



SOICT

PROJECT II

Research and Deployment of Suricata as a Network-Based Intrusion Detection System

CHU TRUNG ANH

anh.ct225564@sis.hust.edu.vn

Major: Cybersecurity IT-E15

Supervisor: Assoc. Prof. Nguyen Linh Giang

M.Sc. Nguyen Quoc Khanh

Signature

Department: Computer Engineering

HANOI, 3/2025

© 2025-Chu Trung Anh

All rights reserved. Re-distributed by Hanoi University of Science and Technology under license with the author.

This work is licensed under a Creative Commons
“Attribution-NonCommercial-ShareAlike 3.0 Un-
ported” license.



1. Student's information:

Name: Chu Trung Anh

Email: anh.ct225564@sis.hust.edu.vn

Class: IT-E15 01 K67

Affiliation: School of Information and Communication Technology, Hanoi University of Science and Technology

Duration: 18/03/2025–XX/XX/2025

2. Project title:

Research and Deployment of Suricata as a Network-Based Intrusion Detection System

3. Project statement:

Update Later

4. Declarations/Disclosures:

I, Chu Trung Anh, hereby declare that::

1. This project titled "**Research and Deployment of Suricata as a Network-Based Intrusion Detection System**" represents my original work conducted under the supervision of Assoc. Prof. Nguyen Linh Giang and M.Sc. Nguyen Quoc Khanh at Hanoi University of Science and Technology.
2. The content presented is our own academic endeavor, based on research and understanding of Suricata IDS and its application in network protection.
3. All sources used in this thesis, including publications and articles, have been appropriately cited and acknowledged.
4. Any opinions, findings, conclusions, or recommendations expressed herein are solely those of the authors and do not necessarily reflect the views of Hanoi University of Science and Technology or any other entity.
5. This thesis or any part of it has not been submitted for any other degree or qualification at any university or institution.
6. We affirm that all content in this thesis is original and properly attributed, understanding the severity of plagiarism in academic contexts.
7. We grant Hanoi University of Science and Technology permission to archive and provide access to this thesis for educational and research purposes.

Hanoi, date month year 2025
Authors

Chu Trung Anh

5. Attestation of project advisors:

.....
.....

Hanoi, date month year 2025
Project advisors

Nguyen Linh Giang Nguyen Quoc Khanh

ACKNOWLEDGMENTS

Throughout the process of studying and conducting research for this project, in addition to my own efforts, I have received invaluable guidance from my professors, as well as support and encouragement from my family, friends, and colleagues. With deep respect and gratitude, I would like to extend my heartfelt thanks to **Assoc. Prof. Nguyen Linh Giang** and **M.Sc. Nguyen Quoc Khanh** for their dedicated support, guidance, and encouragement, which greatly facilitated my research and the completion of this project.

I am also deeply grateful to my family, friends, and colleagues for their unwavering support and encouragement during difficult times, enabling me to complete my studies and this project.

Despite my best efforts, due to time constraints and limited research experience, certain shortcomings in this thesis may be unavoidable. I sincerely welcome any comments and suggestions from professors, colleagues, and friends to further improve my knowledge and work.

Sincerely,

Hanoi, date month year 2025
Authors

Chu Trung Anh

ABSTRACT

Update later

Student
(Signature and full name)

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION.....	1
1.1 Motivation	1
1.2 Objectives and scope of the graduation thesis	1
1.3 Tentative solution	1
1.4 Thesis organization.....	1
CHAPTER 2. THEORETICAL FOUNDATION OF IDS.....	2
2.1 Introduction to IDS	2
2.1.1 Definition of IDS	2
2.1.2 Importance of IDS in cybersecurity	2
2.1.3 General architecture of an IDS	3
2.2 Types of ID	4
2.2.1 Network-based IDS (NIDS)	4
2.2.2 Host-based IDS (HIDS)	7
2.2.3 Hybrid IDS.....	10
2.2.4 Other Specialized IDS Types.....	11
2.3 Detection Methods.....	11
2.3.1 Anomaly-based detection.....	12
2.3.2 Other Detection Methods.....	14
2.4 IDS Evasion Techniques.....	14
2.5 IDS Integration with Other Security Tools	16
2.5.1 IDS vs. Firewall.....	18
2.5.2 IDS and SIEM	19
2.5.3 IDS and Honeypots	20
2.6 Some Popular IDS/IPS In The Industry.....	21

CHAPTER 3. RESEARCH ON SURICATA IDS	26
3.1 Introduction to Suricata.....	26
3.1.1 What is Suricata.....	26
3.1.2 Development History.....	26
3.2 Suricata Architecture	27
3.2.1 Packet Capture.....	28
3.2.2 Decoding.....	29
3.2.3 Detection.....	31
3.2.4 Output.....	31
3.3 Suricata Rule Structure	32
3.3.1 Actions	33
3.3.2 Header	34
3.3.3 Options	36
CHAPTER 4. LAB ENVIRONMENT SETUP	46
4.1 Application Building.....	46
4.1.1 Libraries and Tools.....	46
4.1.2 Achievement	46
4.1.3 Illustration of main functions	46
4.2 Testing.....	46
4.3 Deployment	46
CHAPTER 5. CONCLUSION AND FUTURE WORK.....	47
5.1 Conclusion.....	47
5.2 Future Work.....	47
REFERENCE	48
APPENDIX	48

LIST OF FIGURES

Figure 2.1	General Architecture of an IDS	4
Figure 2.2	NIDS placement within a network	5
Figure 2.3	Port Mirroring (SPAN) on Switch	6
Figure 2.4	NIDS (left) vs. HIDS (right)	9
Figure 2.5	Signature and Anomaly based	14
Figure 2.6	IDS (left) vs IPS (right)	17
Figure 2.7	Firewall and IDS in a network	19
Figure 2.8	IDS as a data source of SIEM	19
Figure 2.9	IDS combines with Honeypots within a network	20
Figure 3.1	Suricata logo	26
Figure 3.2	Suricata Timeline	26
Figure 3.3	Suricata Architecture	27
Figure 3.4	Suricata source code files for decoders	30
Figure 3.5	Example of a Suricata rule	33
Figure 3.6	Example of Suricata rule header	34
Figure 3.7	Class types inside config file	39
Figure 3.8	Reference types defined inside Suricata config file	40
Figure 3.9	This rule will only match if the string 'def' appears in a gap between 1 to 4 Bytes counting from the end of the string 'abc'.	43
Figure 3.10	Start from the last mach, letter "d" in this case, count till the end of payload, there are only 7 bytes left.	43

LIST OF TABLES

Table 2.1	Main data sources for NIDS	7
Table 2.2	Main data sources for HIDS	8
Table 2.3	Comparison Between Network-based IDS (NIDS) and Host-based IDS (HIDS) [1]	10
Table 2.4	Comparison of IDS and IPS Characteristics	17
Table 2.5	Summary Table of IDS/IPS Solutions	25
Table 3.1	Packet Capture Mechanisms Across Different Platforms	29
Table 3.2	Summary of decoding layers and extracted information	30
Table 3.3	Behavior of Suricata actions in IDS vs. IPS modes	33
Table 3.4	IP address specification	35
Table 3.5	Description of port matching types and examples	35
Table 4.1	List of libraries and tools used	46

LIST OF ABBREVIATIONS

Abriviation	Full Expression
DL	Deep Learning
IDS	Intrusion Detection System
ML	Machine Learning
NIDS	Network-Based Intrusion Detection System
NIDS	Host-Based Intrusion Detection System
SIEM	Security Information and Event Management

CHAPTER 1. INTRODUCTION

1.1 Motivation

Update Later

1.2 Objectives and scope of the graduation thesis

Update Later

1.3 Tentative solution

Update Later

1.4 Thesis organization

Update Later

CHAPTER 2. THEORETICAL FOUNDATION OF IDS

2.1 Introduction to IDS

2.1.1 Definition of IDS

An Intrusion Detection System (IDS) is a security tool designed to detect unauthorized access, malicious activities, or policy violations within a network or computer system. It works by continuously monitoring network traffic, system logs, and application behavior, then alerting security personnel when it detects suspicious activity.

2.1.2 Importance of IDS in cybersecurity

In today's connected digital world, Intrusion Detection Systems have become essential components of organizational security. Their importance can be understood through several key benefits:

- **Early Warning System**

IDS provides real-time monitoring and detection of suspicious activities, which is crucial in today's fast-evolving cyber threat landscape. By analyzing network traffic, it identifies patterns matching known attack signatures or deviations from normal behavior, enabling swift responses. For instance, it can detect a sudden spike in traffic from an unusual IP address, potentially indicating a Distributed Denial-of-Service (DDoS) attack. This early warning reduces the risk of data loss and system compromise, with real-time alerts enhancing incident response times.

- **Part of Layered Security**

Cybersecurity relies on a layered defense strategy, and IDS plays a crucial role within this ecosystem. For example, IDS can complement firewalls by detecting threats that evade primary security measures. While firewalls block unauthorized access based on predefined rules, they might miss sophisticated attacks. IDS fills this gap by analyzing the content of network packets and identifying unusual patterns. Another example of IDS integration is its combination with Security Information and Event Management (SIEM) systems. SIEM collects and correlates data, alerts from various sources, including IDS, to provide a comprehensive view of security events. This integration enhances threat detection and response capabilities, allowing organizations to identify complex attack patterns that may go unnoticed by individual security tools.

- **Regulatory Compliance**

Many industries must follow specific security standards in their region, such as the Payment Card Industry Data Security Standard (PCI-DSS), which requires IDS implementation. Having an IDS helps organizations meet these requirements and avoid penalties. This is especially important in sectors like finance and healthcare, where data breaches can have serious consequences.

- **Monitoring and Analysis**

IDS offers comprehensive monitoring capabilities across the organization infrastructure by combining different approaches to ensure robust security. For example, Network-based IDS (NIDS) focuses on analyzing traffic across the entire organizational network. It is typically deployed at strategic points, such as behind firewalls or at network entry and exit points, to detect suspicious activities like unauthorized access attempts or Distributed Denial-of-Service (DDoS) attacks. On the other hand, Host-based IDS (HIDS) operates at the individual host level, monitoring specific devices such as servers, workstations, or routers. HIDS is particularly effective at detecting unauthorized file modifications, unusual processes, or unexpected system behavior on the host. By integrating both NIDS and HIDS, organizations gain a holistic security view, enabling them to detect threats at both the network and host levels, thereby enhancing their overall defense posture.

- **Alerting and Reporting**

Intrusion Detection Systems (IDS) play a crucial role in cybersecurity by generating alerts and reports that enable proactive threat mitigation. These alerts notify security teams of potential breaches, allowing for swift responses to minimize damage. A notable example is the 2022 Shields Health Care Group breach, where a delayed IDS alert contributed to the exposure of 2 million patients' data, underscoring the importance of timely detection. IDS logs also support event correlation, network forensics, and compliance audits, strengthening incident response and digital investigations. Effective alerting and reporting enhance an organization's ability to detect, analyze, and respond to security incidents efficiently.

