documents with macros and increased detection confidence through multiple rule triggers This research differs from ours as our solution focuses on varied email-based malware and Bernal's is only about Office documents.In addition, Bernal used Zeek, a network analysis tool for logging and protocol inspection and we use Suricata which focuses on high-speed intrusion detection and prevention using signatures. [5]

2.2.2 Detection of Malware by Using YARA Rules

The goal of the research is to develop an efficient method for detecting malware using the YARA tool by leveraging static analysis techniques. The study aims to provide a solution that can analyze multiple files simultaneously, generate YARA rules based on specific patterns, and produce reports for each detected malware instance, ultimately reducing detection time and improving accuracy. The proposed approach uses multiple YARA rules to detect malware through static analysis of file characteristics such as unique patterns, strings, and hexadecimal values. The methodology focuses on pattern-based detection, testing YARA rules against known malware signatures, and evaluating their efficiency. The system enables concurrent rule execution and individual result reporting for optimized accuracy. Results show that the YARA-based detection system effectively identifies various malware types, including embedded scripts and obfuscated code, with minimal processing time. The study highlights that static analysis with optimized rules enhances detection efficiency. Although, when it comes to malware that can spread across an entire organization through emails then it is not sufficient and it needs a dynamic detection using an IDS like our project. [6]

2.2.3 Evaluating Automatically Generated YARA Rules and Enhancing Their Effectiveness

The research paper evaluates the efficacy of YARA rules generated by three distinct tools-yarGen, yaraGenerator, and yabin-in detecting malware, and explores the enhancement of these tools through the integration of a fuzzy hashing method. The study performs detailed experiments on various samples of malware and goodware, illustrating that the application of fuzzy hashing can significantly improve the detection capabilities of these automatically generated YARA rules. The results highlight that, despite the inherent strengths and weaknesses of each tool, incorporating fuzzy hashing leads to more accurate and reliable malware detection outcomes, thereby optimizing cybersecurity measures.

In contrast, our project is centered around a practical application of YARA signatures within an organizational setting, integrating these signatures with the Suricata engine to detect email-based malware threats. It describes the setup of a real-world system aimed at enhancing organizational network security and details the system׳s operational functionality without delving into the underlying technical enhancement methods like fuzzy hashing.[7]

2.2.4 A YARA-based approach for detecting cyber security attack types

This paper from Firat University details extensive research on using YARA rules for cybersecurity threat detection. It centers on setting up a virtual environment to apply and defend against various cyber attack methods on Windows and Linux systems, with a focus on forensic informatics and incident response. The study evaluates how YARA, coupled with other manual and open-source methods, can detect malware in these operating systems. It discusses creating effective YARA rules, simulating cyberattacks, analyzing their impact, and refining incident response strategies to enhance organizational cybersecurity measures. This comprehensive approach helps in understanding the dynamics of cyber threats and the development of robust defenses.
They focus on using YARA rules to simulate, detect, and analyze cyberattacks within a virtual environment, providing a detailed examination of cyber threat dynamics and defense mechanisms which is in contrast to our project that primarily deals with implementing YARA signatures in an organizational context to improve the detection of email-based malware threats using Suricata. While the journal paper delves into theoretical and practical applications across operating systems with a broader scope on cyber defense, this project is more application-specific, targeting email security in organizational networks.[8]

3. Methods

3.1 Virtual Test Environment Design

To effectively simulate and test the detection of cyber threats within a network, it is essential to build a synthetic organizational environment [See Figure 2] that reflects the behavior of a real enterprise. This environment will be created using Proxmox, an open-source virtualization platform that allows for flexible and isolated deployment of multiple virtual machines, each assigned a specific role. The setup will include three main endpoints. The first machine, based on Kali Linux, will serve as the security monitoring node. It will run Suricata, an intrusion detection system used to inspect and analyze network traffic in real time. In addition, a Python-based detection script will be deployed on this machine to scan files using YARA rules and possibly enrich the analysis by querying services like VirusTotal. This machine will effectively replicate the role of a security analyst's system within a corporate network. The second and third machines will both run Windows operating systems and simulate typical email-based interactions in an organization. One will act as the sender, generating and transmitting emails with various attachments, some of which may be intentionally crafted to mimic malicious payloads. The other machine will function as the receiver, opening and interacting with the incoming emails and their contents. This setup is designed to simulate realistic scenarios involving internal communication and potential threat vectors, such as phishing or malware-laden attachments. By observing the traffic between these endpoints, and analyzing the behavior triggered by email exchanges, the synthetic network will provide a practical testbed for evaluating how well the detection tools can identify and flag suspicious activity, helping to refine defenses and improve incident response strategies.

