Intrusion Detection System

System Description

Abstract

This is the template for System of Systems Description (SoSD document) according to the Eclipse Arrowhead documentation structure.

Contents

1 Overview **3**
1.1 Significant Prior Art 4
1.2 How This SoS Is Meant to Be Used 4
1.3 SoS functionalities and properties 4
1.4 Important Delimitations 5

2 Services **6**
2.1 Produced services 6
2.2 Consumed services 6

3 Security **7**
3.1 Overview of Security Level 7
3.2 Security Model 7

4 References **7**

5 Revision History **8**
5.1 Amendments 8
5.2 Quality Assurance 8

1 Overview

This document describes the Intrusion Detection System system of systems (SoS), which provides security with the detection of patterns/signatures characteristics of known attacks or suspicious patterns deviating from the ”normal” behavior recreated by an artificial intelligence model. These patterns are detected on the IP packets, of which headers also provides the IP addresses in order to ban connections with the suspicious addresses. The data used to train the models are stored in temporary buffers.

The rest of this document is organized as follows. In Section 1.1, we reference major prior art capabilities of the SoS. In Section 1.2, we describe the intended usage of the SoS. In Section 1.3, we describe fundamental properties provided by the SoS. In Section 1.4, we describe delimitations of capabilities of the SoS. In Section 2, we describe the microsystem (abstract level with references to their SysDs) which constitutes the SoS. In Section 3, we describe the security capabilities of the SoS.

1.1 Significant Prior Art

The Intrusion Detection System (IDS) based on artificial intelligence on edge is built on previous works in the areas of network security, artificial intelligence and edge computing.

- **Traditional IDS:** These systems, such as Snort or Suricata, have established the bases to detect patterns of malicious traffic. A way to detect a threat is to observe network traffic and analyze the packets in circulation. By comparing these exchanges with a database, it is possible to identify recurring patterns that herald attacks. In this way, an alert can be raised, and human intelligence can act accordingly.
- **AI in network security:** Since knowledge of the attack is required to enter it into the database, the detection of new threats seems complicated. Machine learning will enable new threats to be detected more effectively. Their implementation uses programming languages such as Python, machine learning libraries such as Scikit-learn and various frameworks.
- **Edge computing:** Edge computing enables data processing close to the source of generation, reducing latency and the bandwidth required to transmit data to the cloud.

1.2 How This SoS Is Meant to Be Used

The artificial intelligence-based intrusion detection System of Systems (SoS) is intended for use in environments where network security is critical, such as critical infrastructure, industrial networks, and IoT systems. The system uses artificial intelligence algorithms to analyze network traffic in real time and detect anomalous patterns that may indicate the presence of security threats quicker and more efficiently. The streams issued from the different sensors are analyzed as IP packets headers

1.2.1 SysML/UML block diagram

1.3 SoS functionalities and properties

1.3.1 Functional properties of the SoS

- **Real-Time Detection:** The SoS is capable of analyzing network traffic in real time, using artificial intelligence algorithms to identify anomalous patterns that may indicate the presence of security threats.
- **Edge Processing:** Processing capability at the edge enables rapid response to detected threats, reducing latency and the bandwidth required to transmit data to the cloud.
- **Adaptability:** The SoS can adapt to different network environments and device types, adjusting its detection algorithms as needed to maximize effectiveness.
- **Scalability:** The system is designed to scale horizontally, allowing the addition of more processing nodes at the edge as needed to handle higher volumes of network traffic. Also thanks to the Service Registry and the Orchestration, a device can be easily added and communicate.

1.3.2 Configuration of SoS properties

- **Dynamic Configuration:** System parameters, such as detection thresholds and response policies, can be dynamically configured to adapt to the changing needs of the network.
- **Management Interface:** The SoS includes a management interface that allows network administrators to monitor system status, review security alerts and adjust settings as needed.

1.3.3 Data stored by the individual microsystem

The data, the packets preprocessed, are stored on servers located at the periphery of the network. When the models have been trained, only the weights of the model are stored and new data replaces the dataset.

1.3.4 Non-functional properties

- **Security:** The SoS implements robust security measures to protect both stored data and communications between processing nodes.
- **Reliability:** The system is designed to be highly reliable, with redundancy and failover mechanisms to ensure continuous availability.
- **Energy Efficiency:** The processing nodes on the edge are optimized to consume as little energy as possible, contributing to the sustainability of the system. However, to guarantee computing power, they should not be designed on batteries.
- **Low Latency:** Thanks to edge processing, the SoS can detect and respond to threats with minimal latency, improving network protection.

1.3.5 Stateful or stateless

- **Stateful:** SoS maintains the state of network connections and traffic patterns over time, enabling more accurate detection of persistent and advanced threats.

1.4 Important Delimitations

- **Universal Compatibility:** The SoS may not be compatible with all device types and network protocols. Integration with certain systems may require customization and specific settings.
- **Computing power:** With edge devices, the size of the models must be reduced and is limited to remain plugged directly to a power source.
- **Computing power:** With edge devices, the size of the models must be reduced and is limited to remain plugged directly to a power source when a local inference is done.

2 Services

2.1 Produced services

The produced services are the sendWarning when an intrusion is detected, the reportFile, a report on the network status and warnings sent periodically or on command.

2.2 Consumed services

The consumed services are getData, the network packets are analysed in real time, and authentication and authorisation services in order to mantain the security and validate users and devices authorised.