2.1.3 General architecture of an IDS

A typical IDS has three main components:

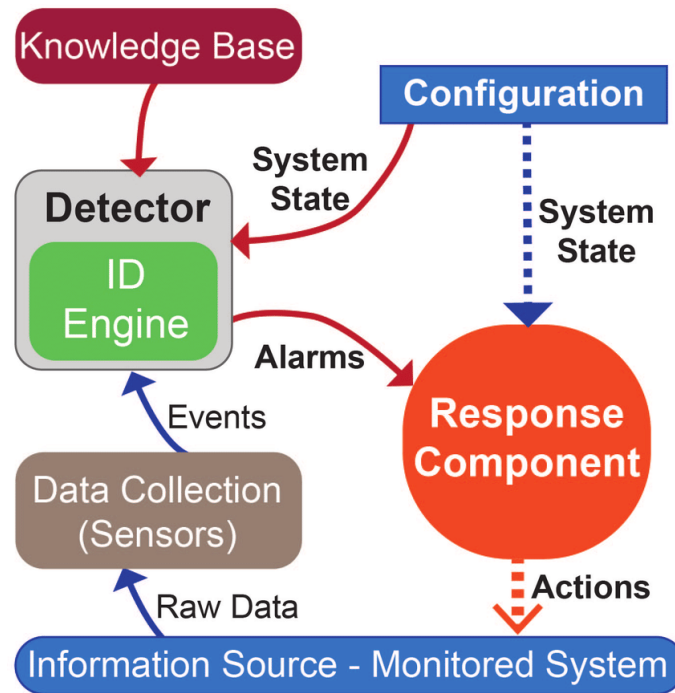


Figure 2.1: General Architecture of an IDS

- **Sensors or Agents:** These components collect data from various sources such as network traffic, system logs, or application activities. *Monitored network nodes are typically referred to as sensors, while monitored endpoint devices such as laptops are commonly referred to as agents.*
- **Analysis Engine:** This component processes the collected data using detection methods to identify potential security threats.
- **Response System:** Once a threat is detected, this component generates alerts and reports for the security team to take action.

2.2 Types of ID

IDSs are categorized based on their placement within a system or the types of activities they monitor:

2.2.1 Network-based IDS (NIDS)

A Network-based Intrusion Detection System (NIDS) monitors traffic across an entire network, analyzing data packets or copies of them for suspicious activity.

a, Placement

NIDS is strategically deployed at critical locations within the network to enhance visibility and threat detection. Key placement points include:

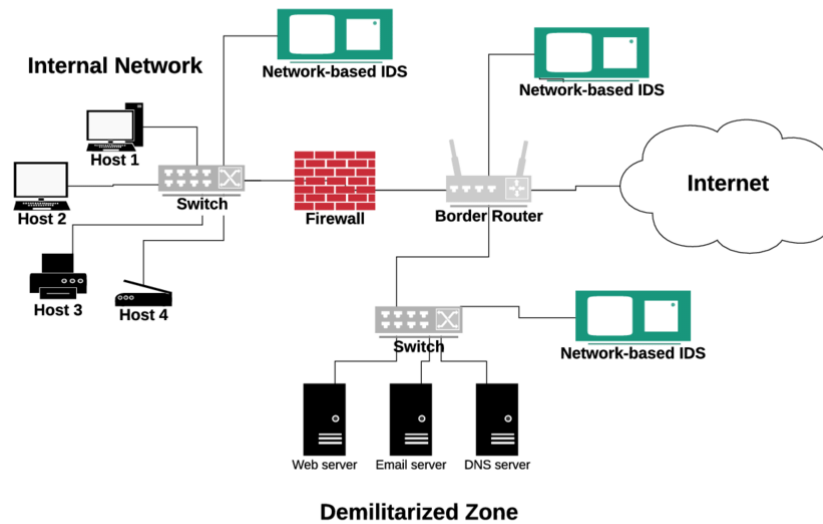


Figure 2.2: NIDS placement within a network

- Behind firewalls behind firewalls: Allowing them to flag malicious traffic that penetrates initial defenses.
- Between different network segments: Monitor lateral movement within the network, detecting potential insider threats or compromised hosts.
- At network entry and exit points: Observes inbound and outbound traffic between subnets, identifying unauthorized external communications or data ex-filtration attempts.

To maintain network performance, a NIDS is typically deployed out-of-band, meaning that network traffic does not flow directly through it. Instead, the NIDS analyzes copies of network packets rather than intercepting the original packets themselves. In enterprise environments, this is commonly achieved using a feature called SPAN (Switched Port Analyzer), also known as Port Mirroring. This feature on network switches allows traffic from one or more source ports to be mirrored to a designated destination port — the SPAN port — which is typically connected to a monitoring device such as an IDS. This configuration ensures that legitimate traffic flows without delay while the NIDS can still identify and flag malicious activity.

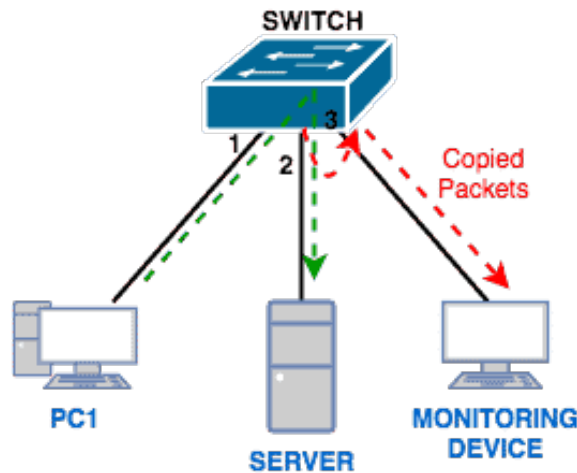


Figure 2.3: Port Mirroring (SPAN) on Switch

b, What NIDS does

NIDS can help protect the entire network by monitoring network traffic for malicious activity. They analyze packets for signs associated with a wide range of hostile or unwanted behaviors, including:

- Malware command and control
- Exploitation tools
- Scanning
- Data exfiltration
- Contact with phishing sites
- Corporate policy violations

c, How NIDS works

1. **Traffic Mirroring:** NIDS does not sit in-line with network traffic, but instead operates out-of-band. This is typically achieved using techniques such as:
 - **SPAN (Switched Port Analyzer) / Port Mirroring:** Copies traffic from one or more switch ports to a designated monitoring port.
 - **Network TAP (Test Access Point):** Hardware device that splits network traffic for monitoring purposes without affecting performance.
2. **Packet Capture:** NIDS sensors capture copies of network packets in real time. These packets are parsed, decoded, and analyzed against a set of detection rules and signatures.

NIDS Data Sources	Examples
Network traffic (packet data)	HTTP, DNS, SMB, FTP, etc.
Packet headers	IP, TCP/UDP, port numbers, flags
Payload content	Command and control strings, exploit signatures
Network flow metadata	Flow direction, duration, byte count

Table 2.1: Main data sources for NIDS

3. **Analysis and Detection:** After capturing the network packets, the system extracts relevant information such as headers, payloads, and metadata. This data is then passed to the detection engine for analysis. The inner workings of the detection engine, including techniques such as signature-based and anomaly-based detection, will be discussed in a later section of this report.
4. **Alerting and Logging:** Once suspicious activity is detected, NIDS logs the event and generates alerts for the security operations team. Integration with SIEM platforms allows for centralized analysis and correlation.

Some common network-based IDS can be named as: Snort, Suricata, Zeek, Security Onion, etc.

2.2.2 Host-based IDS (HIDS)

Not all malicious activities generate detectable network traffic, meaning a NIDS might not see these certain threats. For example, local attacks like privilege escalation or file tampering often leave no network trace. This is why Host-based Intrusion Detection Systems was .

A Host-based Intrusion Detection System (HIDS) is deployed on individual endpoints such as computers, servers, or routers to monitor local activity and detect potential security threats. Unlike Network-based IDS (NIDS), which observes network-wide traffic, HIDS focuses exclusively on the host system, analyzing incoming and outgoing traffic, file system changes, system logs, and resource usage.

a, What HIDS does

Key monitoring functions of HIDS include:

- **File integrity monitoring:** Detects unauthorized modifications to critical system files, configuration settings, and script additions.
- **Process and application monitoring;** Identifies suspicious or unauthorized programs running on the host, e.g., malware execution
- **Login anomaly detection:** Flags unusual authentication attempts, such as brute-

force login attempts or logins from unexpected locations.

- **Behavioral analysis:** Observes deviations from normal system behavior that may indicate compromise.

b, How HIDS works

1. **Agent Installation:** HIDS typically requires the installation of an agent on each machine that needs to be monitored.
2. **Data Collection and Transmission:** The agent on each host collects data from the following system sources and sends it to a central processing node:

HIDS Data Sources	Examples
Application and system logs	/var/log/, Windows Event Logs
Windows Registry	Sensitive changes
System performance	Abnormal CPU, RAM usage
File system status	Unauthorized file modifications or creations

Table 2.2: Main data sources for HIDS

Additionally, HIDS typically operates by taking periodic snapshots of critical system files and comparing checksum over time to detect unauthorized changes.

3. **Analysis and Alerts:** The central node uses detection engines (similar to NIDS) to analyze the data and trigger alerts for the security team to investigate when suspicious activity is detected.

By providing in-depth visibility into host activities, HIDS enhances endpoint security, aiding in threat detection, forensic analysis, and compliance enforcement.

Deploying HIDS in large environments is complex because:

- It requires managing many agents across multiple hosts.
- Manual configuration is required to properly monitor specific applications — many default applications may be overlooked.

Solution: Using automation tools like **Ansible** for auto deployment, configuration, and management of HIDS agents. In reality, without automation, deploying and maintaining HIDS for each individual container would be infeasible.

Some common host-based IDS can be named as: Wazuh, OSSEC, Tripwire, Sagan etc.

c, Different between NIDS and HIDS

The figure below illustrates different deployment locations of NIDS and HIDS within a network infrastructure.

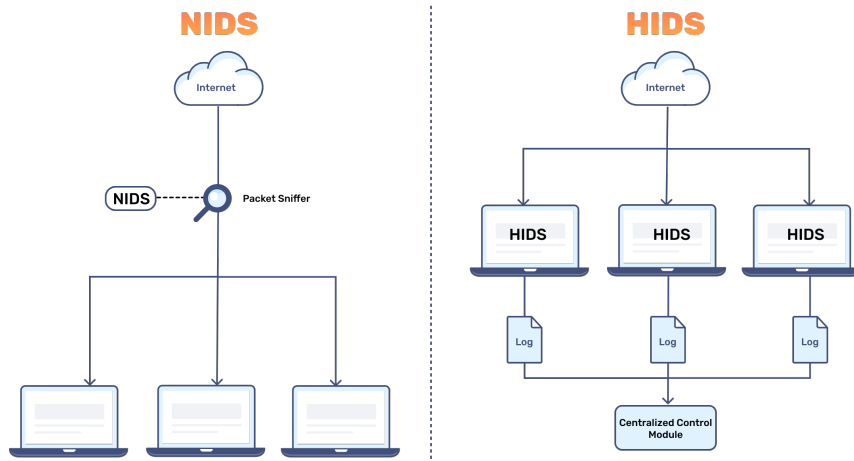


Figure 2.4: NIDS (left) vs. HIDS (right)

To better understand their differences in functionality and deployment, Table 2.3 provides a comparative analysis of NIDS and HIDS in terms of key features such as scope, installation, detection methods, and response time.

