

# Monitoring Remote Access Trojan (RAT) Communications Using Wireshark and Suricata

Nurhashikin Mohd Salleh<sup>1\*</sup>, Muhammad Amin bin Jamaluddin<sup>2</sup>, Siti Rahayu Selamat<sup>3</sup>, Aimi Liyana Amir<sup>4</sup>,  
Mohd Taufik Mishan<sup>5</sup>

<sup>1,2,3</sup>Faculty of Artificial Intelligence and Cyber Security, Universiti Teknikal Malaysia Melaka, Hang Tuah  
Jaya, 76100 Durian Tunggal, Melaka, Malaysia.

<sup>4,5</sup>Faculty of Computer and Mathematical Sciences, Universiti Teknologi Mara, Cawangan Melaka, Kampus  
Jasin, Jalan Lembah Kesang 1/1-2, Kampung Seri Mendapat, 77300 Merlimau, Melaka, Malaysia.

\*Corresponding Author's

DOI: <https://doi.org/10.47772/IJRISS.2026.100801118>

Received: 09 September 2026; Accepted: 14 September 2026; Published: 24 September 2026

## ABSTRACT

Remote Access Trojans (RATs) enable an attacker to establish unauthorized remote control over a compromised host and can maintain command-and-control communication that resembles ordinary network traffic. This study developed and functionally tested a network-monitoring system for observing simulated RAT communication in a controlled virtual environment. The system combined Suricata for rule-based intrusion detection, Filebeat and Elasticsearch for log collection and indexing, Kibana for alert visualisation, and Wireshark for packet-level verification. Kali Linux acted as the attacker, while Ubuntu served as the victim and monitoring host. A Metasploit msfvenom payload was delivered through an HTTP server, executed on the victim, and used to establish a reverse TCP session. Custom Suricata rules were configured for payload delivery, suspicious outbound connections, RAT command-and-control activity, heartbeat traffic, and related test events. Nine predefined functional test cases examined traffic capture, alert generation, JSON logging, event forwarding, dashboard display, and packet verification. All nine test cases passed under the tested configuration. This result represents a 100% functional test pass rate within the controlled environment; it does not represent detection accuracy, general classification performance, or operational effectiveness against other RAT families. The study demonstrates the functional feasibility of integrating open-source monitoring components for staged RAT traffic observation and provides a reproducible foundation for broader evaluation. Future work should incorporate multiple real-world RAT families, representative benign background traffic, encrypted command-and-control communications, and independently labelled observations to calculate precision, recall, F1-score, and false-positive rate. Resource consumption, processing throughput, and packet-drop rate should also be evaluated before the system is considered suitable for enterprise deployment.

**Keywords:** Remote Access Trojan (RAT), Suricata, Wireshark, Elastic Stack, Intrusion Detection System, Functionality Testing

## INTRODUCTION

Remote Access Trojans (RATs) are malicious programs that allow attackers to remotely access and control compromised systems. Once executed, a RAT may establish a reverse connection to an attacker-controlled system and support continued command-and-control communication [1]. The network component is therefore important because the malware must exchange traffic with the attacker or command server to receive instructions and return information. Although endpoint security tools can identify known malware, polymorphic, modified, or obfuscated malware may evade conventional signature-based detection [2], [3]. Network monitoring provides an additional source of evidence by revealing connection direction, protocols, destination ports, communication frequency, and packet-level behavior.

Suricata is an open-source network intrusion detection and prevention engine that uses configurable rules to inspect packets, streams, and application-layer transactions [4]. Its Extensible Event Format, known as EVE JSON, records alerts, anomalies, metadata, file information, and protocol-specific events in a structured format [5]. These records can be forwarded to Elasticsearch and presented through Kibana, allowing analysts to examine alert signatures, network endpoints, ports, categories, and event frequency [6]. Wireshark complements rule-based monitoring by capturing packets and allowing their protocols, fields, and communication streams to be examined directly [7]. The two tools therefore serve different but complementary functions: Suricata automatically produces structured alerts when configured rule conditions are satisfied, whereas Wireshark enables analysts to verify the underlying network communication.

Intrusion detection studies commonly evaluate system performance using measures such as accuracy, precision, recall, F1-score, and false-positive rate. These measures require independently labelled malicious and benign observations to determine the numbers of true positives, true negatives, false positives, and false negatives [8]. The experiment conducted in this study instead examined whether an implemented monitoring pipeline produced its expected outputs during a controlled Metasploit-based RAT simulation. Therefore, the study does not propose a universal RAT detection method or claim detection accuracy. Its contribution is limited to the implementation and functional evaluation of an integrated Suricata, Elastic Stack, and Wireshark monitoring workflow across selected RAT communication stages. A comprehensive operational evaluation involving multiple RAT families, benign background traffic, encrypted communications, and resource-performance measurements is reserved for future work.

The objectives of this study were to establish a controlled environment for monitoring simulated RAT communication, configure Suricata rules and Elastic Stack visualizations for the selected communication stages, and verify whether the system could detect, record, display, and corroborate the predefined events. The remainder of this paper is organized as follows. Section 2 reviews the related work, Section 3 describes the experimental methodology, Section 4 presents and analyzes the functionality testing results, Section 5 discusses the findings and their limitations, and Section 6 concludes the study and identifies directions for future work.

## **Related Work**

This section reviews previous studies on RAT communication, behavioral detection, and network monitoring tools relevant to the proposed system.

## **Remote Access Trojan Communication**

RAT activity can be examined through several stages, including payload delivery, execution, establishment of a reverse connection, command-and-control communication, and post-exploitation activities. The cyber kill chain provides a structured perspective for associating observable network events with corresponding attacker actions [9]. Phased detection approaches extend this concept by combining evidence collected from multiple stages of malicious communication [10]. Behavioral studies have also examined connection patterns and network-flow characteristics to distinguish RAT-related communication from ordinary traffic [11]. More recent work has addressed encrypted RAT communication by modeling malicious behavior without relying solely on readable packet contents [12]. Collectively, these studies suggest that RAT detection may become more comprehensive when multiple behavioral indicators and communication stages are considered. However, such approaches generally require representative datasets, repeated experiments, and appropriate validation procedures, which are beyond the scope of a small functional prototype.