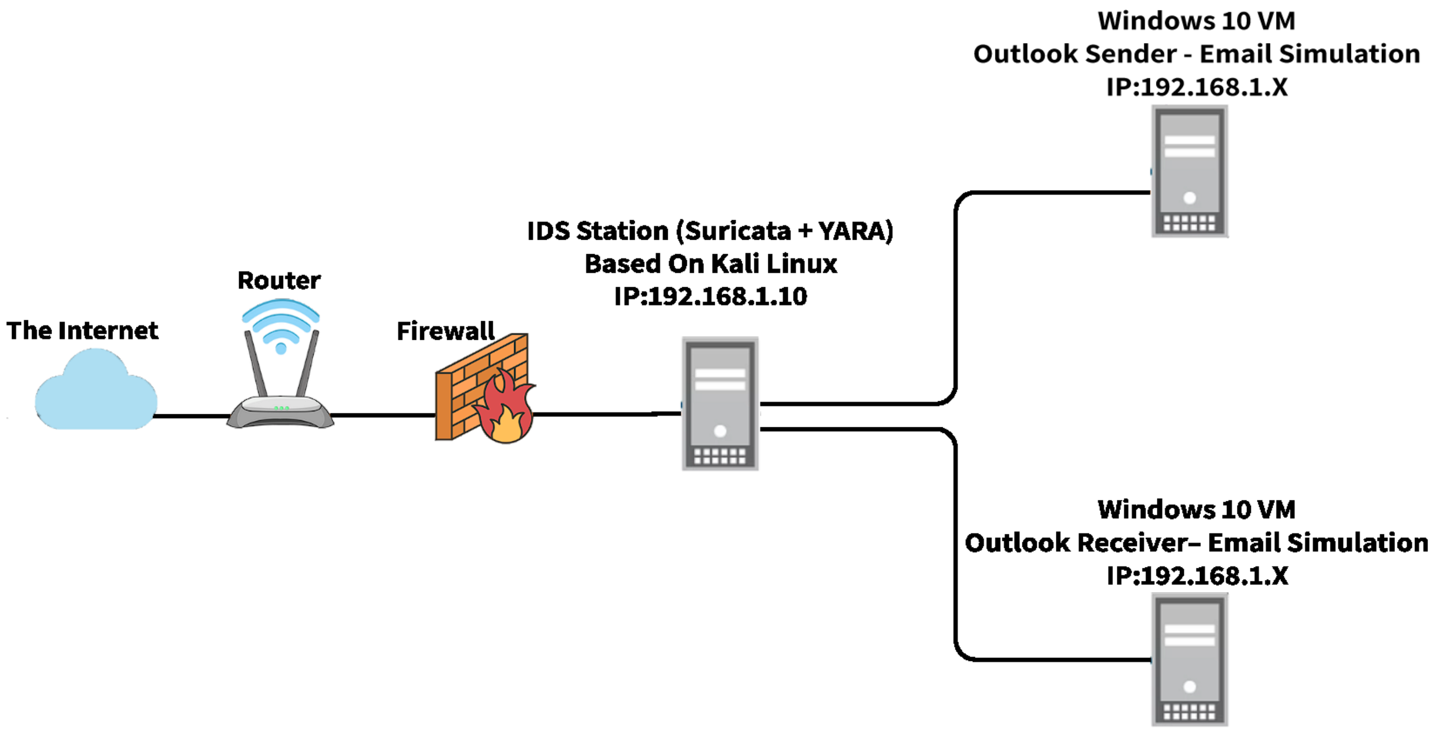


Figure 2: Network Architecture: Cloud–Router–Firewall–IDS–Windows Clients
This diagram illustrates the overall network architecture of the test environment. It depicts the flow of data starting from the external cloud, passing through the router and firewall, reaching the Intrusion Detection System (IDS), and finally arriving at Windows client machines. This setup is designed to simulate realistic enterprise network traffic and security layers for testing email-based threat detection mechanisms.

3.2 Infrastructure Deployment with Proxmox

The entire network infrastructure was deployed using Proxmox Virtual Environment [See figure 3], an open-source virtualization platform. Proxmox enabled us to create an isolated, enterprise-like setup with multiple interconnected virtual machines, each serving a specific role such as client, mail server, or intrusion detection system. Its flexibility and stability allowed for seamless traffic simulation, resource management, and reproducible test conditions, making it an ideal foundation for developing and evaluating our detection system. [10]

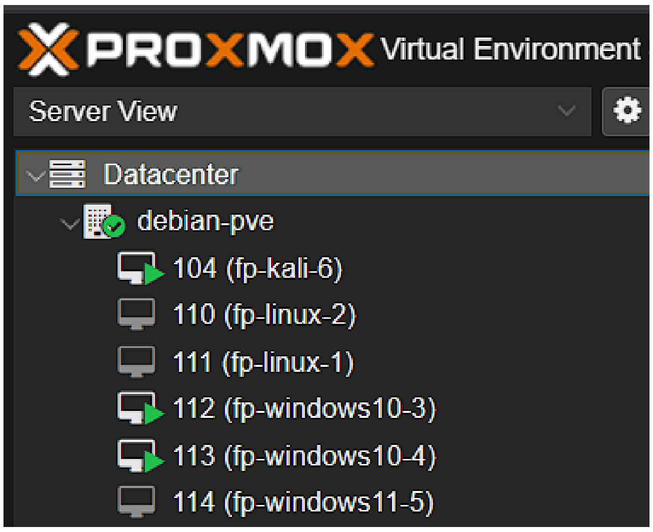


Figure 3: Proxmox Environment with IDS and Clients
This figure illustrates the virtualized testbed setup created using Proxmox. It depicts the Intrusion Detection System (IDS) and multiple client machines running within isolated virtual machines. This environment enables controlled simulation of network traffic, including email exchanges and security monitoring, facilitating realistic testing of threat detection workflows.

3.3 Minimizing False Positives and False Negatives

Effectively handling false-positive detections is a critical aspect of any cybersecurity detection system. While identifying and flagging malicious activity is the main objective, it is equally important to ensure that benign traffic is not mistakenly classified as a threat. High rates of false positives can lead to unnecessary disruptions in normal business operations, waste analysts' time, and ultimately reduce the efficiency and credibility of the detection system. To address this, we will focus on the careful fine-tuning of our YARA rules. This process involves adjusting the rule conditions, refining the patterns used for matching, and incorporating contextual metadata to make the rules more precise. Rather than relying solely on broad or generic indicators, we aim to define more targeted signatures that minimize overlaps with legitimate content. Validation will be carried out in a controlled testing environment that mirrors the structure and behavior of a real-world network. In this environment, we will introduce both known malicious samples and a wide range of benign files and activities. By monitoring the system's responses, we can measure the accuracy of our YARA rules, identify sources of false positives, and iteratively adjust our detection logic. This approach ensures that our detection system remains both reliable and practical-able to catch real threats without interfering with normal, harmless activity. The ultimate goal is to achieve a balanced setup that maintains strong security coverage while minimizing unnecessary alerts and preserving the usability of the network.

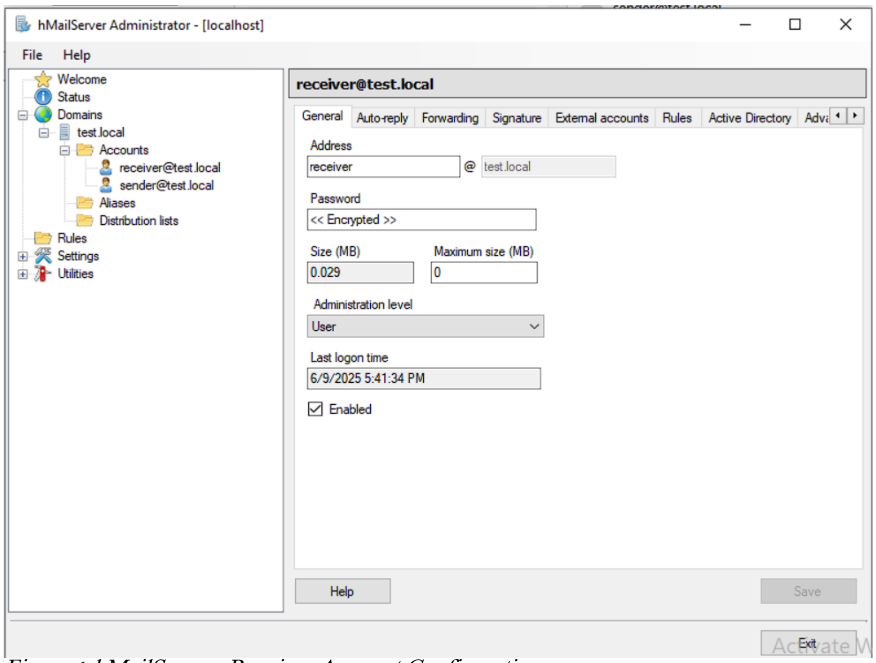
3.4 Email Transmission with hMailServer

To facilitate realistic email-based communication within the virtual environment, we configured hMailServer [9] as the internal mail transfer agent (MTA). This lightweight and flexible mail server enabled us to simulate enterprise-level email transmissions, including the exchange of attachments between virtual machines. Within our setup, we created specific user accounts to act as the sender and receiver, mimicking typical roles in an organization. These accounts were used to generate SMTP traffic that could be closely monitored by Suricata, allowing us to test the effectiveness of our YARA-based detection system.