Table 2.3: Comparison Between Network-based IDS (NIDS) and Host-based IDS (HIDS)
[1]

Feature	NIDS (Network-based IDS)	HIDS (Host-based IDS)
Coverage	Monitors traffic across the entire network (wide scope)	Monitors activity on a specific host (narrow scope)
Installation	Deployed at strategic network locations; often on a dedicated server connect to switch' SPAN port	Installed on many individual endpoints; using automation tools to setup on devices
Cost	Generally more expensive due to additional infrastructure and maintenance needs	More cost-effective as it operates on individual systems
Data Source	Analyzes network traffic, including packet headers and payloads. Inspects network packets for anomalies and attack signatures	Relies on system logs, file integrity monitoring, and process analysis
Response Time	Provides real-time alerts upon detecting suspicious network activity	Detects attacks post-execution, generating alerts after system changes occur
OS Dependency	Operating system independent	OS-dependent, tailored to the host system's architecture
Attack Assessment	Focuses on attack attempts and network-based threat detection	Evaluates the success and impact of attacks on the host
Visibility into encrypted	No	Yes (at host)
Impact on host performance	No	Potentially high

2.2.3 Hybrid IDS

Hybrid Intrusion Detection Systems (Hybrid IDS) integrate multiple intrusion detection approaches, typically combining Network-based IDS (NIDS) and Host-based IDS (HIDS), to provide a more comprehensive security solution.

While NIDS monitors network traffic for suspicious activity across multiple hosts, HIDS provides granular monitoring of specific devices, including file integrity checks and log analysis. This dual-layered approach enhances detection accuracy and mitigates blind spots inherent in standalone NIDS or HIDS implementations.

For example, NIDS can detect unusual network behavior, such as repeated failed login attempts from an external source, while HIDS can verify whether those attempts resulted in unauthorized access to a critical system. Additionally, HIDS

can identify malware propagation from a compromised machine, complementing NIDS by providing endpoint-level insights.

Hybrid IDS solutions are generally more robust than single-method detection systems, leveraging both signature-based and anomaly-based detection techniques.

2.2.4 Other Specialized IDS Types

In addition to NIDS, HIDS, and Hybrid IDS, several specialized IDS types exist to address unique security concerns that less commonly used:

- **Protocol-based IDS (PIDS):** Monitor communication protocols (e.g., HTTP, HTTPS) to detect protocol anomalies or attacks. It is commonly deployed on web servers to identify malicious HTTP requests, such as SQL injection attempts. With HTTPS protocol, it requires placement at the point where encrypted traffic is decrypted to analyze HTTPS connections effectively.
- **Application Protocol-based IDS (APIDS):** Operate at the application layer to monitor application-specific protocols. Typically placed between critical components, such as a web server and an SQL database, to detect anomalies like unauthorized SQL queries. This type of IDS is useful in preventing attacks such as SQL injection and API exploitation

2.3 Detection Methods

IDSs can be deployed as software applications, dedicated hardware appliances, or cloud-based services. They primarily use two core detection methodologies:

a, Signature-based detection

Signature-based detection, also known as Rule-based detection or Misuse Detection[2], relies on predefined signatures—unique patterns representing known attack behaviors. IDS scans network traffic for matches against these signatures and triggers alerts upon detection.

- **Advantages:**

- Highly effective against known threats.
- Low false positive rate when signatures are well defined.

- **Disadvantages:**

- It is challenging to build a rule set that is flexible enough to reliably detect malicious traffic, while avoiding false positives from legitimate applications that may exhibit similar behavior. This process requires an experienced administrator, careful tuning of the IDS to match the specific characteristics of the network it is monitoring, as well as regular audits

consistently applied.

- The effectiveness of signature-based detection depends heavily on frequent updates to the signature database with the latest threat intelligence. As new cyber attacks appear and known attack methods evolve, the system must be updated regularly. A major limitation of this approach is that it cannot detect unknown (zero-day) attacks that have not yet been analyzed and included in the signature database.

Signature-based systems identify characteristic patterns derived from statistical analyses. These signatures do not need to match 100 percent, as security professionals aim to create signatures with high generalization capabilities that can detect attacks even with minor modifications

2.3.1 Anomaly-based detection

On the other hand, Anomaly-based IDS, also known as “Behavior-based Detection” [2], depend on identifying patterns. With advances in artificial intelligence and particularly machine learning, deep learning in recently years, anomaly-based detection has become increasingly sophisticated and widely adopted. This method employs machine learning or deep learning algorithms to establish and continuously refine a baseline model of normal network activity. The system then compares ongoing network activity against this model, flagging deviations such as processes consuming abnormal bandwidth or devices opening unexpected ports. For more specific, the workflow is:

1. Establish a baseline of normal network behavior
2. Flag activities that deviate from this normal pattern
3. Detect potentially unknown threats, including zero-day exploits

This modern method offers several advantages over traditional signature-based detection, including:

- **Advantages:**

- **Detection of Unknown Threats:** Ability to detect previously unknown threats that might evade signature-based detection, including zero-day exploits—attacks that exploit software vulnerabilities before developers become aware of them or implement patches.
- **Adaptability:** Anomaly-based systems can adapt to changing network environments and evolving attack techniques, making them more resilient against sophisticated threats.

- **Disadvantages:**

May generate more false positives than signature-based approaches. Even legitimate activities, such as authorized users accessing sensitive resources for the first time, might trigger alerts in an anomaly-based system.

Another problem with Heuristic-based IDS, a type of anomaly detection, which focuses on unusual activities and atypical user behaviors. For sophisticated intrusion attempts, individual malicious actions may appear normal and not raise any suspicion. However, when these actions are combined in a specific sequence, they can indicate an ongoing attack. In such cases, a system might gradually fall/manipulated under the control of an intruder through a series of seemingly legitimate operations. To address this issue, engineers developed a tool which is called **inference engines**, which link multiple suspicious actions into a coherent pattern. These engines can operate using one of two fundamental principles: state-based detection or model-based detection. We will not go into the details of these principles here, but the general idea is to detect suspicious behavior sequences that could potentially lead system to an unsafe state [3].

Despite significant improvements in AI-driven detection, anomaly-based IDS still require careful tuning to balance accuracy and reduce false positives. This tuning process is often complex and ongoing, as it must adapt to an organization's specific environment and evolving threat landscape.

To enhance detection capabilities, some IDS solutions combine signature-based and anomaly-based methods, creating a hybrid model that can detect both known and unknown threats. These IDS can operate in either a stateless or stateful manner. Stateless IDS analyze individual packets independently, while stateful IDS track entire network flows to provide better context for threat detection. Modern IDS implementations increasingly favor stateful approaches, as they offer deeper insights into traffic patterns and attack behaviors.

The table below outlines the key differences between signature-based and anomaly-based IDS structures:

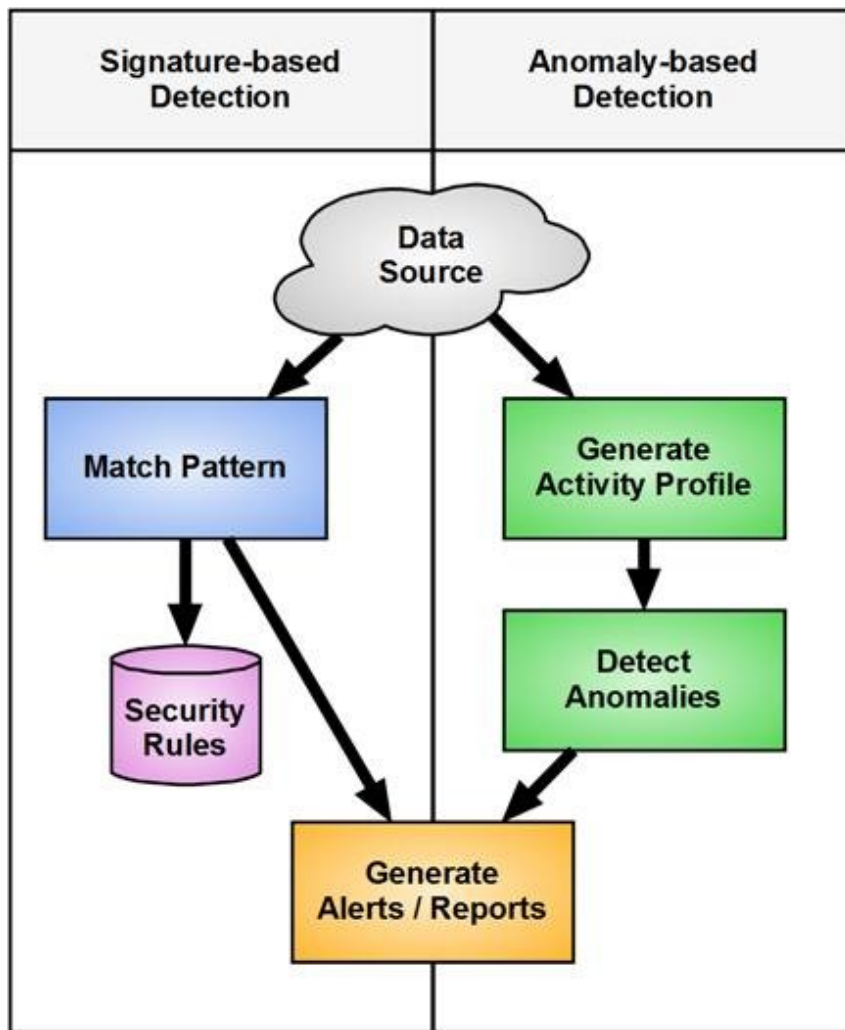


Figure 2.5: Signature and Anomaly based

2.3.2 Other Detection Methods

Some less common detection techniques include:

- **Reputation-based:** Blocks traffic from IP addresses and domains with poor reputation scores based on historical threat intelligence.
- **Stateful protocol analysis:** Detects protocol-based anomalies, such as denial-of-service (DoS) attacks that exploit TCP/IP protocol vulnerabilities.

2.4 IDS Evasion Techniques

Despite their effectiveness, Intrusion Detection Systems are not infallible. Sophisticated attackers employ various techniques to circumvent IDS detection. Understanding these evasion techniques is crucial for organizations to strengthen their security posture and implement appropriate countermeasures. Some common IDS evasion methods:

- **Distributed denial-of-service (DDoS):** An Intrusion Detection System (IDS)

operates as either a dedicated network device or software running on a computer, leaving it susceptible to attacks aimed at disabling its protective features. In a typical scenario, a threat actor who has successfully breached the system will first quickly try to deactivate the IDS to avoid detection while targeting the broader network. A common Distributed Denial of Service (DDoS) tactic involves flooding the IDS with spoofed User Datagram Protocol (UDP) and Internet Control Message Protocol (ICMP) traffic [3]. This traffic is designed to look obviously malicious, thus overwhelming the IDS's processing capacity. As the IDS struggles to handle these fake threats, attackers can exploit this distraction to execute their actual attacks undetected. This technique effectively creates a smokescreen of activity that conceals the real intrusion attempt.

To address this, there is a need to implement IDS in stealth mode. In this configuration, the IDS operates with two network interfaces:

- A monitoring interface that passively observes network traffic without interacting with it. This interface operates with a hidden address that is not published anywhere, not even to internal routers. It simply allows packets to pass through while recording activity.
- A separate alert interface that operates on an entirely different network. When the IDS detects suspicious activity, it uses this second interface to issue alerts and warnings to the security team.

This dual-interface design significantly enhances IDS resilience against direct attacks, as attackers cannot easily identify and target the monitoring components of the system.