## **Network Monitoring and Intrusion Detection Tools**

Wireshark is widely used for network analysis and packet inspection because it enables analysts to examine packet headers, protocol fields, TCP streams, and communication between network endpoints [7]. It is valuable for confirming observed network activity, but its analyst-driven inspection does not replace automated intrusion detection. Suricata can automatically inspect live or recorded network traffic and apply configured rules to generate alerts [13]. It can also record alerts and protocol-related events in the structured EVE JSON format

[14]. Previous comparisons involving Suricata, Snort, and Zeek indicate that their performance and alerting behavior are influenced by rule sets, traffic characteristics, hardware resources, and system configuration [15], [16].

Table 2.1: Comparison of relevant approaches

| Approach                                             | Primary role                              | Strength                                                                   | Limitation                                                |
|------------------------------------------------------|-------------------------------------------|----------------------------------------------------------------------------|-----------------------------------------------------------|
| Wireshark packet analysis [7]                        | Analyst-driven packet inspection          | Provides etailed packet and stream evidence                                | Requires analyst interpretation                           |
| Suricata signature-based detection [13], [14]        | Automated rule matching                   | Produces real-time structured alerts                                       | Detection coverage depends on the rules and configuration |
| Behavioural or phased RAT detection [10], [11], [12] | Multi-stage or multi-feature detection    | May identify a broader range of behavioral patterns                        | Requires representative data and systematic validation    |
| Proposed system                                      | Integrated functional monitoring pipeline | Links automated alerts, event visualization, and packet-level verification | Evaluated using a controlled single-payload simulation    |

As summarized in Table 2.1, existing approaches provide either detailed packet-level evidence, automated rule-based alerts, or broader behavioral analysis. However, fewer studies emphasize the functional integration and traceability of these capabilities within a single monitoring workflow. This study addresses that practical gap by connecting a repeatable RAT simulation with custom Suricata rules, EVE JSON event recording, Elastic Stack-based visualization, and Wireshark packet verification. The evaluation determines whether each component and event path produces its predefined output rather than whether the configuration generalizes to previously unseen malware or operational network conditions. Beyond functional validation, a comprehensive intrusion detection evaluation requires both detection-performance and operational-performance measurements.

Detection metrics such as precision, recall, F1-score, and false-positive rate require independently labelled malicious and benign traffic [8]. Testing with multiple RAT families is also necessary because different RATs may use different communication protocols, ports, traffic patterns, and encryption mechanisms [10], [12]. From an operational perspective, CPU utilization, memory consumption, processing throughput, and packet-drop rate are important for determining whether an intrusion detection pipeline can process high-volume traffic without losing network visibility [15], [16]. These considerations establish the requirements for broader validation beyond the functional scope of the present study.

## METHODOLOGY

This study adopted an experimental approach to monitor Remote Access Trojan (RAT) communication in a controlled virtual environment. The methodology consisted of four main phases: experimental setup, RAT communication simulation, network monitoring and verification, and functionality testing. Figure 3.1 presents an overview of these phases, while the activities conducted in each phase are explained in the following subsections.



Figure 3.1: Overall experimental methodology

## Experimental Setup

The experiment was conducted using VMware Workstation Pro within an isolated host-only network. Kali Linux, assigned the IP address 192.168.101.10, acted as the attacker. Ubuntu 24.04 LTS, assigned the IP address 192.168.101.20, acted as both the victim and monitoring host. Metasploit Framework and msfvenom were installed on Kali Linux to generate the test payload and establish a reverse TCP connection. Suricata, Wireshark, Filebeat, Elasticsearch, and Kibana were installed on the Ubuntu machine. Figure 3.2 illustrates the testbed architecture and the relationships among its main components.

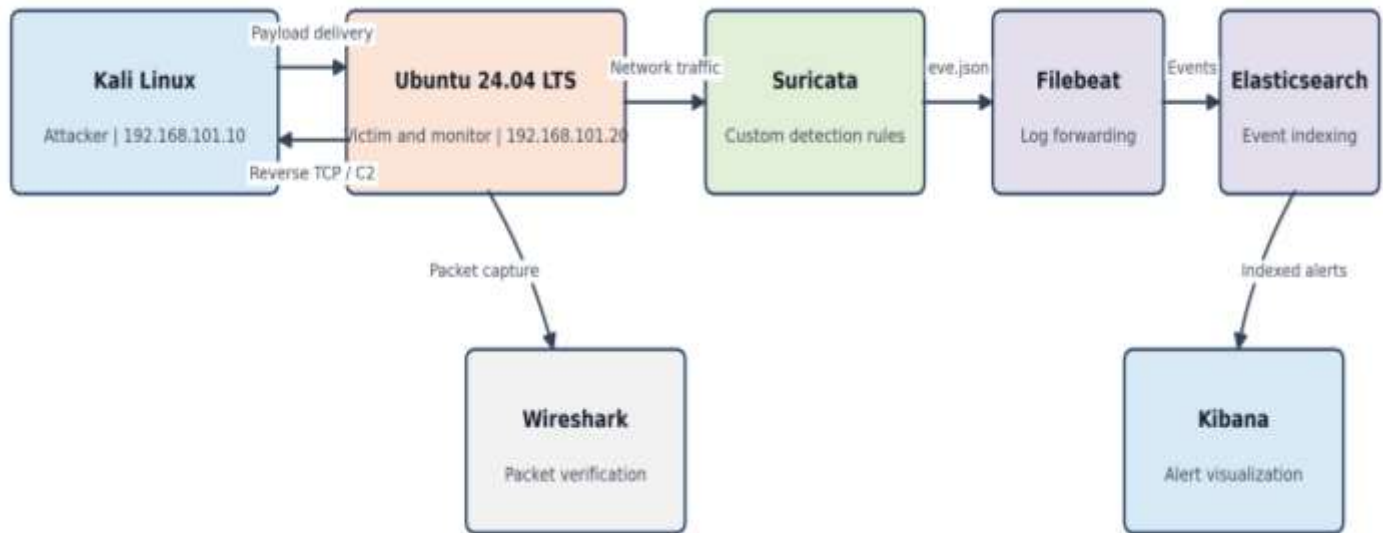


Figure 3.2: Architecture of the RAT communication monitoring system.

As shown in Figure 3.2, the payload was delivered from the Kali Linux attacker to the Ubuntu victim through the isolated network. Once executed, the payload established a reverse TCP connection to the attacker. Suricata inspected the resulting network traffic and recorded the generated alerts in fast.log and eve.json. Filebeat forwarded the EVE JSON events to Elasticsearch for indexing, and Kibana displayed the indexed alerts. At the same time, Wireshark captured the network packets directly from the Ubuntu interface to provide packet-level verification of the Suricata alerts. The configuration and function of each testbed component are summarized in Table 3.1.