The hMailServer was configured with a local domain, and SMTP, POP3, and IMAP services were enabled to support full mail delivery and retrieval within the test network [See figure 6]. Authentication was enforced for outgoing messages, and realistic account credentials were assigned to simulate employee-like behavior [See figure 4 & 5]. Using standard email clients such as Mozilla Thunderbird, we were able to compose and send messages that reflected real-world formats, including HTML content, inline links, and base64-encoded attachments.

To enrich our testing scenarios, a range of emails were transmitted - some benign, others deliberately crafted to simulate threats such as phishing attacks and malicious file attachments. These messages helped trigger detection events in Suricata, allowing us to observe the flow of potentially harmful content across the network and analyze how our rule sets performed. The setup also allowed for quick modification of email content and repeated testing under varied threat conditions.

The integration of hMailServer was therefore crucial - not only for generating realistic email traffic but also for providing a controlled, testable platform to evaluate how our detection tools respond to real-time threats embedded in email communications [9].



receiver@test.local

Address: receiver @ test.local

Password: << Encrypted >>

Size (MB): 0.029

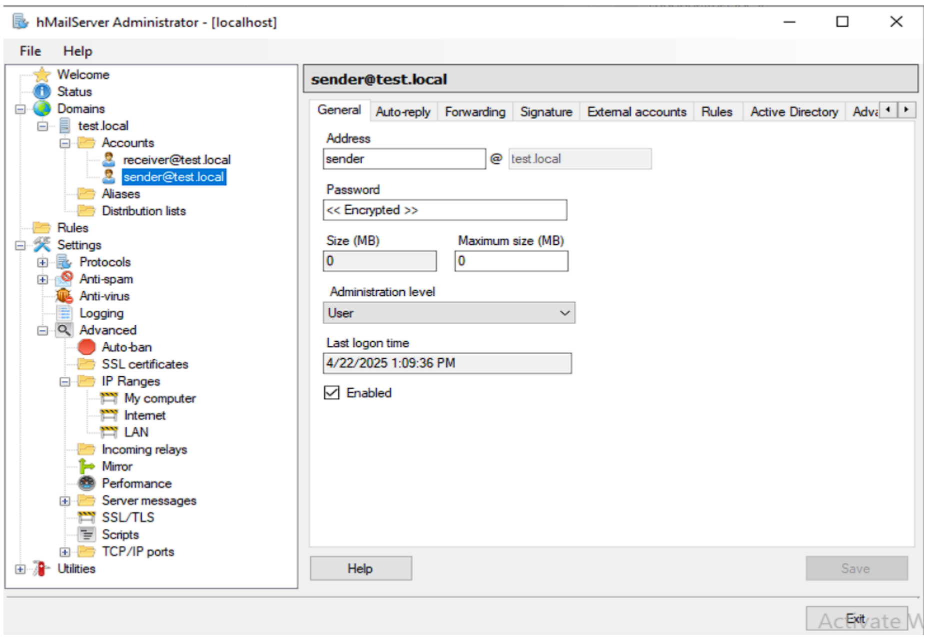
Maximum size (MB): 0

Administration level: User

Last logon time: 6/9/2025 5:41:34 PM

☑ Enabled

Figure 4: hMailServer: Receiver Account Configuration
This figure shows the configuration of a receiver email account in hMailServer. It includes settings such as the account address, associated domain, password, and storage parameters. This setup is essential for simulating the reception of emails in a controlled environment, allowing the system to receive test messages that may include phishing content or malicious attachments for analysis.



sender@test.local

Address: sender @ test.local

Password: << Encrypted >>

Size (MB): 0

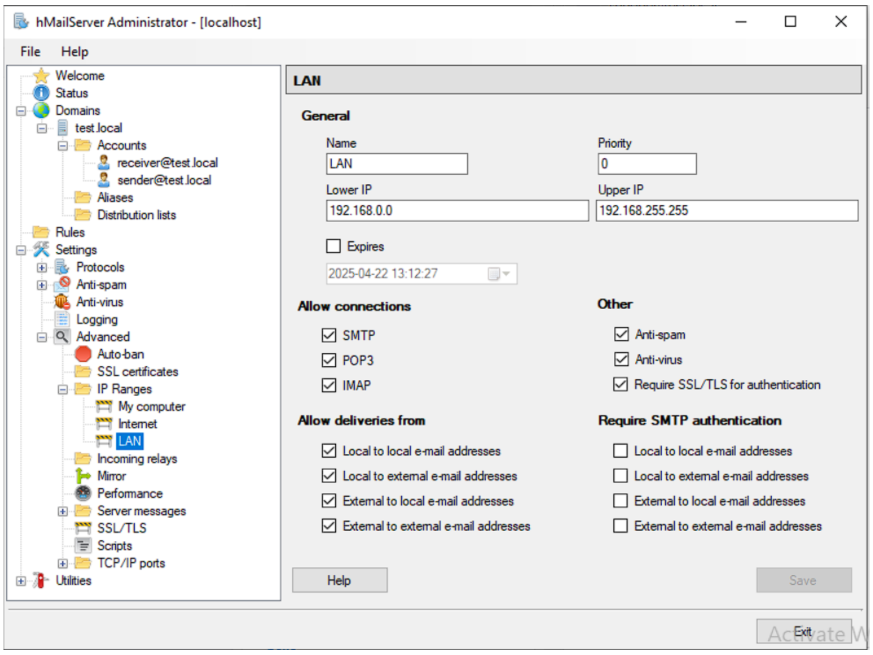
Maximum size (MB): 0

Administration level: User

Last logon time: 4/22/2025 1:09:36 PM

☑ Enabled

Figure 5: hMailServer: Sender Account Configuration
This figure displays the configuration settings for a sender account in hMailServer. It includes the email address, domain, and authentication credentials required to send messages from the simulated environment. This setup enables the generation of test emails containing various payloads - such as attachments or phishing content - to evaluate detection mechanisms and simulate real-world attack scenarios within the testbed.