- **Spoofing:** Spoofing techniques involve falsifying IP addresses and DNS records to make malicious traffic appear to originate from trusted sources. By impersonating legitimate entities, attackers can bypass IDS rules that might otherwise flag suspicious traffic from unknown origins. This technique exploits the trust relationships established within networks. For example, an attacker might spoof the IP address of a trusted internal server to gain unauthorized access to sensitive data or resources.
- **Fragmentation:** Fragmentation involves splitting malicious payloads into smaller packets to obscure their signatures and evade detection. By strategically delaying packet transmission or sending packets out of sequence, attackers can prevent the IDS from properly reassembling and analyzing the complete payload. When the IDS cannot reconstruct the fragmented packets

correctly, it fails to recognize the attack pattern.

- **Packet Encoding:** Encoding packets using methods such as Base64 or hexadecimal representation can effectively hide malicious content from signature-based IDS solutions. These encoding techniques transform the data in ways that may not match known attack signatures while preserving the payload's functionality when executed at the destination. By encoding malicious payloads, attackers can bypass signature-based detection and deliver harmful content to target systems.
- **Traffic Obfuscation:** Obfuscation techniques make network traffic difficult to interpret, thereby concealing attacks from detection systems. The process involves altering program code or network traffic in ways that maintain functional equivalence while significantly reducing readability and detectability. Obfuscated malware, for instance, can bypass IDS detection while retaining its malicious capabilities. The objective is to reduce detectability to reverse engineering or static analysis process by obscuring it and compromising readability. Obfuscating malware, for instance, allows it to evade IDSes.
- **Encryption:** Encrypted protocols present a significant challenge for many IDS solutions. If the IDS does not possess the corresponding decryption keys or capabilities to inspect encrypted traffic, attackers can use encryption to transmit malicious payloads without detection. As encryption becomes more widespread for legitimate traffic, this technique becomes increasingly effective.
- **Operator fatigue:** Attackers may deliberately generate large volumes of IDS alerts by performing non-threatening but unusual actions across the network. This approach creates "operator fatigue," overwhelming security teams with false positives and distracting them from identifying and responding to actual malicious activities occurring simultaneously.

2.5 IDS Integration with Other Security Tools

IDSs are not standalone tools. They are designed to be part of a holistic cyber security system, and are often tightly integrated with one or more of the following security solutions. Therefore, they can sometimes be confused with these other tools. Here's a quick overview of how IDSs differ from these other security solutions:

a, IDS vs. IPS

While Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) share similar detection capabilities, they differ significantly in their response mechanisms and network placement.

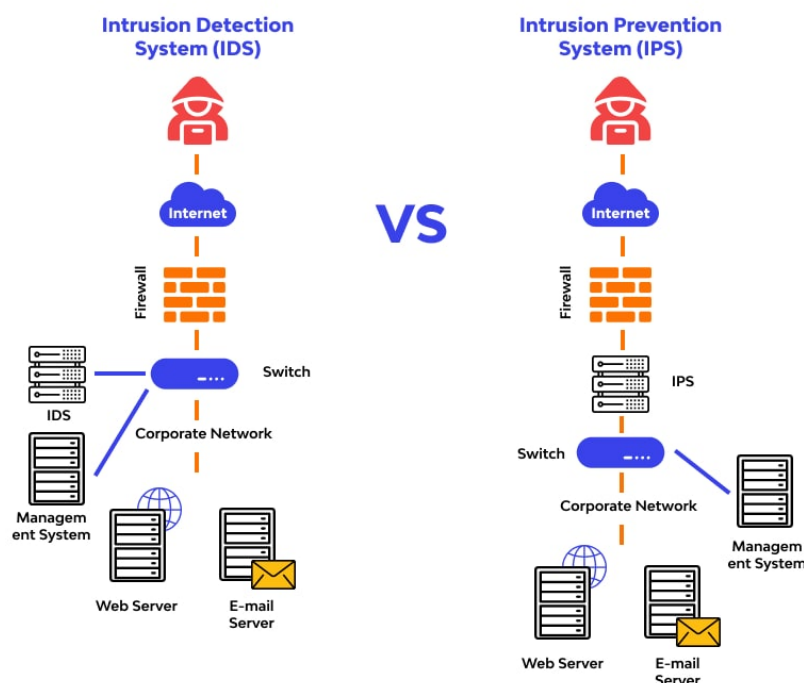


Figure 2.6: IDS (left) vs IPS (right)

Table 2.4 highlights these key differences.

Table 2.4: Comparison of IDS and IPS Characteristics

Characteristic	Intrusion Detection System (IDS)	Intrusion Prevention System (IPS)
Placement in Network	Out-of-band (outside direct communication path)	Inline (part of direct communication path)
System Type	Passive (monitor and notify)	Active (monitor and automatically defend)
Response Capability	Detection only; cannot prevent exploits	Detection and automatic prevention
Traffic Impact	Minimal impact on network performance	Can potentially affect network throughput
Detection Mechanisms	Primarily signature-based detection	Both signature-based and anomaly-based detection

An IDS operates as a listen-only device that monitors traffic and reports findings to administrators but cannot independently take action to prevent detected exploits.

In contrast, an IPS can automatically respond to threats by blocking malicious traffic, resetting connections, or taking other preventative measures.

Many modern organizations implement both capabilities within a single Intrusion Detection and Prevention System (IDPS), which combines detection, logging, alerting, and automated response functions.

2.5.1 IDS vs. Firewall

While firewalls and IDS serve different primary functions, they work together to create a more robust security posture. Most security mechanisms previously discussed, such as user authentication and firewalls, focus on preventing attacks from external sources. However, a significant percentage of documented destructive attacks originate internally, either from malicious users or from external attackers who have successfully penetrated these protective barriers.

How firewall benefit from IDS

Firewalls serve as the first layer of protection, functioning as access control barriers by using predefined rule sets to permit or block traffic based on source, destination, protocol, and port information. They actively filter traffic but may not detect sophisticated attacks that conform to allowed traffic patterns.

In contrast, an IDS monitors traffic that has already passed through the firewall, detecting suspicious activities that might indicate an attack in progress. While firewalls reject traffic that violates security rules, they cannot detect attacks originating from within the network or those disguised as legitimate traffic—scenarios where an IDS proves invaluable.

Why they are better together ?

- IDS helps firewalls: Catches sneaky attacks that slip past firewall rules, providing feedback to improve firewall policies over time.
- Firewalls help IDS: By blocking obvious threats (e.g., spam or port scans), firewalls reduce "noise," letting the IDS focus on complex analysis instead of getting overwhelmed by easily blocked traffic (e.g., port scans).

The complementary relationship between these technologies can be understood through an analogy: A firewall is like a security checkpoint controlling entry, whereas an IDS functions like surveillance cameras inside the premises, analyzing activities post-entry.

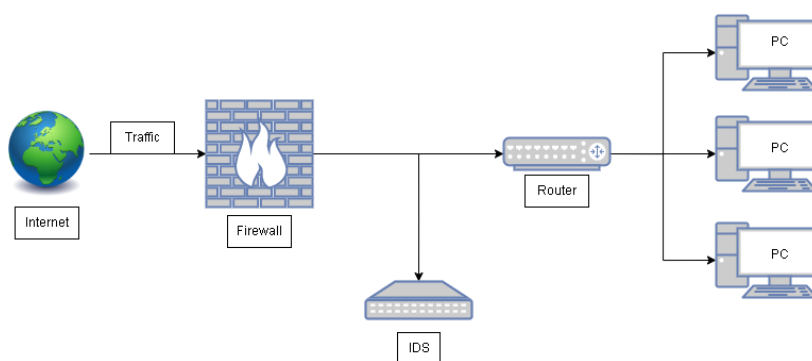


Figure 2.7: Firewall and IDS in a network

In a home lab environment, a common open-source setup often consists of a combination of Snort or Suricata (IDS) and pfSense (firewall).

2.5.2 IDS and SIEM

An IDS acts as a watchful guard, scanning network traffic in real time to spot suspicious activity, such as malware signatures or unusual data flows. Meanwhile, a SIEM system operates like a central command center, collecting and analyzing data from every corner of an organization’s infrastructure—firewalls, servers, applications, and even the IDS itself. Together, they bridge the gap between real-time detection and holistic threat analysis.

Why they are better together ?

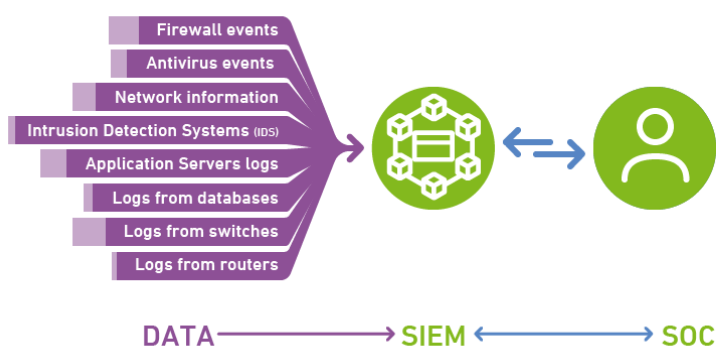


Figure 2.8: IDS as a data source of SIEM

- **SIEM Enhances IDS:** A SIEM adds critical context to IDS alerts. For example, if an IDS flags a sudden surge in data transfers, the SIEM cross-references this event with user authentication logs, geographic IP data, and historical patterns to determine whether it’s a legitimate backup process or a potential data breach. It also automates responses, such as quarantining a compromised device, and stores long-term data to identify trends (e.g., recurring attack pat-

terns) that an IDS might overlook in the moment.

- **IDS Enhances SIEM:** The IDS feeds real-time threat data into the SIEM, acting as its "eyes and ears" on the network. This allows the SIEM to prioritize alerts more effectively—for instance, distinguishing harmless network noise from a coordinated brute-force attack. The IDS also detects subtle anomalies, like encrypted traffic hiding malicious payloads, which the SIEM can then correlate with other events (e.g., unusual login times or privileged account activity) to uncover advanced threats.

In practical, some integrated Solutions like Security Onion bundle IDS tools (Suricata, Zeek) with SIEM capabilities (Elastic Stack) for an all-in-one network monitoring suite.

2.5.3 IDS and Honeypots

Intrusion Detection Systems (IDS) and honeypots serve distinct but complementary roles in cybersecurity. While an IDS monitors real network traffic for signs of malicious activity, honeypots act as bait designed to attract attackers and reveal their tactic. These intelligent can be used to strengthen IDS real-time defense Together, they create a more robust security posture that combines detection with active threat intelligence gathering.

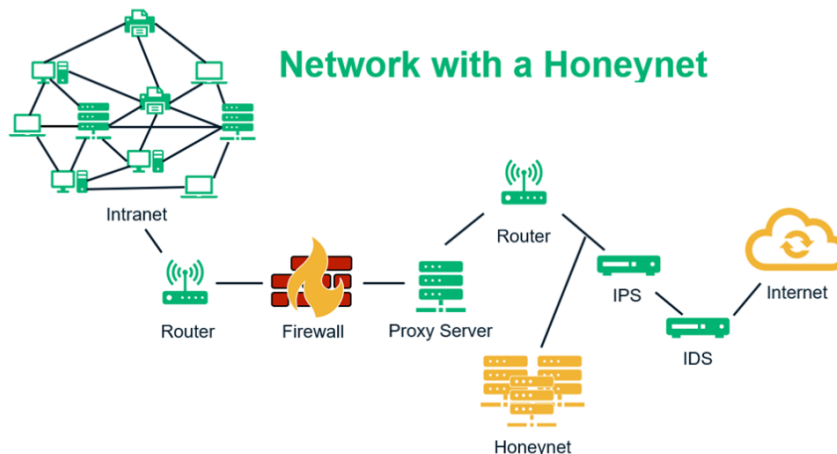


Figure 2.9: IDS combines with Honeypots within a network

Honeypots provide several key benefits to IDS operations:

- They reduce false positives by offering unambiguous evidence of malicious intent - any interaction with a decoy system warrants investigation
- They serve as early warning systems, often detecting attack reconnaissance long before critical systems are targeted
- They provide safe environments for IDS to study about emerging threats with-

out risking real production assets.