Table 3.1: Experimental testbed configuration

| Component           | Configuration                    | Function                                                                    |
|---------------------|----------------------------------|-----------------------------------------------------------------------------|
| Attacker            | Kali Linux, 192.168.101.10       | Generated and delivered the payload and received the reverse TCP connection |
| Victim and monitor  | Ubuntu 24.04 LTS, 192.168.101.20 | Executed the payload and hosted the monitoring tools                        |
| Intrusion detection | Suricata                         | Inspected traffic and generated alerts                                      |
| Packet analysis     | Wireshark                        | Captured and verified network packets                                       |
| Log forwarding      | Filebeat                         | Forwarded EVE JSON events                                                   |
| Event storage       | Elasticsearch                    | Stored and indexed the events                                               |
| Visualisation       | Kibana                           | Displayed the Suricata alerts                                               |

As summarized in Table 3.1, the testbed combined attack simulation, automated network monitoring, alert visualization, and packet-level verification within a single controlled environment. The separation between the attacker and victim machines allowed RAT-related traffic to be generated safely, while the monitoring components provided complementary evidence for the subsequent functionality testing.

## RAT Communication Simulation

The simulation began with the generation of a Linux payload using msfvenom. The payload was hosted on the Kali Linux attacker machine through a Python HTTP server and subsequently downloaded by the Ubuntu victim, representing the delivery stage. After the download was completed, the payload was granted executable permission and executed on the victim machine. A Metasploit handler was configured on the attacker machine to listen for the incoming connection. When the payload was executed on the victim machine, it initiated a reverse TCP connection to the attacker. A Meterpreter session was then established and used to perform limited post-exploitation activities, including system reconnaissance and file-system access. These activities generated RAT-like network communications that could be monitored by Suricata and Wireshark. The three stages included in the simulation are summarized in Table 3.2.

Table 3.2: RAT communication simulation stages

| Stage             | Simulated activity                                                      | Expected network evidence              |
|-------------------|-------------------------------------------------------------------------|----------------------------------------|
| Delivery          | Victim downloaded the payload from the attacker's HTTP server           | HTTP request and file-transfer traffic |
| Execution         | Payload was executed on the Ubuntu victim                               | Outbound reverse TCP connection        |
| Continued control | Meterpreter session was maintained while limited commands were executed | Continued command-and-control traffic  |

As shown in Table 3.2, each stage was associated with observable network evidence. HTTP traffic represented the delivery of the payload, while the outbound reverse TCP connection indicated its execution. Continued traffic between the victim and attacker provided evidence of an active remote-control session. The final stage was described as continued control rather than persistence because the experiment did not test whether access could be maintained after the victim machine was restarted.

## Network Monitoring and Verification

Suricata monitored the traffic captured through the Ubuntu network interface. Custom rules were configured to identify the activities included in the simulation, such as HTTP payload delivery, suspicious outbound connections, reverse TCP communication, and command-and-control traffic. An additional ICMP rule was used to identify the heartbeat-related traffic generated during the controlled experiment. However, an ICMP alert alone was not treated as conclusive evidence of RAT activity. Suricata recorded concise alert information in fast.log, while detailed events were written to eve.json. Filebeat forwarded the EVE JSON events to Elasticsearch for indexing, and Kibana displayed the indexed events through its dashboards. At the same time, Wireshark captured the raw packets from the Ubuntu network interface. The packet captures were used to verify the source and destination IP addresses, port numbers, protocols, and traffic directions associated with the Suricata alerts. An alert was considered successfully verified when the information recorded by Suricata corresponded with the packets observed in Wireshark.

## Functionality Testing

Functionality testing was conducted to determine whether each component of the monitoring system performed its intended function. Each test case contained a predefined expected output. A test was assigned a Pass status when the required Suricata alert, log entry, dashboard record, or packet-level evidence was successfully produced and observed; otherwise, it was assigned a Fail status.

Table 3.3: Functionality test cases

| ID   | Function tested                  | Expected output                                                     |
|------|----------------------------------|---------------------------------------------------------------------|
| FT01 | Capture payload-delivery traffic | The HTTP request and file-transfer packets are visible in Wireshark |
| FT02 | Detect payload-delivery activity | A corresponding Suricata alert is recorded                          |
| FT03 | Detect reverse TCP activity      | The configured reverse TCP alert is generated                       |

|      |                                                    |                                                                                                      |
|------|----------------------------------------------------|------------------------------------------------------------------------------------------------------|
| FT04 | Detect continued command-and-control communication | The corresponding C2-related alert is generated                                                      |
| FT05 | Record security events                             | Alert information is written to fast.log and eve.json                                                |
| FT06 | Forward and index events                           | The Suricata events are available in Elasticsearch                                                   |
| FT07 | Visualize security events                          | The indexed alerts are displayed in Kibana                                                           |
| FT08 | Verify reverse-session traffic                     | The corresponding TCP packets are visible in Wireshark                                               |
| FT09 | Correlate monitoring evidence                      | The simulated attack activity, Suricata alert, and corresponding Wireshark packets can be correlated |

The functional test pass rate was calculated using Equation (1):

$$\text{Functional Test Pass Rate} = \frac{\text{Number of Passed Test Cases}}{\text{Total Number of Executed Test Cases}} \times 100\% \quad (1)$$

The functional test pass rate represents the proportion of predefined system functions that operated successfully under the controlled test conditions. It should not be interpreted as detection accuracy or as evidence that the system can detect all RAT communications. Consequently, a confusion matrix and detection-performance measures such as precision, recall, F1-score, false-positive rate, and false-negative rate were not calculated. Resource-performance measurements were also outside the scope of the functionality testing. The experiment did not measure CPU utilization, memory consumption, packet-processing throughput, event-processing latency, or packet-drop rate under high traffic loads. Similarly, the test configuration did not include multiple real-world RAT families, representative benign background traffic, or encrypted command-and-control channels. These conditions are therefore treated as requirements for subsequent detection-performance and operational-performance evaluation.

## RESULTS AND ANALYSIS

This section presents the results obtained from the functionality testing of the RAT communication monitoring system. The findings are first reported according to the predefined functionality test cases and are then examined across the payload-delivery, execution, and continued-control stages. Evidence collected from Suricata logs, Kibana dashboards, and Wireshark packet captures was used to determine whether the monitoring components operated as expected within the controlled test environment.