hMailServer Administrator - [localhost]

File Help

- Welcome
- Status
- Domains
  - test.local
    - Accounts
      - receiver@test.local
      - sender@test.local
    - Aliases
    - Distribution lists
- Rules
- Settings
  - Protocols
  - Anti-spam
  - Anti-virus
  - Logging
  - Advanced
    - Auto-ban
    - SSL certificates
    - IP Ranges
      - My computer
      - Internet
      - LAN
    - Incoming relays
    - Mirror
    - Performance
    - Server messages
    - SSL/TLS
    - Scripts
    - TCP/IP ports
- Utilities

LAN

General

Name	LAN
Priority	0
Lower IP	192.168.0.0
Upper IP	192.168.255.255

☐ Expires 2025-04-22 13:12:27

Allow connections
- ☑ SMTP
- ☑ POP3
- ☑ IMAP

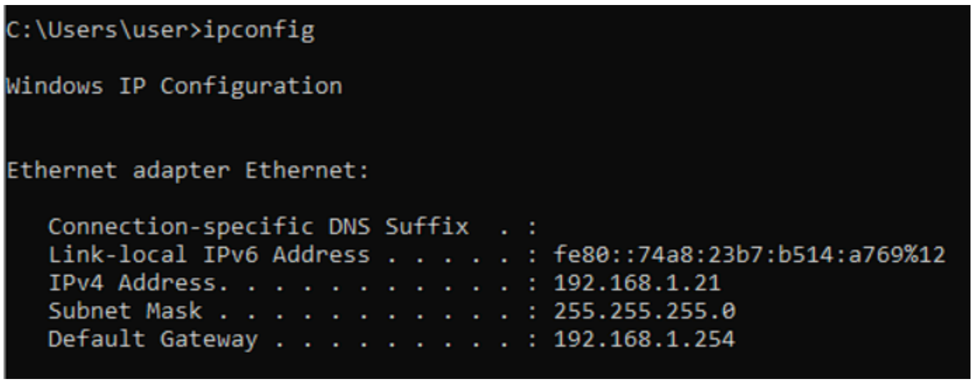
Other
- ☑ Anti-spam
- ☑ Anti-virus
- ☑ Require SSL/TLS for authentication

Allow deliveries from
- ☑ Local to local e-mail addresses
- ☑ Local to external e-mail addresses
- ☑ External to local e-mail addresses
- ☑ External to external e-mail addresses

Require SMTP authentication
- ☐ Local to local e-mail addresses
- ☐ Local to external e-mail addresses
- ☐ External to local e-mail addresses
- ☐ External to external e-mail addresses

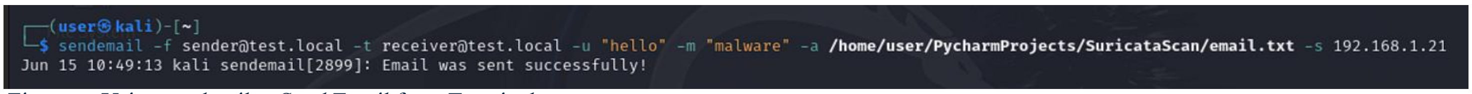
Help Save Exit

Figure 6: hMailServer: LAN Configuration Settings
This figure illustrates the LAN configuration settings within hMailServer, including the binding to a specific local IP address (e.g., 192.168.x.x) and port configuration. These settings ensure proper communication between virtual machines on the internal network, enabling reliable email transmission and reception during testing scenarios.



```
C:\Users\user>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::74a8:23b7:b514:a769%12
   IPv4 Address. . . . . . . . . . . : 192.168.1.21
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.254
```

Figure 7: hMailServer Network Address Confirmed as 192.168.1.21
This figure confirms the network address of the hMailServer instance as 192.168.1.21 within the local test environment. Verifying this IP address is crucial to ensure accurate email routing between virtual machines and to allow other components, such as the email client and Suricata, to interact correctly with the mail server during simulated attack scenarios.

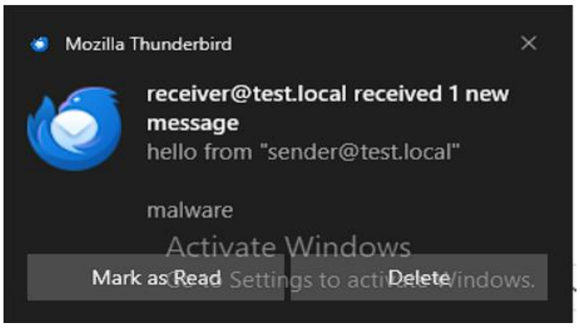


```
┌──(user㉿kali)-[~]
└─$ sendemail -f sender@test.local -t receiver@test.local -u "hello" -m "malware" -a /home/user/PycharmProjects/SuricataScan/email.txt -s 192.168.1.21
Jun 15 10:49:13 kali sendemail[2899]: Email was sent successfully!
```

Figure 8: Using sendmail to Send Email from Terminal
This figure demonstrates the use of the sendmail command-line utility to send an email directly from the terminal. It simulates an attacker or automated script sending an email with a malicious payload to the target environment. This method allows precise control over the message content, headers, and attachments, making it ideal for testing detection mechanisms in a controlled setting.

3.5 Email Client Simulation with Thunderbird

To simulate realistic user interaction with email content, we configured Mozilla Thunderbird as the email client on the Windows machines. Thunderbird provided a reliable and user-friendly interface for sending, receiving, and opening emails, including those with file attachments [See figure 9]. Its support for standard SMTP and IMAP protocols ensured smooth integration with hMailServer, allowing for accurate testing of email-based threat scenarios. Using Thunderbird allowed us to replicate typical user behavior - such as opening suspicious attachments or viewing phishing messages—which was essential for triggering network activity that could be analyzed by Suricata and scanned with YARA. This step added an important layer of realism to our testing workflow and helped validate the system's ability to detect threats within genuine user interactions.