In practice, security teams often deploy honeypots in strategic network segments - near critical assets, in DMZs, or in seemingly neglected areas that might attract attackers. When integrated with an IDS, any activity detected in these honeypots can trigger enhanced monitoring of related network segments or specific security controls. Some practical integration can be tools like T-Pot (honeypot suite) and Suricata (IDS) work in tandem.

2.6 Some Popular IDS/IPS In The Industry

- **Snort**

- **Ownership and History:** Owned by Cisco, Snort was first released in 1998, making it one of the earliest open-source IDS/IPS tools. It gained prominence through its acquisition by Sourcefire, which Cisco later acquired in 2013.
- **Technical Details:** Snort operates as a network-based IDS/IPS, using a rule-based language for signature-based detection. It supports real-time traffic analysis, packet logging, and can be configured as a sniffer, logger, or full intrusion prevention system. Its community ruleset and subscriber ruleset enhance its detection capabilities.
- **Use Case:** Widely used for monitoring network traffic, detecting known threats like buffer overflows and stealth port scans, and integrating with SIEM systems for broader analysis. See more detail about Snort via [this link](#)

- **Suricata**

- **Ownership and History:** Developed by the Open Information Security Foundation (OISF), Suricata was first released in beta in 2009, with a standard release in 2010. It is maintained by a global community, emphasizing open-source collaboration.
- **Technical Details:** A network-based IDS/IPS, Suricata offers multi-threaded performance for high-speed traffic analysis, featuring multi-threaded architecture, GPU acceleration, deep packet inspection, and file extraction. It supports both signature-based and anomaly-based detection, making it a robust alternative to Snort. Suricata is compatible with Snort rules, making migration seamless.
- **Use Case:** Ideal for enterprises needing high-performance network security monitoring, with integration capabilities for tools like Wazuh for

enhanced endpoint visibility. See more detail about Suricata via [this link](#)

- **Zeek**

- **Ownership and History:** Formerly known as Bro, originating in 1995 at Lawrence Berkeley National Lab under Vern Paxson, Zeek is now maintained by the Zeek community. It is known for its long history in network security monitoring.
- **Technical Details:** Zeek is a network-based security monitor, not a traditional IDS, focusing on log generation and scriptable analysis. It supports semantic misuse detection, anomaly detection, and behavioral analysis, with over 70 default log files and 3,000+ network events tracked.
- **Use Case:** Used by universities, research labs, and enterprises for detailed network traffic analysis, often integrated with SIEM systems for incident response. See more detail about Zeek via [this link](#)

- **OSSEC:**

- **Ownership and History:** Created by Daniel B. Cid in 2003, OSSEC is now managed by Atomicorp, a company providing additional support and commercial versions. It was acquired by Third Brigade in 2008, with open-source version is popular for small teams, while Atomic OSSEC offers enterprise scalability.
- **Technical Details:** A host-based IDS, OSSEC focuses on log analysis, file integrity monitoring, Windows registry monitoring, rootkit detection, and active response. It runs on multiple platforms, including Linux, Windows, and MacOS, with a centralized architecture for distributed monitoring.
- **Use Case:** Suitable for organizations needing endpoint security, compliance auditing (e.g., PCI-DSS, HIPAA), and real-time alerting for host-level threats.. See more detail about OSSEC via [this link](#)

- **Wazuh**

- **Ownership and History:** Released in 2018 as a fork of OSSEC, Wazuh is maintained by Wazuh Inc., offering both free and paid services to users. It has grown in popularity, with thousands of enterprise users.
- **Technical Details:** Primarily host-based, Wazuh provides log analysis, file integrity monitoring, vulnerability detection, and incident response. It integrates with network IDS like Suricata for broader coverage, offer-

ing SIEM and XDR capabilities. It also integrates with Elasticsearch for visualization and supports cloud, on-premises, and hybrid environments. Wazuh is known for its lightweight agents and automated threat response.

- **Use Case:** Used for endpoint protection, cloud workload monitoring, and compliance, with flexible deployment options for on-premise and cloud environments. See more detail about Wazuh via [this link](#)

- **Security Onion**

- **Ownership and History:** Created by Doug Burks in 2008, Security Onion is maintained by Security Onion Solutions, with over 2 million downloads. It is a Linux distribution based on Ubuntu, designed for security professionals.
- **Technical Details:** Combines network and host-based tools, including Snort, Suricata, Zeek, Wazuh, and others, for intrusion detection, network security monitoring, and log management. It uses the Elastic Stack for analysis, Sguil for alert management, Kibana for dashboards and offers full packet capture via netsniff-ng. Ideal for centralized intrusion detection and log analysis.
- **Use Case:** Ideal for enterprises needing a scalable, open-source platform for distributed sensor grids, used in home offices, governments, and Fortune 500 companies. See more detail about Security Onion via [this link](#).

- **Cisco Secure IDS**

- **Ownership and History:** Owned by Cisco, with roots in the 1990s, Cisco Secure IDS has evolved into the Cisco Secure Firewall, which includes IPS features. It is part of Cisco's broader security portfolio, leveraging Cisco Talos for threat intelligence.
- **Technical Details:** Network-based, it uses signature databases for intrusion detection, now integrated with firewall capabilities for inline prevention. It offers simplified configuration and real-time event reporting.
- **Use Case:** Suitable for large enterprises needing integrated network security. See more detail via [this link](#).

- **IBM QRadar IDS**

- **Ownership and History:** Developed by Q1 Labs in 2007, acquired by IBM in 2011, QRadar is a commercial SIEM platform, recognized as a leader in the 2024 Gartner Magic Quadrant for SIEM for the 14th consec-

utive time.

- **Technical Details:** Primarily a SIEM, it integrates with IDS/IPS for log collection, event correlation, and threat prioritization using AI and machine learning. It supports federated search across cloud and on-premises data.
- **Use Case:** Used for centralized security management, compliance, and incident response, with a focus on reducing noise and enhancing SOC analyst productivity. See more detail about IBM QRadar IDS via this [link](#).

- **McAfee Network Security Platform, now Trellix:**

- **Ownership and History:** Originally part of McAfee, with significant developments around 2010, it became part of Trellix following the merger with FireEye in 2021. Trellix focuses on integrated cybersecurity solutions.
- **Technical Details:** Network-based, it offers next-generation IPS with signature-less technologies, machine learning for real-time threat detection, and fail-open capabilities for high availability.
- **Use Case:** Suitable for organizations needing advanced threat protection, with integration into broader security ecosystems for comprehensive defense. See more detail about McAfee via this [link](#).

- **Trend Micro TippingPoint**

- **Ownership and History:** Founded in 1999 as TippingPoint Technologies, acquired by Trend Micro in 2015, it is part of Trend Micro's network security offerings, with a focus on real-time threat protection.
- **Technical Details:** Network-based, it uses deep packet inspection, threat reputation, URL reputation, and machine learning for detecting and blocking malicious traffic. Renowned for its Digital Vaccine® threat intelligence, offering real-time vulnerability protection and automated virtual patching. Excels in high-availability environments with fail-safe bypass mechanisms. Integrates with Trend Micro's Vision One platform.
- **Use Case:** Ideal for enterprises needing proactive network security, with flexible deployment options and centralized management. See more detail about Trend Micro via this [link](#).

Summary Table: [4]

Name	Company Owned	Year	Open-source	Primary Focus	Cost
Snort	Cisco	1998	Yes	Network-based	Free, subscription (\$30/yr personal, \$300/yr business)
Suricata	OISF	2010	Yes	Network-based	Free (GPL v2 license)
Zeek	Community	1995	Yes	Network-based	Free*
OSSEC	Atomicorp	2003	Yes	Host-based	Free, \$5/agent/month for commercial version
Wazuh	Wazuh Inc.	2018	Yes	Host-based	Free (open-source), commercial support available
Security Onion	Security Onion Solutions	2008	Yes	Both	Free*
Cisco Secure IDS	Cisco	1990s	No	Network-based	Starts at \$5,000 (varies by model)
IBM QRadar	IBM	2007	No	SIEM	Contact vendor (based on events per second)
McAfee Network Security	Trellix	2010	No	Network-based	\$10,995+
Trend Micro TippingPoint	Trend Micro	1999	No	Network-based	Contact vendor for pricing

Table 2.5: Summary Table of IDS/IPS Solutions

**Support or preloaded appliances available from 3rd party vendors for a fee.
All pricing data is collected as of March 31, 2025*

In the following chapters, we will explore Suricata in detail, including installation, configuration, and set up a lab environment.

CHAPTER 3. RESEARCH ON SURICATA IDS

3.1 Introduction to Suricata

3.1.1 What is Suricata



Figure 3.1: Suricata logo

Suricata is an open-source tool developed by the Open Information Security Foundation (OISF). It can work as an Intrusion Detection System (IDS) and Intrusion Prevention System (IPS). It monitors network traffic, identifies suspicious activities, and alerts system administrators when potential threats are detected. Suricata is a flexible and efficient tool, integrating well with modern network security setups therefore offering powerful options for advanced network security architectures.

As a multi-threading tool, Suricata excels at analyzing network traffic. It's optimized to work with modern multi-core CPUs or GPU, ensuring high performance, especially with compatible network cards and software.

3.1.2 Development History

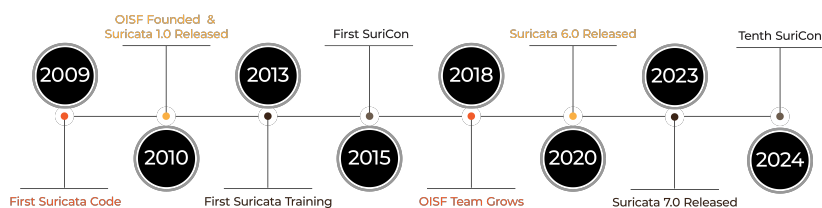


Figure 3.2: Suricata Timeline

Origins and Early Development

The development of Suricata began in 2007, initiated by Victor Julien and Matt Jonkman, with contributions from William Metcalf. This early phase involved creating the initial code and presenting a prototype in early 2008, setting the stage for future advancements. The collaborative effort during this period was crucial in establishing the technical foundation for Suricata's capabilities.

Initial Releases and Growth

The first public beta version of Suricata was released on December 31, 2009, marking a significant step toward public testing and feedback. This beta release was announced on platforms like Slashdot, indicating early community interest. Following this, the first standard release, version 1.0, was launched in July 2010. This release cycle established Suricata as a viable tool, with major updates typically occurring every three months, reflecting a commitment to continuous improvement. For example, the first public, in-person training in 2013 at Hack.lu in Luxembourg, and the inception of SuriCon conferences starting in 2015 has made Suricata's growth include significant community engagement. Additionally, later milestones like the release of Suricata 6.0 in 2020 and Suricata 7.0 in 2023. For more detail on important milestones of Suricata, see this Suricata Timeline

3.2 Suricata Architecture

Suricata is a high-performance, open-source IDS/IPS and network security monitoring engine. It leverages native multi-threading, efficient algorithms, and modular preprocessing to achieve real-time analysis on multi-core systems without manual traffic balancing.