### Functional Test Results

The monitoring pipeline produced the expected evidence across the payload-delivery, execution, and continued-control stages. During payload delivery, Suricata recorded an alert associated with the HTTP file transfer, while Wireshark captured the corresponding HTTP GET request and transfer packets. During execution, the configured reverse TCP rule generated an alert when the victim initiated a connection to the attacker. Wireshark captured the corresponding TCP communication. During continued control, Suricata recorded alerts associated with the active command-and-control session. The EVE JSON output contained structured event information, including timestamps, IP addresses, ports, protocols, alert signatures, and alert categories. Filebeat successfully forwarded these events to Elasticsearch for indexing. Kibana subsequently displayed the indexed events through alert-trend, signature, category, and detailed-event views. The Kibana records confirmed that the event-forwarding and visualization components operated as intended. The packet-level observations in Wireshark were also consistent with the endpoints, protocols, ports, and traffic directions recorded by Suricata.

Table 4.1: Functionality testing results

| ID   | Observed output                                                  | Status |
|------|------------------------------------------------------------------|--------|
| FT01 | Wireshark captured the HTTP request and file transfer packets.   | Pass   |
| FT02 | Suricata recorded the corresponding HTTP payload-delivery alert. | Pass   |

|      |                                                                                                              |      |
|------|--------------------------------------------------------------------------------------------------------------|------|
| FT03 | Suricata generated the configured reverse TCP alert                                                          | Pass |
| FT04 | Suricata recorded an alert associated with the continued command-and-control communication                   | Pass |
| FT05 | Alert details were successfully written to fast.log and eve.json                                             | Pass |
| FT06 | The EVE JSON events were forwarded to and indexed in Elasticsearch                                           | Pass |
| FT07 | Kibana displayed the indexed alerts, signatures, categories, trends, and event details.                      | Pass |
| FT08 | Wireshark captured the TCP packets associated with the reverse session.                                      | Pass |
| FT09 | The simulated activities, Suricata alerts, and corresponding Wireshark packets were successfully correlated. | Pass |

As presented in Table 4.1, all nine predefined functionality test cases passed. Based on Equation (1), the functional test pass rate was calculated as follows:

$$\text{Functional test pass rate} = (9 \text{ passed tests} / 9 \text{ executed tests}) \times 100\% = 100\%$$

The 100% functional test pass rate indicates that all predefined system functions operated as expected within the controlled test environment. In particular, the system successfully captured the simulated traffic, generated the required alerts, recorded and forwarded the events, displayed them through Kibana, and provided the corresponding packet-level evidence. However, this outcome does not represent 100% RAT detection accuracy. The experiment did not use an independent collection of malicious and legitimate network traffic. Therefore, the results did not support the construction of a confusion matrix or the calculation of precision, recall, F1-score, false-positive rate, or false-negative rate.

## Results by Attack Stage

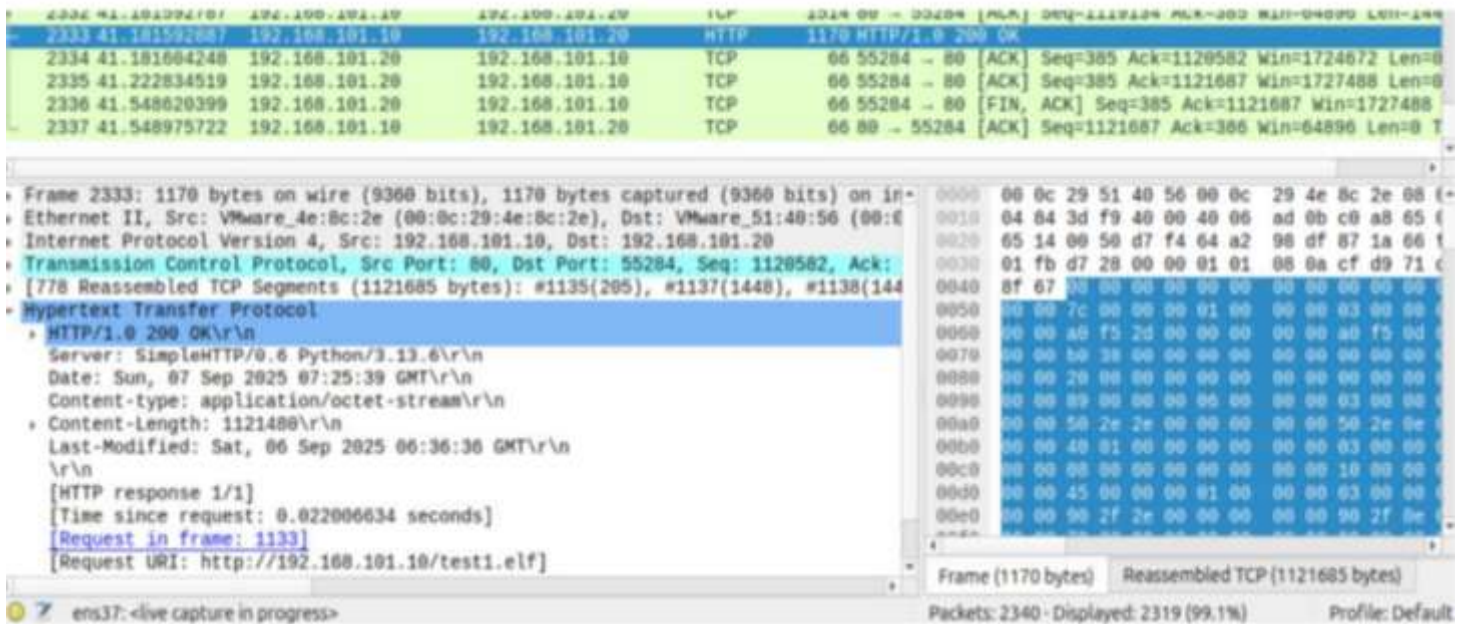
Representative evidence from the functionality testing is presented in Figures 4.1-4.3. The Suricata alert records are shown together with the corresponding Wireshark packet captures to demonstrate that the automated alerts were supported by packet-level evidence. Only selected outputs are presented to avoid repeating similar screenshots for every test case.

```

"proto": "TCP",
"src_port": "80",
"dest_ip": "192.168.101.20",
"dest_port": 45174,
"proto": "TCP",
"pkt_src": "wire/pcap",
"tx_id": 0,
"alert": {
  "action": "allowed",
  "gid": 1,
  "signature_id": 2000000,
  "rev": 5,
  "signature": "Python SimpleHTTP file serving from attacker",
  "category": "Misc activity",
  "severity": 3
},
"http": {
  "hostname": "192.168.101.10",
  "url": "/test1.elf",
  "http_user_agent": "Wget/1.21.4",
  "http_content_type": "application/octet-stream",
  "http_method": "GET",
  "protocol": "HTTP/1.1",
  "status": 200,
  "length": 0
},
"app_proto": "http",
"direction": "to_client",
"flow": {
  "pkts_to_server": 4,
  "pkts_to_client": 4,
  "bytes_to_server": 410,
  "bytes_to_client": 1925,
  "start": "2025-09-07T09:18:47.347583+0800",
  "src_ip": "192.168.101.20",
  "dest_ip": "192.168.101.10",
  "src_port": 45174,
  "dest_port": 80
}