Mozilla Thunderbird

receiver@test.local received 1 new message

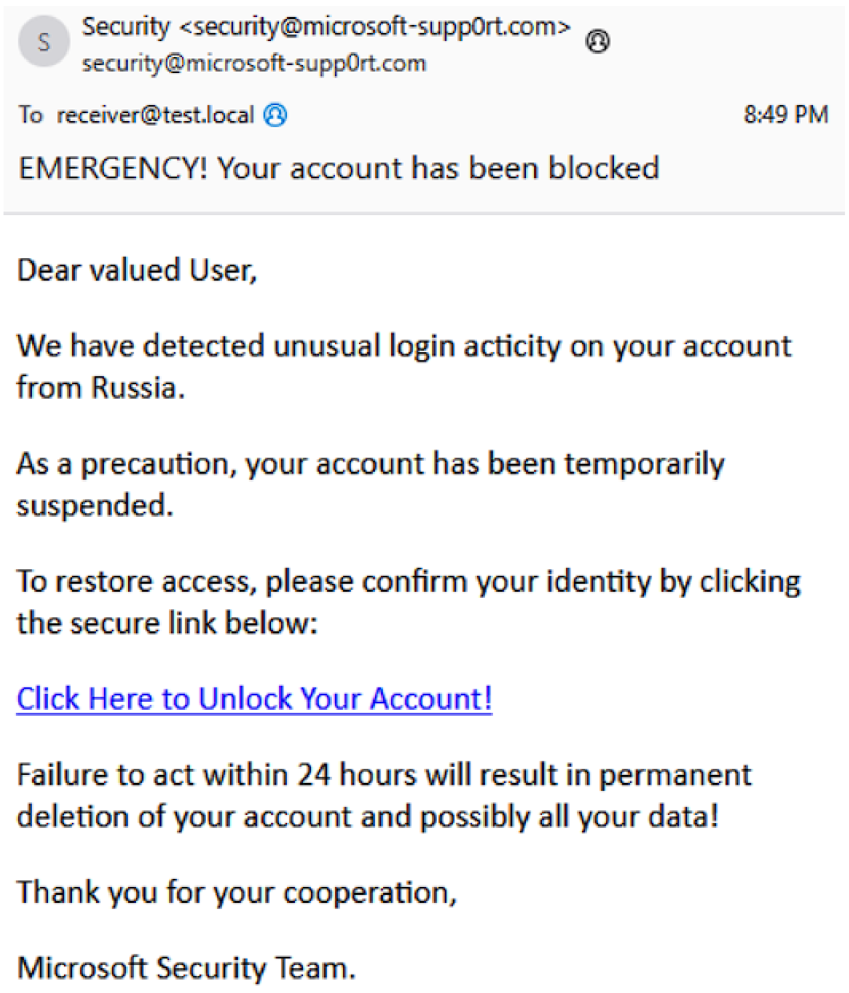
hello from "sender@test.local"

malware

*Figure 9: Email Successfully Transmitted and Received by the server
This figure confirms that the crafted email was successfully delivered to the hMailServer and received by the target account. It verifies the proper functioning of the email transmission pipeline within the simulated environment, including message delivery, server configuration, and readiness for further analysis by detection components such as Suricata and the email client.*

As part of the simulation, a phishing email [See figure 10] was sent and received within the hMailServer environment. Upon inspection, we can observe that the sender's email address is suspicious—using a misleading domain name such as microsft-supp0rt.com, which closely imitates a legitimate service but contains subtle misspellings. The email content includes urgent language designed to pressure the recipient into immediate action, such as claims of account compromise or threats of deletion. Additionally, it contains a clickable link that purports to resolve the issue but actually directs the user to a potentially malicious external site. These are all common characteristics used in phishing attacks to deceive users and extract sensitive information.

For comparison, a legitimate email was also sent within the same environment [see Figure 11]. In contrast to the phishing message, the legitimate email uses a recognizable and correctly spelled domain name, neutral language, and clearly defined contact details. It contains no deceptive urgency or misleading links, and the overall structure aligns with the communication standards of professional or personal correspondence. This contrast helped validate the system’s ability to distinguish between benign and malicious email traffic during testing.



Security <security@microsoft-supp0rt.com>
security@microsoft-supp0rt.com

To receiver@test.local 8:49 PM

EMERGENCY! Your account has been blocked

Dear valued User,

We have detected unusual login acticity on your account from Russia.

As a precaution, your account has been temporarily suspended.

To restore access, please confirm your identity by clicking the secure link below:

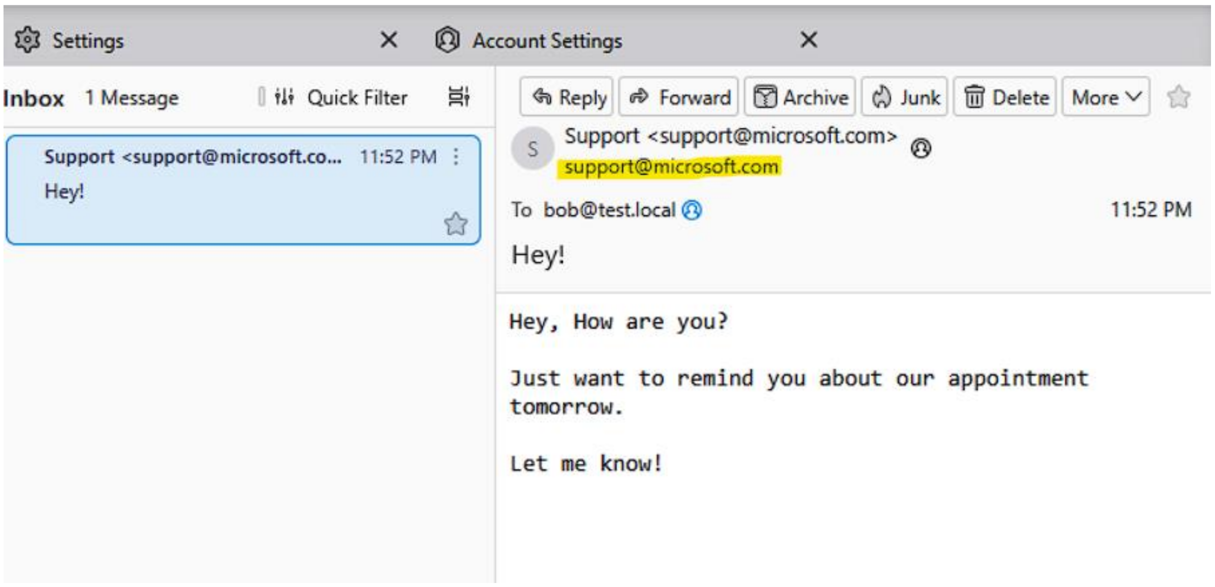
Click Here to Unlock Your Account!

Failure to act within 24 hours will result in permanent deletion of your account and possibly all your data!

Thank you for your cooperation,

Microsoft Security Team.