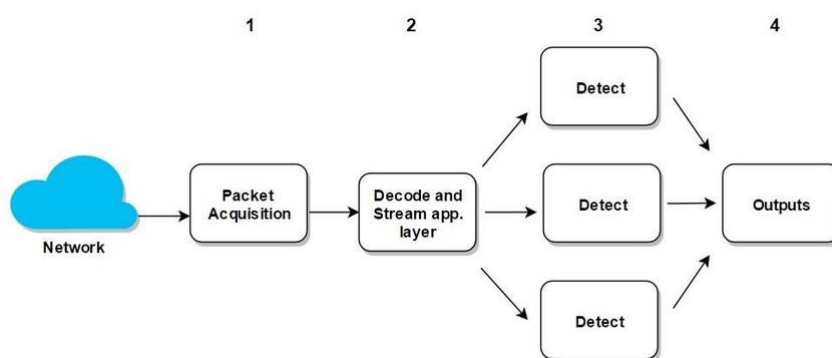
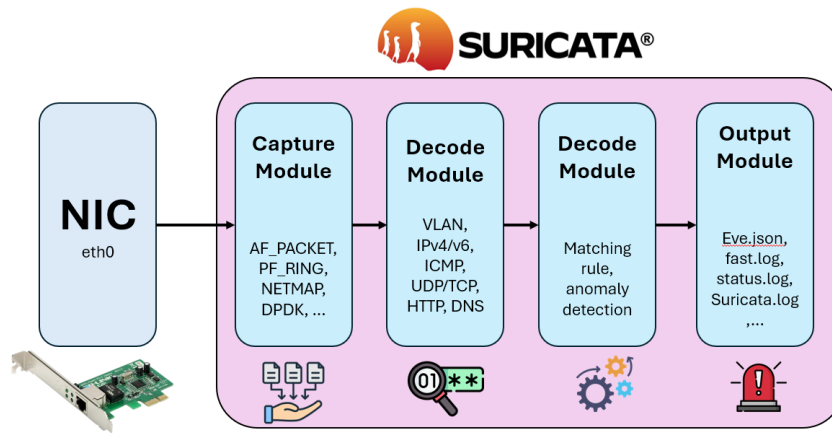


Figure 3.3: Suricata Architecture

Its core components—*packet capture*, *decoding*, *detection*, and *output*—work in concert to capture, normalize, inspect, and report on network traffic. We will cover each of these components in detail.



3.2.1 Packet Capture

Purpose: The first stage is acquiring raw packets from one or more interfaces.

Method: Suricata uses different capture mechanisms depending on the platform:

Platform	Capture Mechanism	Description	Use Case
Linux	AF_PACKET (default)	Employs Linux's AF_PACKET sockets for high-speed packet capture. Supports AF_PACKET v3 for zero-copy mode. Can use eBPF for load balancing.	Inline IPS, high-performance IDS
Linux (alternative)	PF_RING	High-speed packet capture framework from ntop, with kernel bypass. Needs PF_RING kernel module and compatible NICs.	Environments needing efficient packet capture with load balancing.
Linux / FreeBSD / macOS	Netmap	Kernel bypass framework for fast packet I/O. Requires NETMAP support in the OS and NIC drivers.	Specialized high-speed packet processing tasks.
Linux (5.3+)	AF_XDP	Leverages eXpress Data Path for ultra-low latency packet processing. Requires kernel 5.3+ and compatible NIC drivers.	High-performance scenarios requiring minimal latency.
Linux	NFQUEUE	Captures packets via Netfilter queue.	Inline IPS setups: Integrates with iptables/nftables; suitable for packet modification.
Linux	DPDK	High-performance packet processing using Data Plane Development Kit, bypassing kernel for low-latency networking. Requires dedicated CPU cores and NIC support.	High-speed packet processing, low-latency applications, network function virtualization (NFV).
Windows	WinDivert	A kernel-level driver that intercepts, filters, and can inject packets in Windows.	Inline IPS, traffic redirection, deep filtering. Requires manual installation and configuration.
Generic (cross-platform)	PCAP (libpcap) (default fallback)	Standard libpcap for compatibility across systems where other methods are unavailable.	Simple setups, offline analysis. Limited performance; not ideal for high-throughput environments.

Table 3.1: Packet Capture Mechanisms Across Different Platforms

Packet capture can be distributed across multiple threads to handle high-speed networks.

3.2.2 Decoding

Purpose: Each packet is processed through a series of decoders that extract information from the data link layer (L2) up to the application layer (L5) in TCP/IP model. This decoded information is stored in an internal packet representation and will be used later by the detection engine.

Method: Suricata processes packets through a modular pipeline of decoders, each tailored to a specific protocol or network layer. This ensures efficient and

accurate parsing, critical for handling encapsulated protocols (e.g., Ethernet → IPv4 → TCP). For each protocol, the packet structure is updated to point to the corresponding header or payload in the packet data. Depending on the decoded information, Suricata calls the next decoder to process the next layer of the packet until no more layers remain to decode (recursive decoding).

The table below summarizes key information extracted at each layer:

Layer	Information Extracted
L2 – Data Link	Ethernet frame, MAC addresses, VLAN tags
L3 – Network	IP addresses, header, fragmentation and reassembly
L4 – Transport	Ports, TCP/UDP flags, sequence numbers, checksum, TCP stream reassembly
L5 – Application	Automatically identifies protocols (based on behaviour, not just port since connection may not use standard port) and extracts metadata such as HTTP URIs, DNS query names, and TLS certificate information.

Table 3.2: Summary of decoding layers and extracted information

To view these plugins in Suricata, you can check the Suricata’s official Github repository, under `src/decode-*`

File	Description	Commit Date
decode-ethernet.h	src: make include guards more lib...	last year
decode-events.c	af-packet: add event for packets t...	last month
decode-events.h	af-packet: add event for packets t...	last month
decode-geneve.c	conf: prefix conf API with SC	3 weeks ago
decode-geneve.h	src: make include guards more lib...	last year
decode-gre.c	decode/gre: decode arp packets	11 months ago
decode-gre.h	src: make include guards more lib...	last year
decode-icmpv4.c	decode/icmpv4: rename ICMPV4...	last year
decode-icmpv4.h	decode/icmpv4: rename ICMPV4...	last year
decode-icmpv6.c	style: remove some useless return	11 months ago
decode-icmpv6.h	decode/icmpv6: store embedded...	last year
decode-ipv4.c	decode/tcp: add and use Packets...	last year
decode-ipv4.h	decode/ipv4: prep for turning ip4...	last year
decode-ipv6.c	style: remove some useless return	11 months ago
decode-ipv6.h	clean: remove unused struct defi...	11 months ago
decode-mpls.c	decode/mpls/tests: improve pkt ...	3 years ago
decode-mpls.h	src: make include guards more lib...	last year

Figure 3.4: Suricata source code files for decoders

For instance:

- `decode_ipv4.c` contains functions to parse IPv4 headers and extract fields

such as source IP, destination IP, protocol type, and header length.

- `decode_ethernet.c` handles Ethernet headers, extracting source and destination MAC addresses and the EtherType field.

The result of this decoding process is a rich *packet structure* that carries all relevant metadata and payload fragments to the next stage of Suricata's processing pipeline.

3.2.3 Detection

Suricata's detection engine is optimized for high-volume traffic, leveraging modern multi-core CPU architectures through native multi threading. Unlike other IDS/IPS systems requiring manual load balancing, Suricata automatically distributes workloads across threads, simplifying deployment in high-traffic environments. Packet capture, decoding, and detection are all multi-threaded, enabling horizontal scaling. Detection engine is the heart of every IDS/IPS system. This detection engine can work by signature-bases, or anomaly-based as mentioned in the first part of the report.

a, Signature-Based Detection

Purpose: Identifies threats by applying rules to decoded packets. **Method:** Using rules, compatible with Snort's format, include an action (e.g., pass, drop), header (e.g., protocol, source/destination IP, ports), and options (e.g., content matching, protocol-specific keywords). Example: `alert tcp any any -> 192.168.1.0/24 111 (content:"|00 01 86 a5|"; msg:"mountd access");`. This rule structure will be discussed in detail in the next section. Then, extracted packet information from decode stage is compared against rules using efficient algorithms like Aho-Corasick, Hyperspace, or PCRE for content matching.

b, Anomaly Detection

Update later.

3.2.4 Output

After detection, potential threats and related packets are forwarded to the logging/alert component, configurable via `suricata.yaml`. By default, Suricata outputs `fast.log` for concise overviews and `eve.json` for detailed JSON logs, ideal for SIEM integration. In IPS mode, Suricata can block malicious traffic with inline mode or firewall integration, requiring specific setup.

Additional, as an open-source tool, Suricata's architecture supports customization. Components can be modified, plugins adding, to meet your organization

specific needs.

3.3 Suricata Rule Structure

Suricata rules form the core of its detection capabilities. They specify which network traffic to inspect and what actions to take when specific patterns are matched. These rules are signature-based: each rule defines a signature, and when network traffic matches that signature, Suricata can generate alerts, log events, or even block the traffic (in IPS mode). Rules may be:

- Sourced from community-maintained sets such as Emerging Threats. These rulesets can be easily download, update and manage with `suricata-update` dependency.
- Custom-written by your own to meet specific organizational needs, making Suricata highly flexible and adaptable.

Suricata is also compatible with Snort rules, but with additionally offers extended keywords and capabilities. In this section , I cover briefly about Suricata rule structure, for detail information about this rule structure as well as rule options, please refer to the official Suricate document guide.

A typical Suricata rule consists of three main parts:

1. **Action:** Specifies what Suricata does when the rule matches.
2. **Header:** Defines the protocol, IP address, port and direction of the rule
3. **Options:** Provide detailed matching conditions and metadata.

The overall syntax is:

```
action protocol source_address source_port  
-> destination_address destination_port (options)
```

Example of a Suricata rule, this will also be a case study for some of our sections below:

```

Action           Option
alert http $HOME NET any -> $EXTERNAL NET any
(msg:"HTTP GET Request Containing Rule in URI";
flow:established,to_server; http.method;
content:"GET"; http.uri; content:"rule";
fast_pattern; classtype:bad-unknown; sid:123;
rev:1;)
Protocol

```

Figure 3.5: Example of a Suricata rule

In the following we explore each part in detail.

3.3.1 Actions

The `action` specifies Suricata's response when a rule matches (behavior also depend on whether Suricata is working in IDS or IPS mode). Available actions include:

Action	IDS Mode	IPS Mode
alert	Generate an alert; allow the packet.	Generate an alert; allow the packet.
pass	Stop further inspection; allow the packet.	Stop further inspection; allow the packet.
drop	(Treated as alert)	Drop the packet; generate an alert.
reject	(Treated as alert)	Drop the packet; send TCP RST or ICMP unreachable error to the sender.
reject_src	(Treated as alert)	Drop the packet; send TCP RST or ICMP to the sender.
reject_dst	(Treated as alert)	Drop the packet; send TCP RST or ICMP to the receiver of matching package.
reject_both	(Treated as alert)	Drop the packet; send TCP RST or ICMP to both sides off the conversation.

Table 3.3: Behavior of Suricata actions in IDS vs. IPS modes

IN IPS mode, using any of the `reject` actions also perform `drop`.

3.3.2 Header

The header begins with the protocol, followed by source and destination IPs and ports, and the direction operator.