```

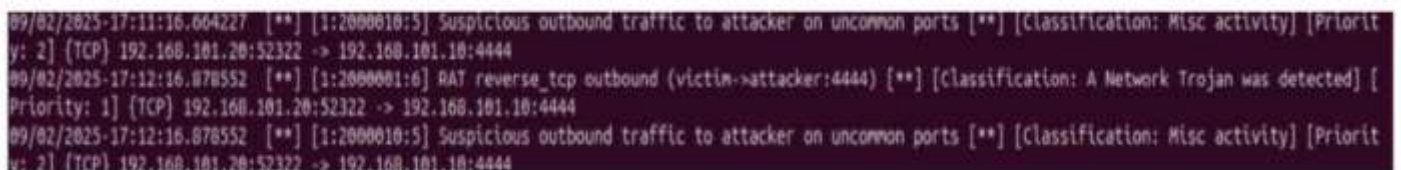
(a) Suricata EVE JSON alert



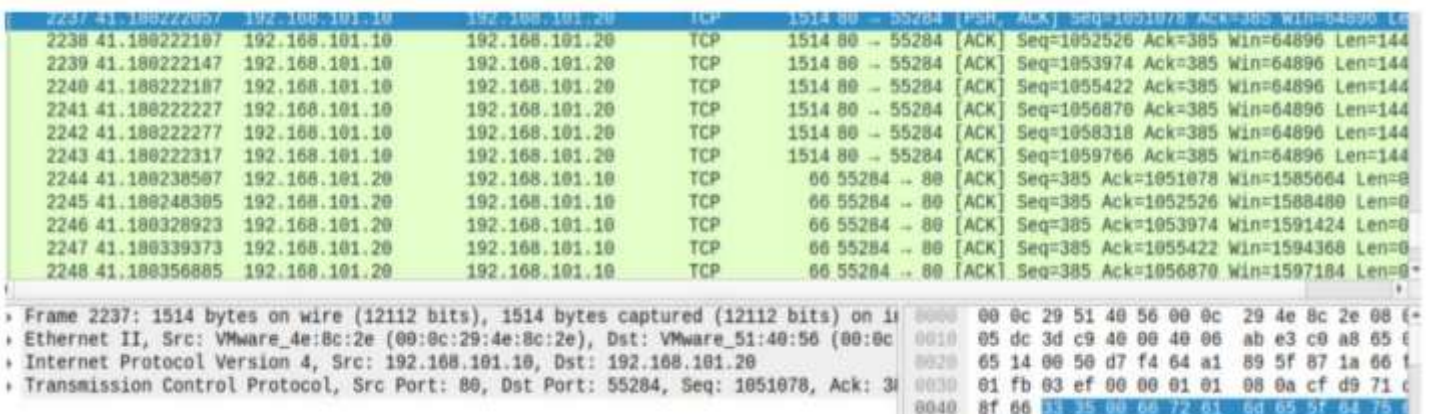
(b) Corresponding Wireshark HTTP packets

Figure 4.1. Suricata alert and Wireshark packet evidence for the payload-delivery stage:  
(a) EVE JSON alert and (b) corresponding HTTP file-transfer packets.

Figure 4.1 presents the monitoring evidence obtained during the payload-delivery stage. The Suricata EVE JSON record identified HTTP communication from the attacker at 192.168.101.10 to the victim at 192.168.101.20 and recorded the configured payload-delivery alert. The corresponding Wireshark capture confirmed the HTTP GET request and file-transfer packets between the same endpoints. The agreement between these two sources shows that the monitoring system successfully detected the delivery activity and that the alert could be verified using packet-level evidence. Following payload delivery, the execution of the payload caused the victim to initiate a reverse TCP connection to the attacker. The monitoring evidence obtained during this execution stage is presented in Figure 4.2.



(a) Suricata reverse TCP alert



(b) Corresponding Wireshark TCP communication

Figure 4.2: Monitoring evidence for the execution stage: (a) Suricata reverse TCP alert and (b) corresponding TCP communication captured in Wireshark

Figure 4.2 presents the monitoring evidence obtained during the execution stage. Suricata generated the configured reverse TCP alert when the Ubuntu victim initiated an outbound connection to the Kali Linux attacker. The corresponding Wireshark capture confirmed continuous TCP communication between the same IP addresses and port numbers recorded in the alert. This agreement demonstrates that the system successfully identified the reverse connection and that the Suricata alert was supported by packet-level evidence. After the reverse TCP connection was established, the Meterpreter session remained active while limited commands were executed. Figure 4.3 presents the monitoring evidence collected during this continued-control stage.

```

"signature": "RAT reverse_tcp outbound (victim->attacker:4444)",
"category": "A Network Trojan was detected",
"severity": 1
},
"app_proto": "failed",
"direction": "to_server",
"flow": {
  "pkts_toserver": 507,
  "pkts_toclient": 2749,
  "bytes_toserver": 82939,
  "bytes_toclient": 3311031,
  "start": "2025-09-02T12:31:05.440915+0800",
  "src_ip": "192.168.101.20",
  "dest_ip": "192.168.101.10",
  "src_port": 52322,
  "dest_port": 4444
},
"timestamp": "2025-09-07T06:06:43.995677+0800",
"flow_id": 486344744967781,
"in_iface": "ens37",
"event_type": "alert",
"src_ip": "192.168.101.20",
"src_port": 52322,
"dest_ip": "192.168.101.10",
"dest_port": 4444,
"proto": "TCP",
"pkt_src": "stream (flow timeout)",
>alert": {
  "action": "allowed",
  "gid": 1,
  "signature_id": 2000010,
  "rev": 5,
  "signature": "Suspicious outbound traffic to attacker on uncommon ports",
  "category": "Misc activity",