Figure 10: Suspicious Phishing Email Displayed in Thunderbird using hMailServer
This figure shows a phishing email received through the hMailServer and displayed in the Thunderbird email client. It demonstrates the simulation of a user encountering a potentially malicious email, including typical phishing characteristics such as deceptive sender information, suspicious links, or attachments.



Support <support@microsoft.co... 11:52 PM
Hey!

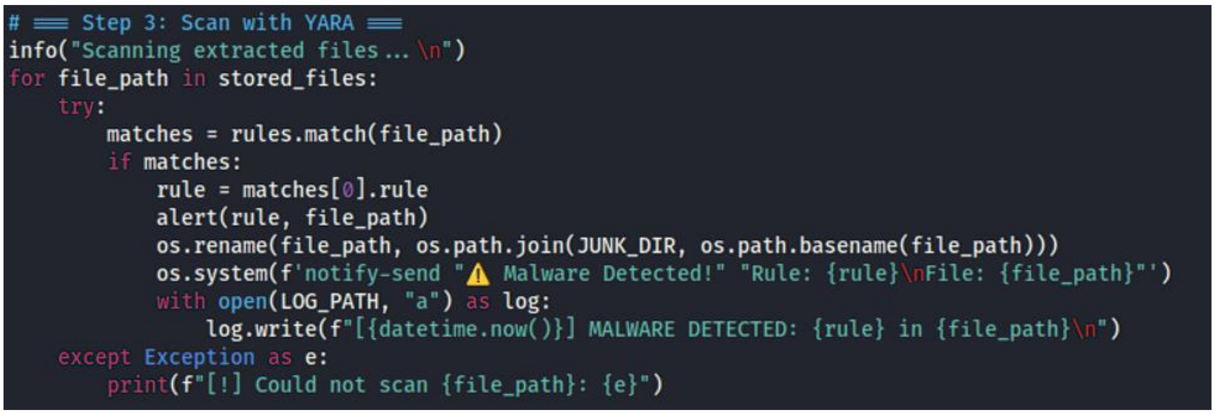
Support <support@microsoft.com>
support@microsoft.com
To bob@test.local 11:52 PM
Hey!

```
Hey, How are you?

Just want to remind you about our appointment
tomorrow.

Let me know!
```

Figure 11: Legitimate Email Sent via Thunderbird
This figure illustrates a legitimate email composed and sent through the Thunderbird email client. It serves as a baseline for normal email traffic within the test environment, helping to differentiate benign communications from malicious or phishing attempts during detection and analysis processes.

3.6 Automated Detection with Python Script

A custom-built Python script [Appendix A] was developed to automate the detection process by bridging Suricata's network monitoring with YARA's file-scanning capabilities. The script continuously parses Suricata's eve.json log file, which contains a rich stream of structured data representing various types of network events, including file transfers, alerts, and SMTP communications. It filters and extracts only the relevant entries—particularly those involving file attachments or payloads—and retrieves associated metadata such as filenames, MIME types, SHA-256 hashes, and source/destination IP addresses. Once the relevant data is isolated, the script locates the corresponding files that were extracted by Suricata, and proceeds to scan them using a set of YARA rules tailored for email-based threats. This includes rules designed to detect ransomware lures, phishing attachments, suspicious scripts, and encoded payloads. The script supports both broad scanning and rule-specific targeting, depending on the context of the traffic or testing scenario. Robust error handling is implemented throughout the script to ensure that malformed entries, missing files, or incomplete metadata do not interrupt the detection workflow. The script logs both detection results and operational details, such as matched rule identifiers, and file



```python
# == Step 3: Scan with YARA ==
info("Scanning extracted files... \n")
for file_path in stored_files:
    try:
        matches = rules.match(file_path)
        if matches:
            rule = matches[0].rule
            alert(rule, file_path)
            os.rename(file_path, os.path.join(JUNK_DIR, os.path.basename(file_path)))
            os.system(f'notify-send "⚠ Malware Detected!" "Rule: {rule}\nFile: {file_path}"')
            with open(LOG_PATH, "a") as log:
                log.write(f"[{datetime.now()}] MALWARE DETECTED: {rule} in {file_path}\n")
    except Exception as e:
        print(f"[!] Could not scan {file_path}: {e}")
```

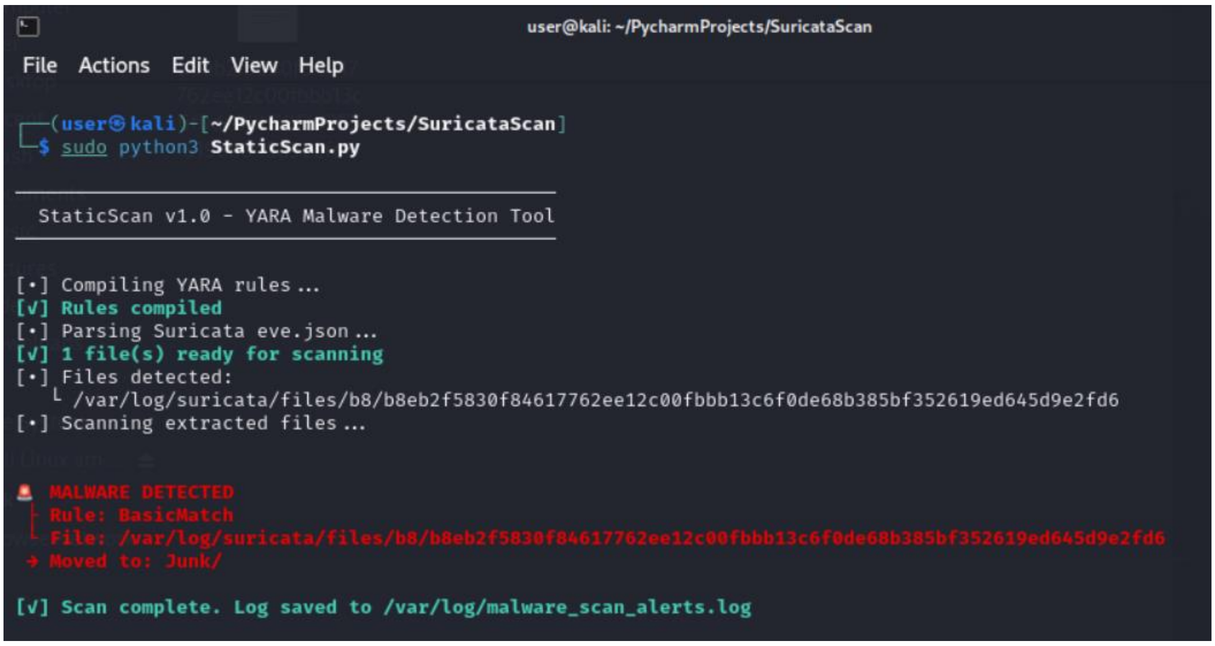
Figure 12: Script Snippet for Matching YARA Rules

paths, in a structured output that can be further processed or visualized. In its current form, the script is modular and easily extendable. It lays the foundation for additional features, such as querying online threat intelligence platforms (e.g., VirusTotal), alert forwarding, or real-time scanning capabilities. Overall, it plays a pivotal role in operationalizing threat detection, turning passive traffic monitoring into actionable insights.