Protocol	Source IP + Port	Direction	Destination IP + Port
http	\$HOME_NET any	->	\$EXTERNAL_NET any

Figure 3.6: Example of Suricata rule header

a, Protocol

This keyword will specify which network protocol Suricata need to concerns. Suricata supports four basic protocol:

- Network layer: `ip`, `icmp`
- Transport layer: `udp`, `tcp` (and some additional TCP related options like: `tcp-pkt` for matching content in individual tcp packages, `tcp-stream` for matching content only in a reassembled tcp stream)

Upon these basic protocol, there are some Layer 5 (Application Layer) protocols are also supported: `http` (either `http1` or `http2`), `ftp`, `tls` (this includes `ssl`), `smb`, `dns`, `dcerpc`, `dhcp`, `ssh`, `smtp`, `imap`, `pop3`, `modbus*`, `dnp3*`, `enip*`, `nfs`, `ike`, `krb5`, `bittorrent-dht`, `ntp`, `rfb`, `rdp`, `snmp`, `tftp`, `sip`, and `websocket`.

(Protocols with `*` are disable by default, require manual enable in the `suricata.yaml` config file.)

b, IP Addresses and Ports

IP address can be specified by single values, ranges, negations, or variables defined in the configuration.

Type	Example	Description
Single IP	191.168.1.8	Matches exactly with this address.
CIDR range	192.168.0.0/24	Matches all addresses in this /24 subnet.
Variable	\$HOME_NET	This variable is defined inside <code>suricata.yaml</code> .
Negation	!\$HOME_NET	Matches any address outside \$HOME_NET.
Grouping	[\$HOME_NET, 192.168.1.0/24]	Union of multiple networks.
Any	any	Matches all IP addresses.

Table 3.4: IP address specification

Suricata supports both **IPv4** and **IPv6**

Port specifications follow the same patterns:

Type	Example	Description
Single Port	80	Matches exactly this port
Any Port	any	Matches all ports (0-65535).
Port Range	100-102	Matches a continuous range of ports from start to end.
Negation	!80	Matches any port except the specified one.
Grouping Operator	[100:120, !101]	All port in range 100 to 102, except 101.

Table 3.5: Description of port matching types and examples

c, Direction

Suricata uses three different arrow operators to control which way traffic is evaluated:

- *Unidirectional*: "→"

Matches only packets flowing from the source to the destination as written. For example,

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (...)
```

will only inspect packets sent from \$EXTERNAL_NET toward \$HOME_NET on port 80.

- *Session-aware bidirectional*: "=>"

Behaves like `->` for initial packet direction, but allows matching on both sides of an established transaction when used with keywords that track session state (e.g. `flow:established;` or `flowbits`). In effect, you still write the rule in one direction, but Suricata will check both client-to-server and server-to-client payloads once the session is seen. Example:

```
alert http $HOME_NET any => $EXTERNAL_NET any
(flow:established; content:"password="; ...)
```

This will catch “password=” in either direction after the HTTP session is established.

- *Fully bidirectional:* “`<>`”

Instantly inspects packets in both directions without requiring session keywords. Useful when you want a single rule to match client-to-server or server-to-client traffic equally:

```
alert tcp $HOME_NET any <> $EXTERNAL_NET any
(content:"secret"; ...)
```

When to use each:

- `->` Simple, stateless patterns where only one direction matters.
- `=>` Stateful checks—e.g. look for data both ways once a session is set up.
- `<>` Stateless bidirectional matching, at the cost of slightly more CPU since both directions are always checked.

Note that there is **no** `<-` operator; use separate rules or variables if needed for reverse-only inspection.

3.3.3 Options

Options are enclosed in parentheses “`()`” after the header. Options contain key-value pair, separated by semicolons “`;`” like this:

```
(option1:value1; option2:value2; ...)
```

Options may be grouped into four categories:

- **Metadata:** Metadata or additional visualize when logging/alert, no effect on inspection of the package (e.g., `msg`)
- **Payload:** Content-based matching (e.g., `content`, `pcre`).

- **Non-Payload:** Inspect condition that not related to package payload, like fields in package header, flow attributes (e.g., `flow`, `ttl`, `dsize`).
- **Post-Detection:** Actions after a match (e.g., `logto`, `tag`).

a, Metadata Options

General options provide descriptive information and references without affecting the matching logic:

1. **msg** (Message)

Give the contextual information about the signatures and the alert. This is the message that will be displayed when the rule is triggered. It should be descriptive enough to understand what the rule is doing.

- **Syntax:** `msg:"text_here"`, but notice that escape characters `"`, `;`, and `\` in a text string must be escaped with a backslash `\`.
- **Example** with out case study rule:

```
(msg:"HTTP GET Request Containing Rule in URI";  
flow:established,to_server; http.method;  
content:"GET"; http.uri; content:"rule";  
fast_pattern; classtype:bad-unknown; sid:123;  
rev:1;)
```

2. **sid** (Signature ID)

A unique identifier for the rule. Must be unique across fore all rules within the same rule group id (`gid`).

- **Syntax:** `sid:<number>`
- **Example** with out case study rule:

```
alert http $HOME_NET any -> $EXTERNAL_NET any  
(msg:"HTTP GET Request Containing Rule in URI";  
flow:established,to_server; http.method;  
content:"GET"; http.uri; content:"rule";  
fast_pattern; classtype:bad-unknown; sid:123;  
rev:1;)
```

3. **rev** (Revision)

Each sid can have multiple revisions. This is used to track changes to the rule over time. Whenever a rule is updated, the rev number should be incremented by the rule writer.

- **Syntax:** `rev:<number>`

- **Example** with out case study rule:

```
alert http $HOME_NET any -> $EXTERNAL_NET any
(msg:"HTTP GET Request Containing Rule in URI";
flow:established,to_server; http.method;
content:"GET"; http.uri; content:"rule";
fast_pattern; classtype:bad-unknown; sid:123;
rev:1;)
```

4. **gid** (Group ID)

Use to organize rules into separate groups. Within the same group, each rule must have a unique Signature ID (sid). However, the same sid can appear in different groups without conflict. For example, gid:1 and gid:2 can both contain a rule with sid:1000001, and these rules are considered distinct. By default, Suricata assigns a gid of 1 to all rules. This grouping has no technical impact on detection—it is primarily used for identification and logging purposes, such as in the fast.log output.

- **Syntax:** `gid:<number>`

- **Example** in the output of **fast.log** file:

```
07/12/2022-21:59:26.713297 [**] [1:123:1] HTTP GET Request Containing Rule in
URI [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP}
192.168.225.121:12407 -> 172.16.105.84:80
```

Look at this output part: '[1:123:1]', the first component is gid=1, the second is sid=123 and the last one is rev=1.

5. **classtype** (Classification Type)

Provide the classification of the alert. This is used to categorize the rules and the alert.

- **Syntax:** `classtype:<type>;`

- **Example:** `classtype:attempted-admin;`

When writing rules, if we assign it to a classification type using `classtype`, we must define that classification type first inside the `/var/lib/suricata/rules/classification.config` file. You can open this file to see the list of available classification types.

```
chutrunanh@chutrunanh:~$ sudo su
[sudo] password for chutrunanh:
root@chutrunanh:/home/chutrunanh# cd /var/lib/suricata/rules/
root@chutrunanh:/var/lib/suricata/rules# ls
classification.config  suricata.rules  test.rules
root@chutrunanh:/var/lib/suricata/rules# cat classification.config
config classification: not-suspicious,Not Suspicious Traffic,3
config classification: unknown,Unknown Traffic,3
config classification: bad-unknown,Potentially Bad Traffic, 2
config classification: attempted-recon,Attempted Information Leak,2
config classification: successful-recon-limited,Information Leak,2
config classification: successful-recon-largescale,Large Scale Information Leak,2
config classification: attempted-dos,Attempted Denial of Service,2
config classification: successful-dos,Denial of Service,2
config classification: attempted-user,Attempted User Privilege Gain,1
config classification: unsuccessful-user,Unsuccessful User Privilege Gain,1
config classification: successful-user,Successful User Privilege Gain,1
```

Figure 3.7: Class types inside config file

Each classification type follow this format:

```
config classification: <short_name>,<full_name>,<priority
_level>
```

Example: `config classification: attempted-admin,Attempted Administrator Privilege Gain,1`

Once we have defined the classification in the configuration file, we can use the `classtype` in our rules. A rule with `classtype: attempted-admin` will be assigned a priority of 1 and the alert will contain Attempted Administrator Privilege Gain in the Suricata logs.

6. priority

Use to set the priority of the rule. The higher the number, the higher the priority. Highest is 1 and lowest is 255. Signature with higher priority will be processed first. Normally, if a rule already using `classtype`, then the priority filed is already included in the `classtype` definition, so you do not need to add `priority` in the rule unless you want to override the priority of the `classtype`.

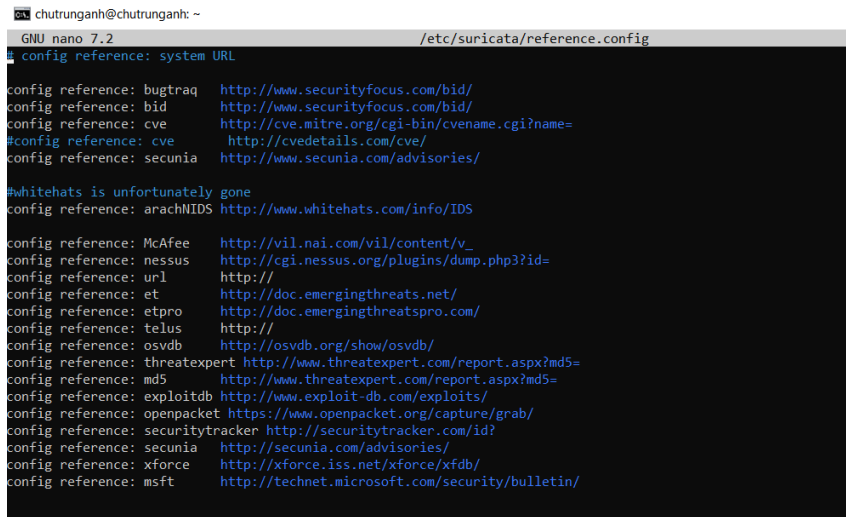
- **Syntax:** `priority:<number>`
- **Example:** `priority: 1`

7. reference

Provides a reference to an external source or document related to the rule that

can provide more information about the rule. This `reference` keyword can appear multiple times in a rule.

- **Syntax:** `reference:<type>`, `<reference_source with type keyword can be url, cve, etc.` See all reference types that are defined in your Suricata by checking the `reference.config` file under `/etc/suricata/`:



```
chutrunganh@chutrunganh: ~
GNU nano 7.2 /etc/suricata/reference.config
# config reference: system URL

config reference: bugtraq http://www.securityfocus.com/bid/
config reference: bid http://www.securityfocus.com/bid/
config reference: cve http://cve.mitre.org/cgi-bin/cvename.cgi?name=
#config reference: cve http://cvedetails.com/cve/
config reference: secunia http://www.secunia.com/advisories/