```

(a) Suricata EVE JSON alert

| No.  | Time         | Source         | Destination    | Protocol | Length | Info                                                    |
|------|--------------|----------------|----------------|----------|--------|---------------------------------------------------------|
| 2975 | 1243.8261427 | 192.168.101.10 | 192.168.101.20 | TCP      | 194    | 4444 → 52322 [PSH, ACK] Seq=2561 Ack=2881 Win=486 Len=  |
| 2976 | 1243.8262634 | 192.168.101.20 | 192.168.101.10 | TCP      | 210    | 52322 → 4444 [PSH, ACK] Seq=2881 Ack=2689 Win=3966 Len= |
| 2977 | 1243.8267129 | 192.168.101.10 | 192.168.101.20 | TCP      | 66     | 4444 → 52322 [ACK] Seq=2689 Ack=3025 Win=485 Len=0 TSV  |
| 2985 | 1304.0306000 | 192.168.101.10 | 192.168.101.20 | TCP      | 194    | 4444 → 52322 [PSH, ACK] Seq=2689 Ack=3025 Win=486 Len=  |
| 2986 | 1304.0308281 | 192.168.101.20 | 192.168.101.10 | TCP      | 210    | 52322 → 4444 [PSH, ACK] Seq=3025 Ack=2817 Win=3966 Len= |
| 2987 | 1304.0312351 | 192.168.101.10 | 192.168.101.20 | TCP      | 66     | 4444 → 52322 [ACK] Seq=2817 Ack=3169 Win=485 Len=0 TSV  |
| 3005 | 1364.2372875 | 192.168.101.10 | 192.168.101.20 | TCP      | 194    | 4444 → 52322 [PSH, ACK] Seq=2817 Ack=3169 Win=486 Len=  |
| 3006 | 1364.2375040 | 192.168.101.20 | 192.168.101.10 | TCP      | 210    | 52322 → 4444 [PSH, ACK] Seq=3169 Ack=2945 Win=3966 Len= |
| 3007 | 1364.2378604 | 192.168.101.10 | 192.168.101.20 | TCP      | 66     | 4444 → 52322 [ACK] Seq=2945 Ack=3313 Win=485 Len=0 TSV  |
| 3028 | 1424.4468756 | 192.168.101.10 | 192.168.101.20 | TCP      | 194    | 4444 → 52322 [PSH, ACK] Seq=2945 Ack=3313 Win=486 Len=  |
| 3029 | 1424.4471018 | 192.168.101.20 | 192.168.101.10 | TCP      | 210    | 52322 → 4444 [PSH, ACK] Seq=3313 Ack=3073 Win=3966 Len= |
| 3030 | 1424.4474167 | 192.168.101.10 | 192.168.101.20 | TCP      | 66     | 4444 → 52322 [ACK] Seq=3073 Ack=3457 Win=485 Len=0 TSV  |
| 3048 | 1484.6567395 | 192.168.101.10 | 192.168.101.20 | TCP      | 194    | 4444 → 52322 [PSH, ACK] Seq=3073 Ack=3457 Win=486 Len=  |
| 3049 | 1484.6569156 | 192.168.101.20 | 192.168.101.10 | TCP      | 210    | 52322 → 4444 [PSH, ACK] Seq=3457 Ack=3201 Win=3966 Len= |
| 3050 | 1484.6573964 | 192.168.101.10 | 192.168.101.20 | TCP      | 66     | 4444 → 52322 [ACK] Seq=3201 Ack=3601 Win=485 Len=0 TSV  |
| 3056 | 1544.8633687 | 192.168.101.10 | 192.168.101.20 | TCP      | 194    | 4444 → 52322 [PSH, ACK] Seq=3201 Ack=3601 Win=486 Len=  |

\* Frame 30: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface e1  
 \* Ethernet II, Src: VMware\_4e:8c:2e (00:0c:29:4e:8c:2e), Dst: VMware\_51:40:56 (00:0c:  
 \* Internet Protocol Version 4, Src: 192.168.101.10, Dst: 192.168.101.20  
 \* Transmission Control Protocol, Src Port: 4444, Dst Port: 52322, Seq: 129, Ack: 145,  
 0000 00 0c 29 51 40 56 00 0c 29 4e 8c 2e 08 06  
 0010 00 34 55 d0 40 00 40 06 99 84 c0 a8 65 0a  
 0020 65 14 11 5c cc 62 b4 e6 ff e5 5e 9a c9 22  
 0030 01 e5 58 37 60 60 01 01 08 0a cf d9 6b d0  
 0040 89 58

## (b) Corresponding Wireshark C2 packets

Figure 4.3: Monitoring evidence for the continued-control stage: (a) Suricata EVE JSON alert and (b) corresponding command-and-control traffic captured in Wireshark

Figure 4.3 presents the monitoring evidence obtained during the continued-control stage. The Suricata EVE JSON record showed command-and-control-related communication between the Ubuntu victim and the Kali Linux attacker. The corresponding Wireshark capture confirmed continued TCP packet exchanges between the same endpoints and ports. This agreement indicates that the monitoring system successfully recorded the ongoing reverse session and provided packet-level evidence to support the generated alert. However, this result represents continued control rather than persistence because the experiment did not examine whether the connection could be restored after restarting the victim machine. Based on the evidence presented in Figures 4.1-4.3, Table 4.2 summarizes the simulated activities and the corresponding evidence recorded by Suricata and Wireshark across the three experimental stages.

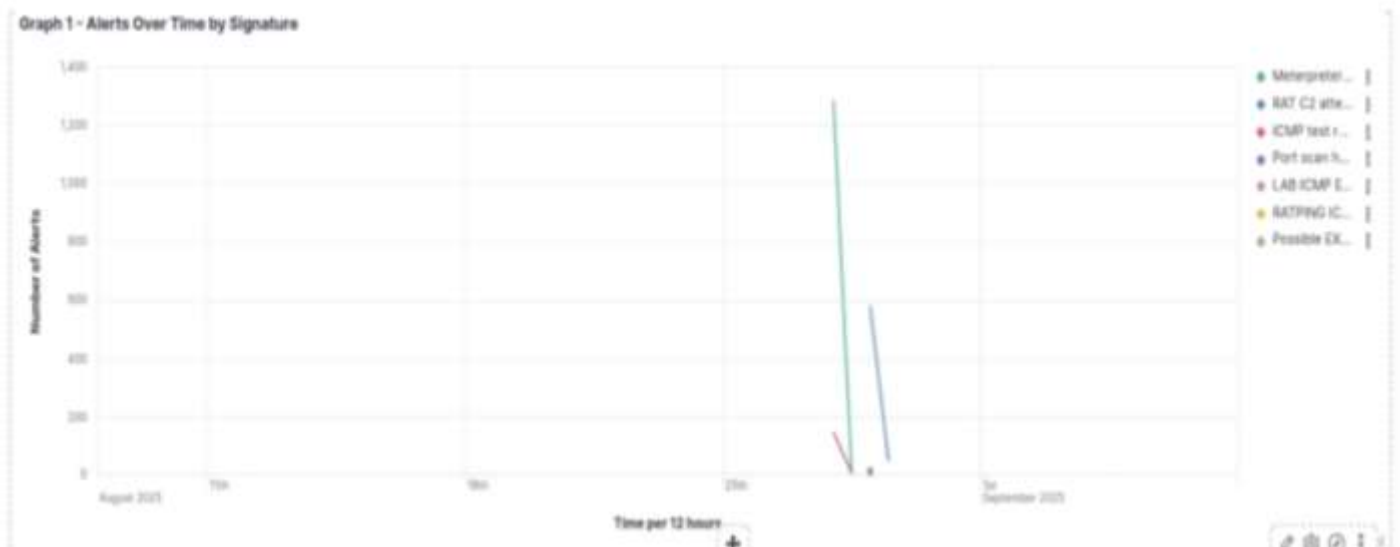
Table 4.2: Stage-based summary of monitoring evidence