4. Results

At this stage, the system demonstrates a high level of operational readiness, with the core components functioning cohesively. The virtual organizational network, built in Proxmox, has proven to be a stable and realistic testbed for simulating common enterprise scenarios, particularly email-based interactions. Regular traffic is being successfully generated between the machines, and Suricata is consistently capturing network events, including file transfers and SMTP communication, without any stability or logging issues.

The Python script responsible for parsing Suricata's JSON logs and scanning files with YARA rules is performing as expected. It processes relevant log entries efficiently, extracts critical data fields such as file names, types, and hashes, and scans the corresponding files with the active rule set. The script handles both expected and edge-case inputs without crashing or producing inconsistent results.


user@kali: ~/PycharmProjects/SuricataScan
File Actions Edit View Help

```
┌──(user㉿kali)-[~/PycharmProjects/SuricataScan]
└─$ sudo python3 StaticScan.py

  StaticScan v1.0 - YARA Malware Detection Tool

[•] Compiling YARA rules ...
[✓] Rules compiled
[•] Parsing Suricata eve.json ...
[✓] 1 file(s) ready for scanning
[•] Files detected:
   └ /var/log/suricata/files/b8/b8eb2f5830f84617762ee12c00fbbb13c6f0de68b385bf352619ed645d9e2fd6
[•] Scanning extracted files ...

🚨 MALWARE DETECTED
 ├ Rule: BasicMatch
 └ File: /var/log/suricata/files/b8/b8eb2f5830f84617762ee12c00fbbb13c6f0de68b385bf352619ed645d9e2fd6
 → Moved to: Junk/

[✓] Scan complete. Log saved to /var/log/malware_scan_alerts.log
```

Figure 3: Detection of Malware Displayed in Terminal
This figure shows the terminal output from the detection system, indicating successful identification of malware within scanned files. The display includes detailed information such as matched YARA rule names, file paths, and actions taken. Notably, the detection script can be run by a regular user - it does not require root privileges; highlighting its ease of deployment and security.

The YARA rules themselves have been iteratively developed and tested across various benign and malicious file samples. Early versions of the rules initially resulted in some false positives, but after successive rounds of refinement-adjusting string patterns, improving conditions, and narrowing rule scope-the rate of false alerts has dropped significantly. The rules are now capable of identifying specific malware traits with a reasonable balance between sensitivity and precision.

In controlled testing scenarios, simulated phishing and malware-laden email attachments were successfully detected by the system. Suricata flagged the relevant traffic and extracted files, which were then scanned and correctly identified by the YARA engine. These results validate both the environment configuration and the practical effectiveness of the detection workflow.

Additionally, the modular structure of the solution-separating detection, logging, and scanning-has proven useful for debugging and future extensions. Logs show clear traceability from the network activity to the detection result, which will be valuable for future correlation and visualization.

5. Discussion

The results achieved so far demonstrate the system׳s robustness and the effectiveness of the overall detection pipeline. The stable deployment of the virtual network within Proxmox provided a realistic and controlled environment for simulating enterprise-grade email interactions. This infrastructure enabled reliable traffic generation and consistent packet capture by Suricata, validating the suitability of the testbed for cybersecurity experimentation.

The successful parsing of Suricata's JSON logs and subsequent YARA scanning via a custom Python script highlights the practicality of integrating lightweight detection tools into a larger monitoring system. The script's resilience against edge-case inputs and its ability to extract, scan, and correlate log data efficiently reflect a strong foundation for automation and scalability.

The iterative refinement of YARA rules emerged as a critical phase in improving detection accuracy. While initial versions suffered from false positives-a common challenge in static detection-subsequent tuning significantly improved precision. This underscores the importance of continual testing against both malicious and benign samples to maintain reliability in real-world conditions.

Controlled tests involving simulated phishing emails and malware-laden attachments further validated the system's end-to-end detection capability. Suricata accurately captured the traffic and files, while the YARA engine successfully flagged the threats, confirming the workflow's practical viability. These results not only prove the technical soundness of the components but also illustrate their synergy when integrated.

Finally, the modular architecture-clearly separating capture, analysis, and detection-proved beneficial for both debugging and extensibility. The clean traceability from network activity to detection outcome is especially valuable for future integration with visualization tools or alerting systems, and sets a solid groundwork for continued development.

6. Conclusion

This project has achieved its primary goal of designing and implementing an email-based threat detection system that leverages YARA rules and Suricata within a realistic, virtualized network environment. By deploying a controlled organizational setup in Proxmox, we were able to simulate typical enterprise communication scenarios, particularly focusing on SMTP-based interactions and file transfers. The environment proved stable, reproducible, and effective for capturing relevant traffic without performance or logging issues, making it a strong foundation for future experimentation and expansion.

The integration between Suricata and the Python-based scanning module demonstrated both reliability and efficiency. Suricata consistently captured detailed network events and generated structured logs, which were accurately parsed by the script to extract critical fields such as file hashes, types, and names. The ability of the script to handle both regular and edge-case entries without crashing or producing inconsistent output confirms its robustness and adaptability. It also reinforces the system's potential for automation and scalability in more complex deployments.

The development of YARA rules was an iterative and insightful process. Initial versions of the rules produced several false positives; an expected outcome in signature-based detection systems. However, through repeated testing and refinement, including adjustments to string selection, condition logic, and scope narrowing, the ruleset matured significantly. The final set of rules displayed a strong balance between sensitivity and precision, successfully identifying phishing attempts and malware-laced attachments in controlled tests, while minimizing false alerts on benign samples.

These results not only validate the operational readiness of the system but also demonstrate its practical viability. The detection workflow - from network capture to threat identification was consistently accurate in identifying malicious content embedded in email traffic. Furthermore, the modular structure of the system has proven to be a key advantage. By decoupling network monitoring, log parsing, and file scanning, each component can be updated or replaced independently, which simplifies debugging, enables clearer traceability, and facilitates future development such as real-time dashboards, alerting mechanisms, or threat intelligence integration.