#Whitehats is unfortunately gone
config reference: arachnIDS http://www.whitehats.com/info/IDS

config reference: McAfee http://vil.nai.com/vil/content/v_
config reference: nessus http://cgi.nessus.org/plugins/dump.php?id=
config reference: url http://
config reference: et http://doc.emergingthreats.net/
config reference: etpro http://doc.emergingthreatspro.com/
config reference: telus http://
config reference: osvdb http://osvdb.org/show/osvdb/
config reference: threatexpert http://www.threatexpert.com/report.aspx?md5=
config reference: md5 http://www.threatexpert.com/report.aspx?md5=
config reference: exploitdb http://www.exploit-db.com/exploits/
config reference: openpacket https://www.openpacket.org/capture/grab/
config reference: securitytracker http://securitytracker.com/id?
config reference: secunia http://secunia.com/advisories/
config reference: xforce http://xforce.iss.net/xforce/xfdb/
config reference: msft http://technet.microsoft.com/security/bulletin/
```

Figure 3.8: Reference types defined inside Suricata config file

- **Example:** `reference:url,https://www.example.com/rule-info` or `reference:cve,CVE-2025-24985`

8. metadata

Use to add additional information about the rule in free-form, but usually in key-value pair since Suricata can include these in the eve formatted log, which is in JSON format. This is useful for adding information such as the author of the rule, the date the rule was created, or any other information that may be relevant to the rule

- **Syntax:** `metadata: <key 1> <value 1>, <key 2> <value 2>`
- **Example:** `metadata: author CTA, created_at 2025_04_21`

9. target

Used by rule writer to indicate which side of the communication is the **target (victim)** of the attack. By default, Suricata assumes the destination IP is the target, but in some cases (like data exfiltration), the **source IP** may actually be the intended target. This keyword improves the accuracy of the generated

alert by explicitly marking the target side. The resulting event in JSON (EVE format) will include a separate field for this.

- **Syntax:** `target: [src_ip | dest_ip]`. If thr
- **Example:** `target: dest_ip` indicate that the destination IP address machine is the target of the attack

10. requires

Allows a rule to specify that certain Suricata features must be enabled or a certain Suricata version must be used. If the environment does not meet these requirements, the rule will be ignored. This is useful for rules that require specific features/ version to be enabled in order to work properly. For example, if a rule requires the `http_inspect` preprocessor to be enabled, you can use this option to specify that requirement.

- **Syntax:** `required: feature <feature_name>, version <version_condition>`
- **Example:** `required: feature geoip, version >= 7.0.4 < 8`. This means the rule will only be loaded if the `geoip` feature is enabled and the Suricata version is at least 7.0.4 but less than 8.

b, Payload Options

Inspect the payload content of a packet or a stream.

1. content (Content Match)

If no sub option is added, try to find a match in all bytes of the payload

- **Syntax:** `content: "<string>"`; this string can be in hex format or ASCII format. If you want to use the hex format, place it between `| |`
- **Example:** `content: "A|3A|b|"` which represents the string `A:b` literally.

Some sub-options that are used in conjunction with `content` option:

- **replace:** The this options can only be used in IPS, to modify the packet payload. Example usage: `content: "abc"; replace: "def"`. This will changes the content matching part found in packet payload from text `('abc')` to `('def')`. This is useful for replacing sensitive information in network traffic.
- **nocase:** to make the match case insensitive. Example: `content: "A|3A|b"; nocase;` can match with `A:b`, `a:b`, `A:B`, `a:B`.

- **depth**: to limit the search to the first n bytes of the payload. Example usage: `content:"A|3A|b"; depth: 3`; will only search the first 3 bytes of the payload. Example string `A:bcdefghik` in payload will match since three first bytes `A:b` is same as the content.
- **offset**: to skip the first n bytes of the payload. Example: `content:"A|3A|b"; offset: 2`; will skip the first 2 bytes of the payload. Example string `CdA:b` in payload will match since two first bytes `Cd` are ignored.
- **distance**: to fine-tune how close to search for the next content match relative to a previous one. It's super useful for detecting patterns that occur at a specific offset after a previous match. Syntax:

```
content:"first_pattern";
content:"second_pattern"; distance:<number>;
```

Example in a rule:

```
alert tcp any any -> any 80 (
    msg:"Suspicious HTTP request with specific pattern";
    content:"GET"; http_method;
    content:"/admin"; distance:5; http_uri;
    sid:1000001; rev:1;
)
```

First, it looks for "GET" in the HTTP method. Then, starts scanning 5 bytes after the end of "GET" for "/admin" text in the URI. Here is a visualization:

```
[---"GET"---]--(5 bytes gap) --[---"/admin"---]
^ First match                ^ Start second match here
```

In case `/admin` appears too soon (less than 5 Bytes gap, then it will not match. So `distance` keyword limits the "closest" gap allow to start searching for the next content match relative to a previous one. We can combine with **within** keyword to limit the "farthest" gap that still accept searching for the next content match relative to a previous one. This helps create a range of bytes to search for the next content match. Example:

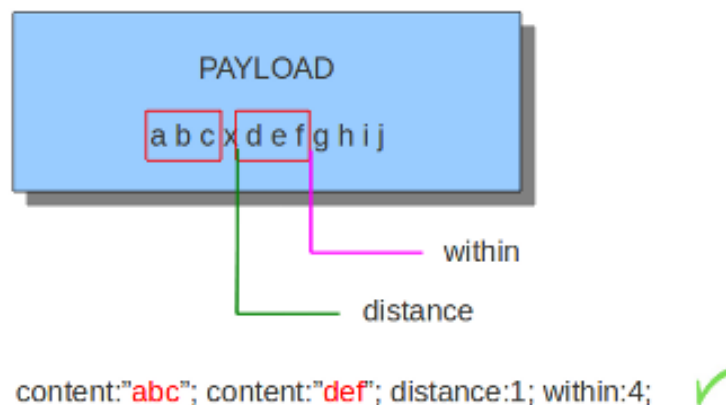


Figure 3.9: This rule will only match if the string 'def' appears in a gap between 1 to 4 Bytes counting from the end of the string 'abc'.

- **isdataat** : Verifies whether there is still a specified number of bytes existing at or after a certain offset (absolute or relative). Syntax: `isdataat:<number>`. With the `relative` option is used to know if there is still data at a specific part of the payload relative to the last match. Examples:

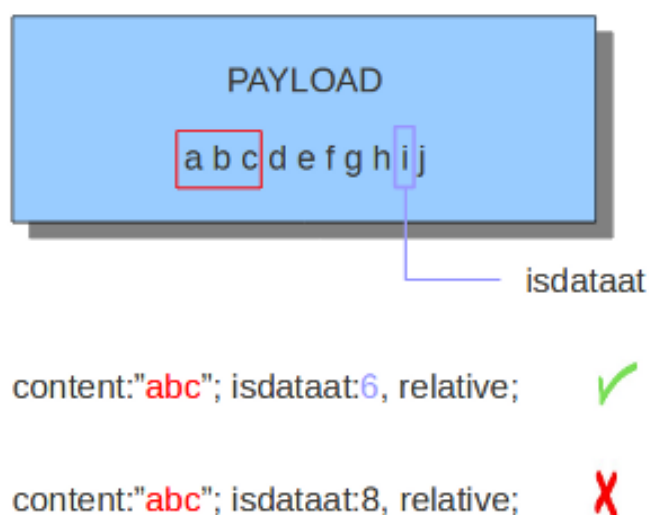


Figure 3.10: Start from the last mach, letter "d" in this case, count till the end of payload, there are only 7 bytes left.

2. **dsize**

Checks the payload size of the packet. It's useful for detecting unusually sizes of payloads, like in detecting buffer overflows or other attacks that rely on large payloads.

- **Syntax:**

```
dsize:<comparison><value>; or dsize:min<>max;
```

- **Example:**

```
alert tcp any any -> any any (msg:"dsize more than  
or equal value"; dsize:>=10; sid:3; rev:1;)
```

This rule will be triggered if the payload size of package is more than 10 bytes. Another example:

```
alert tcp any any -> any any (msg:"dsize range value";  
dsize:8<>20; sid:6; rev:1;)
```

Triggers if the payload size is outside range 8 to 20 bytes.

3. **entropy**

Calculate the Shannon entropy value of the content and compare it with a given value. This is useful for detecting encrypted or compressed data, which often has high entropy. The entropy value is calculated based on the randomness of the payload. High entropy values indicate that the data is more random and less predictable, which can be a sign of encryption or obfuscation.

- **Syntax:**

```
entropy:<operator><value>
```

Range: 0.0 (very predictable) to 8.0 (very random).

- **Example usage:**

```
alert http any any -> any any (msg:"entropy simple  
test"; file.data; entropy: value >= 7.5; sid:1;)
```

This example matches if the file.data content for an HTTP transaction has a Shannon entropy value higher than 7.5 (possibly encrypted or obfus-

cated).

4. **pcre** (Perl Compatible Regular Expressions)

Allows matching of patterns using Perl-compatible regular expressions (PCRE). Because PCRE can be performance-intensive, it is typically used together with `content|`. In such cases, the `content` must match first, and only then is the PCRE evaluated — reducing unnecessary processing.

- **Syntax:**

```
pcre: "<regex>/options";
```

`options` can be: `i` for case insensitive, `s` for not checking newline characters, `E` for ignoring the newline characters at the of the buffer/payload, `R` for matching relative to the last pattern match (similar to distance 0), etc

- **Example:** In this example following:

```
pcre: "/[0-9]{6}/";
```

There will be a match if the payload contains six numbers

There are still some more options keyword to work with packet payload: `startswith`, `endswith`, `absent`, `bsize`, `byte_test`, `byte_math`, `byte_jump`, `byte_extract`, `rpc` that have not been covered in this report. For detailed explanations of these keywords, refer to the Suricata payload keywords.

c, Non-payload Options

CHAPTER 4. LAB ENVIRONMENT SETUP

4.1 Application Building

4.1.1 Libraries and Tools

All software with detail version I used in this project to set up lab environment.

Purpose	Tool	URL Address
Platform to run virtual machines	VMware Workstation Pro 17	https://www.vmware.com/products/desktop-hypervisor/workstation-and-fusion
Virtual Machine to simulate network	PnetLab 4.2.10 (.ova image file format)	https://pnetlab.com/pages/main
Dedicated server for running IDS	Ubuntu Server 24.04.2 (.iso image file format)	https://ubuntu.com/download/server
Act as an attack machine inside LAN network	Kali Linux 2025.1a (.qcow2 image file format)	https://www.kali.org/get-kali/kali-virtual-machines
Simulate Cisco router, switch devices	Cisco image files (.bin image file format)	See in attached folder to this project

Table 4.1: List of libraries and tools used

4.1.2 Achievement

Update Later

4.1.3 Illustration of main functions

Update Later

4.2 Testing

Update Later

4.3 Deployment

Updata Later

CHAPTER 5. CONCLUSION AND FUTURE WORK

5.1 Conclusion

5.2 Future Work

REFERENCE

- [1] N. A. Tú, “Xây dựng hệ thống giám sát mạng dành cho bệnh viện đa khoa cấp tỉnh với mã nguồn mở,” Accessed: 2025-04-05, Ph.D. dissertation, Posts and Telecommunications Institute of Technology, VN, 2022. [Online]. Available: https://ptithcm.edu.vn/wp-content/uploads/2023/07/2020_HTTT_NguyenAnhTu_LV.pdf.
- [2] H. Hindy, “Intrusion detection systems using machine learning and deep learning techniques,” Ph.D. dissertation, School of Design and Informatics Abertay University, UK, 2021.
- [3] N. K. Văn, *Giao trình cơ sở an toàn thông tin*. Hanoi University of Science and Technology, 2023.
- [4] Ron Shamon, ClearNetwork, *Top 10 intrusion detection and prevention systems*. [Online]. Available: <https://www.clearnetwork.com/top-intrusion-detection-and-prevention-systems/> (visited on 03/31/2025).