| Stage             | Simulated event                                                              | Suricata evidence                                         | Wireshark evidence                                          |
|-------------------|------------------------------------------------------------------------------|-----------------------------------------------------------|-------------------------------------------------------------|
| Payload delivery  | The victim downloaded the payload from the attacker's HTTP server            | HTTP payload-delivery alert recorded in the Suricata logs | HTTP GET request and file-transfer packets                  |
| Execution         | The victim executed the payload and initiated a reverse TCP connection       | Configured reverse TCP alert generated                    | Outbound TCP packets and the corresponding TCP stream       |
| Continued control | The Meterpreter session remained active while limited commands were executed | Command-and-control-related alerts recorded               | Continued TCP communication between the victim and attacker |

During payload delivery, the HTTP transfer observed in Wireshark corresponded with the delivery alert generated by Suricata. During execution, the reverse TCP alert was supported by the outbound TCP connection captured in Wireshark. During continued control, the recorded alert corresponded with continued network communication between the victim and attacker.

## Event Visualization and Overall Analysis

Figure 4.4 presents the Kibana visualization used to examine the indexed Suricata events. The alerts-over-time view showed when the configured signatures were triggered, while the detailed record view presented the associated timestamps, endpoints, ports, categories, and signatures.



a) Alerts over time by signature

Table Full Report

2,104 documents

| @timestamp                  | src_ip         | dest_ip        | dest_port | proto | alert.signature                               | alert.category                | alert.severity |
|-----------------------------|----------------|----------------|-----------|-------|-----------------------------------------------|-------------------------------|----------------|
| Aug 29, 2025 @ 12:22:06.796 | 192.168.101.30 | 192.168.101.10 | 4,444     | TCP   | RAT C2 attempt victim->attacker (reverse_tcp) | A Network Trojan was detected | 1              |
| Aug 29, 2025 @ 12:21:42.768 | 192.168.101.30 | 192.168.101.10 | 4,444     | TCP   | RAT C2 attempt victim->attacker (reverse_tcp) | A Network Trojan was detected | 1              |
| Aug 29, 2025 @ 12:21:07.745 | 192.168.101.30 | 192.168.101.10 | 4,444     | TCP   | RAT C2 attempt victim->attacker (reverse_tcp) | A Network Trojan was detected | 1              |
| Aug 29, 2025 @ 12:20:41.734 | 192.168.101.30 | 192.168.101.10 | 4,444     | TCP   | RAT C2 attempt victim->attacker (reverse_tcp) | A Network Trojan was detected | 1              |
| Aug 29, 2025 @ 12:20:06.796 | 192.168.101.30 | 192.168.101.10 | 4,444     | TCP   | RAT C2 attempt                                | A Network Trojan was          | 1              |

(b) Detailed alert records

Figure 4.4: Visualization of indexed Suricata events in Kibana: (a) alerts over time by signature and (b) detailed alert records.

The results demonstrate end-to-end traceability within the monitoring pipeline. A simulated activity could be observed as raw network traffic, identified through a Suricata rule, recorded as a structured EVE JSON event, forwarded to Elasticsearch, and displayed in Kibana. Wireshark provided additional packet-level evidence for corresponding communication. These findings confirm the functional integration of the monitoring components under the tested configuration. However, the controlled environment contained no representative benign background traffic, used only one Metasploit-generated payload, and relied on a limited set of configuration-specific rules. The experiment also did not include encrypted command-and-control communication or resource-performance measurements under different traffic loads. Consequently, the results do not establish detection

accuracy, false-positive performance, scalability, resilience to encrypted traffic, or generalizability to other RAT families and operational networks.

## DISCUSSION

This study evaluated the functionality of a RAT communication monitoring system using Suricata, Wireshark, Filebeat, Elasticsearch, and Kibana in a controlled virtual environment. All nine predefined functionality test cases passed, producing a functional test pass rate of 100%. This result confirms that the components operated as intended and that the monitoring pipeline successfully captured, detected, recorded, forwarded, indexed, visualized, and verified the simulated activities. During the payload-delivery stage, Suricata generated an alert associated with the HTTP file transfer, while Wireshark captured the corresponding HTTP GET request and transfer packets. The agreement between the evidence recorded by both tools demonstrates that the configured payload-delivery activity was successfully identified and verified at the packet level.

During the execution stage, the Ubuntu victim initiated a reverse TCP connection to the Kali Linux attacker. Suricata generated the configured reverse TCP alert, and Wireshark captured the corresponding communication between the same endpoints and ports. During the continued-control stage, Suricata recorded command-and-control-related events, while Wireshark showed continued TCP packet exchanges between the victim and attacker. These findings indicate that the system maintained visibility of the active session beyond the initial connection. However, the activity is described as continued control rather than persistence because the experiment did not examine whether access could be restored after restarting the victim machine. Similarly, ICMP-related alerts were treated only as supporting evidence because legitimate network-management activities may also produce ICMP traffic.

The results also confirmed the successful integration of monitoring and visualization components. Suricata recorded alert information in `fast.log` and structured events in `eve.json`. Filebeat forwarded the EVE JSON events to Elasticsearch for indexing, and Kibana presented the indexed information through alert-trend, signature, category, and detailed-event views. The main contribution of this study is the end-to-end traceability achieved across the monitoring pipeline. A simulated activity could be observed as raw traffic in Wireshark, identified through a Suricata rule, recorded as a structured event, forwarded to Elasticsearch, and displayed in Kibana. This combination provides complementary evidence because Suricata supports automated identification of configured suspicious activities, while Wireshark provides packet-level information for validating the corresponding network communication.