7. Future Work

While the system has shown strong functionality and reliability in detecting email-based threats, there are several directions in which it can be meaningfully expanded. A primary enhancement would be the development of a real-time visualization dashboard, which would provide a centralized view of network activity, detections, and threat indicators. Such a component would not only make analysis more intuitive for security analysts but also aid in quickly identifying attack patterns and responding to incidents with greater efficiency.

The content filter, based on scanning with YARA rules, generally exhibits moderate computational demand and good efficiency. The load depends mainly on the size of the scanned files and the number of active rules simultaneously processed. In our testing, scans completed within reasonable time frames (a few seconds per file) and could be run in the background on standard machines without requiring elevated privileges or high-end hardware.

Another significant area for improvement lies in the expansion and diversification of the YARA rule set. While current rules were sufficient for detecting specific threats in our controlled test scenarios, a broader and more dynamic library is needed to address the fast-evolving nature of malware and phishing campaigns. This includes coverage for polymorphic malware, document-based exploits, script-based threats, and emerging obfuscation techniques.

To further enhance detection and reduce the manual workload associated with writing and maintaining rules, AI and machine learning techniques could be leveraged in the future. By analyzing large volumes of threat data, machine learning models could assist in automatically generating YARA rule candidates, clustering similar malware families, or suggesting improvements to existing rules. Such an approach could accelerate the rule development lifecycle and improve coverage against novel threats.

In addition, the project could benefit from integration with broader security ecosystems, such as SIEM platforms, to support centralized logging, correlation with other threat intelligence sources, and automated alerting. The modular nature of the system makes it well-suited for such integration.

Finally, future testing could include more complex and realistic attack simulations, incorporating encrypted traffic, user interaction models (e.g., simulated email clicks), and layered attacks. These scenarios would further validate the system's resilience and adaptability in real-world threat environments.

8. References

1. Zeng, Y. G. (2017). Identifying email threats using predictive analysis. In 2017 International Conference on Cyber Security and Protection of Digital Services (Cyber Security). IEEE. https://doi.org/10.1109/CyberSecPODS.2017.8074848
2. Mohanta, A., & Saldanha, A. (2020). *IDS/IPS and Snort/Suricata rule writing*. In *Malware Analysis and Detection Engineering*. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-6193-4_23
3. *Suricata Documentation*. Retrieved from https://docs.suricata.io/en/latest/what-is-suricata.html
4. *YARA Documentation*. Retrieved from https://yara.readthedocs.io/en/latest/
5. Bernal Michelena, D. (2019, August). *Detecting Malicious Files with YARA Rules as They Traverse the Network*. SCILabs. Retrieved from https://i.blackhat.com/USA-19/Wednesday/us-19-Bernal-Detecting-Malicious-Files-With-YARA-Rules-As-They-Traverse-the-Network-wp.pdf
6. Mahdi, R. H., & Trabelsi, H. (2024, April). Detection of Malware by Using YARA Rules. IEEE. Retrieved from https://ieeexplore.ieee.org/document/10549308
7. Naik, N., Jenkins, P., Cooke, R., Gillett, J., & Jin, Y. (2020, December 20). Evaluating Automatically Generated YARA Rules and Enhancing Their Effectiveness. IEEE. Retrieved from http://ieeexplore.ieee.org/document/9308179/authors#authors
8. Yıldırım, K., Demir, M. E., Keleş, T., Yıldız, A. M., Doğan, Ş., & Tuncer, T. (2023, May). A YARA-based approach for detecting cyber security attack types. Retrieved from https://dergipark.org.tr/en/download/article-file/3218618
9. hMailServer Documentation. Retrieved from https://www.hmailserver.com/documentation/latest/
10. Proxmox Documentation. Retrieved from https://pve.proxmox.com/pve-docs/

9. List of figures

10. Appendix

```
Appendix A: Python Malware Detection Script
import os
from datetime import datetime
import yara

# === Configuration ===
YARA_RULES_PATH = "/var/log/suricata/pcap/data/yara_rules/"
EVE_JSON_PATH = "/var/log/suricata/eve.json"
SURICATA_FILES_DIR = "/var/log/suricata/files/"
LOG_PATH = "/var/log/malware_scan_alerts.log"
JUNK_DIR = "Junk"
os.makedirs(JUNK_DIR, exist_ok=True)

# === Utilities ===
def info(msg): print(f"[•] {msg}")
def success(msg): print(f"\033[1;32m[✓] {msg}\033[0m")
def alert(rule, path):
    print(f"\n\033[1;31m🚨 MALWARE DETECTED\n ├ Rule: {rule}\n └ File: {path}\n → Moved to: {JUNK_DIR}/\033[0m\n")

# === Banner ===
print("───────────────────────────────────────────────")
print("  StaticScan v1.0 - YARA Malware Detection Tool")
print("───────────────────────────────────────────────\n")

# === Step 1: Compile YARA Rules ===
info("Compiling YARA rules...")
rules = yara.compile(filepaths={
    f: os.path.join(YARA_RULES_PATH, f)
    for f in os.listdir(YARA_RULES_PATH)
    if f.endswith((".yar", ".yara"))
})
success("Rules compiled")

# === Step 2: Parse eve.json for stored files ===
info("Parsing Suricata eve.json...")
stored_files = []
with open(EVE_JSON_PATH, "r") as f:
    for line in f:
        if '"stored":true' in line and '"sha256":"' in line:
            try:
                sha = line.split('"sha256":"')[1].split('"')[0]
                folder = sha[:2]
                file_path = os.path.join(SURICATA_FILES_DIR, folder, sha)
                if os.path.exists(file_path):
                    stored_files.append(file_path)
            except:
                continue
success(f"{len(stored_files)} file(s) ready for scanning")

if stored_files:
    info("Files detected:")
    for f in stored_files:
        print(f"   └ {f}")

# === Step 3: Scan with YARA ===
info("Scanning extracted files...\n")
for file_path in stored_files:
    try:
        matches = rules.match(file_path)
        if matches:
            rule = matches[0].rule
            alert(rule, file_path)
            os.rename(file_path, os.path.join(JUNK_DIR, os.path.basename(file_path)))
            os.system(f'notify-send "⚠ Malware Detected!" "Rule: {rule}\nFile: {file_path}"')
            with open(LOG_PATH, "a") as log:
                log.write(f"[{datetime.now()}] MALWARE DETECTED: {rule} in {file_path}\n")
    except Exception as e:
        print(f"[!] Could not scan {file_path}: {e}")

# === Done ===
success("Scan complete. Log saved to /var/log/malware_scan_alerts.log")
```