The stage-based monitoring approach is consistent with PRATD [10], which associates RAT detection with evidence collected across different communication phases. The use of communication behaviour rather than relying entirely on payload content also reflects the principle applied in MBTree [12]. However, unlike these studies, the present work did not develop or evaluate a new classification method. Its contribution is limited to demonstrating the functional integration and end-to-end traceability of Suricata alerts, Elastic Stack records, and Wireshark packet evidence within the tested configuration.

Despite the 100% functional test pass rate, the result should not be interpreted as 100% RAT detection accuracy. The experiment used one Metasploit-generated payload, fixed network configurations, and a limited set of custom Suricata rules. Representative benign background traffic was not included; therefore, the experiment could not determine whether legitimate network activities would incorrectly trigger the configured rules. The use of a single simulated payload also does not establish whether the rules can identify RAT families with different communication behaviours, such as AsyncRAT, njRAT, or QuasarRAT. Because independently labelled malicious and benign observations were unavailable, a valid confusion matrix could not be constructed, and precision, recall, F1-score, false-positive rate, and false-negative rate could not be calculated.

The experiment was also limited to unencrypted HTTP and reverse TCP communication. Encrypted HTTPS or TLS-based command-and-control channels may conceal payload contents and reduce the effectiveness of content-dependent detection rules. Future evaluation should therefore examine detection based on observable connection characteristics, including endpoints, ports, traffic direction, connection duration, packet sizes,

communication frequency, and TLS metadata. Operational performance should also be evaluated by measuring CPU utilization, memory consumption, processing latency, throughput, and packet-drop rate under different traffic loads. These additional experiments are necessary before the monitoring pipeline can be assessed for deployment in enterprise network environments.

## CONCLUSION

In conclusion, the proposed RAT communication monitoring system successfully integrated Suricata, Wireshark, Filebeat, Elasticsearch, and Kibana within an isolated virtual environment. The system was evaluated across three simulated stages: payload delivery, execution, and continued control. Suricata generated alerts for the configured activities, while Wireshark provided the corresponding packet-level evidence. The generated events were successfully recorded, forwarded, indexed, and displayed through Kibana. All nine functionality test cases passed, resulting in a functional test pass rate of 100%. This outcome confirms that the monitoring components were successfully integrated and produced their expected outputs under the controlled test conditions. The study demonstrates end-to-end traceability by connecting the simulated activities with Suricata alerts, structured event records, Kibana visualizations, and corresponding Wireshark packets. Nevertheless, the results demonstrate functional operation rather than detection accuracy or effectiveness in an operational network. Future work should evaluate multiple real-world RAT families together with representative benign background traffic and independently labelled flow- or session-level observations. Such an evaluation would support the construction of a confusion matrix and the calculation of precision, recall, F1-score, false-positive rate, and false-negative rate. Encrypted command-and-control communication over HTTPS or TLS should also be examined to determine whether the monitoring approach remains effective when packet contents are unavailable. Finally, CPU utilization, memory consumption, processing latency, throughput, and packet-drop rate should be measured under progressively higher traffic loads to assess the operational feasibility of the pipeline for enterprise networks.

## ACKNOWLEDGEMENT

The authors would like to thank Universiti Teknikal Malaysia Melaka (UTeM) and Universiti Teknologi MARA (UiTM) Jasin Campus for the support.

## REFERENCES

1. MITRE. (n.d.). Remote access tools: Technique T1219. MITRE ATT&CK. Retrieved September 9, 2025, from <https://attack.mitre.org/techniques/T1219/>
2. Cybersecurity and Infrastructure Security Agency. (n.d.). Polymorphic code: Technique T1027.014. Retrieved September 9, 2025, from <https://www.cisa.gov/eviction-strategies-tool/info-attack/T1027.014>
3. Mell, P., Kent, K., & Nusbaum, J. (2005). Guide to malware incident prevention and handling (NIST Special Publication 800-83). National Institute of Standards and Technology.
4. Siddiqui, A., Rimal, B. P., Reisslein, M., Gc, D., & Wang, Y. (2024). SUTMS: Designing a unified threat management system for home networks. *IEEE Access*, 12, 80930–80949.
5. Soroka, B. M. (2023). Graph-based analysis of network traffic and Suricata logs [Master's thesis, Masaryk University].
6. Elastic. (n.d.). Kibana: Explore and visualize data stored in Elasticsearch. Retrieved September 9, 2025, from <https://www.elastic.co/kibana>
7. Wireshark Foundation. (n.d.). Wireshark user's guide. Wireshark Documentation. Retrieved September 9, 2025.
8. Scikit-learn developers. (n.d.). Precision, recall, and F-measures. Scikit-learn User Guide. Retrieved September 9, 2025.
9. HosseiniNejad, R., HaddadPajouh, H., Dehghantanha, A., & Parizi, R. M. (2019). A cyber kill chain based analysis of remote access trojans. In K.-K. R. Choo and A. Dehghantanha (Eds.), *Handbook of big data and IoT security* (pp. 273–299). Springer.
10. Guo, C., Song, Z., Ping, Y., Shen, G., & Cui, Y. (2020). PRATD: A phased remote access Trojan detection method with double-sided features. *Electronics*, 9(11), Article 1894.

11. Awad, A. A., Sayed, S. G., & Salem, S. A. (2019). Collaborative framework for early detection of RAT-bots attacks. *IEEE Access*, 7, 71780–71790.
12. Dong, C., Lu, Z., Cui, Z., Liu, B., & Chen, K. (2021). MBTree: Detecting encryption RAT communication using malicious behavior tree. *IEEE Access*, 9.
13. Open Information Security Foundation. (n.d.-b). Suricata rules. *Suricata Documentation*. Retrieved September 10, 2025.
14. Open Information Security Foundation. (n.d.-a). EVE JSON output. *Suricata Documentation*. Retrieved September 10, 2025.
15. Waleed, A., Jamali, A. F., & Masood, A. (2022). Which open-source IDS? Snort, Suricata, or Zeek. *Computer Networks*, 213, Article 109116.
16. Shah, S. A. R., & Issac, B. (2018). Performance comparison of intrusion detection systems and application of machine learning to Snort system. *Future Generation Computer Systems*, 80, 157–170.