3 Security

3.1 Overview of Security Level

- **Startup Modes:** The system can be started in Arrowhead secure mode, which implements all recommended security measures, or in non-secure mode, which can be used for test environments or when security is not a critical concern.
- **X.509 Certificate Handling:** The system handles both Arrowhead-compliant and non-compliant X.509 certificates. This ensures that the system can integrate with a wide variety of devices and services while maintaining a high level of security. Handling the X.509 certificate and a token may be essential to assure the integrity of the system. Being an IDS working in real time, this could affect the performance, but it is safer.

3.2 Security Model

- **Supported Protocols:** The system supports secure protocols such as HTTPS and MQTT with TLS to secure communication between system components.
- **Data Protection:** Data in transit and at rest are encrypted using strong encryption algorithms such as AES-256 to protect data confidentiality and integrity. Also, using a local inference and data not being centralized, this will reduce the amount of data that needs to be transferred over the network.
- **System Authentication Capability:** The system implements strong authentication mechanisms, including certificate-based authentication and multifactor authentication, to ensure that only authorized users and devices can access the system.
- **Authorization Verification of Produced Services:** Each service produced by the system verifies the requestor's authorization before providing access, ensuring that only authorized users and systems can use the services.
- **Monitoring and Auditing:** The system includes monitoring and auditing capabilities to record and analyze security events, enabling rapid detection and response to security incidents.

4 References

5 Revision History

5.1 Amendments

Revision history and Quality assurance as per examples below

No.	Date	Version	Subject of Amendments	Author
1	2023-08- 10	X.Y.Z		Jerker Delsing
2				
3				

5.2 Quality Assurance

No.	Date	Version	Approved by
1	2022-01-10	X.Y.Z	

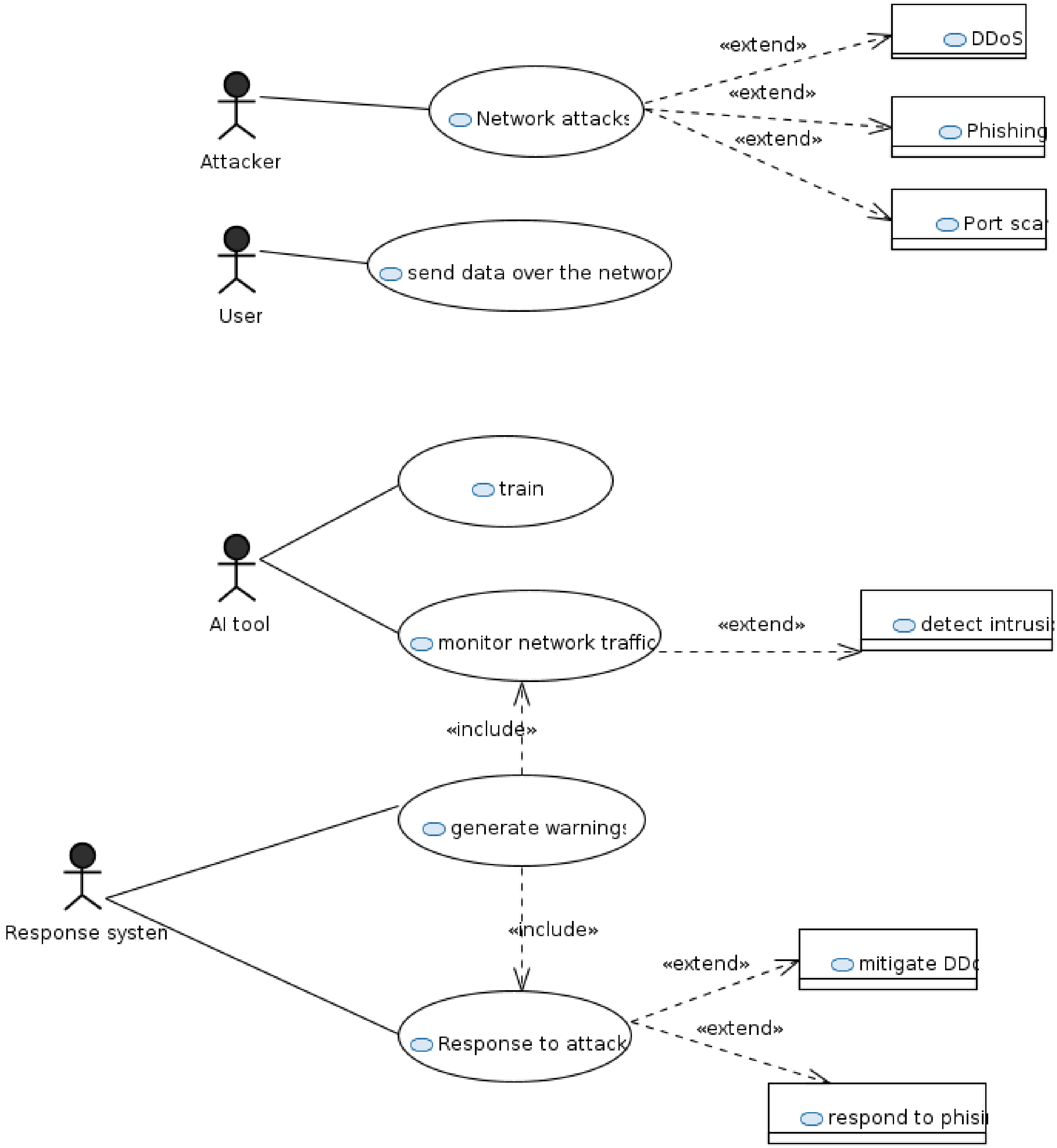

Figure 1: IDS use case

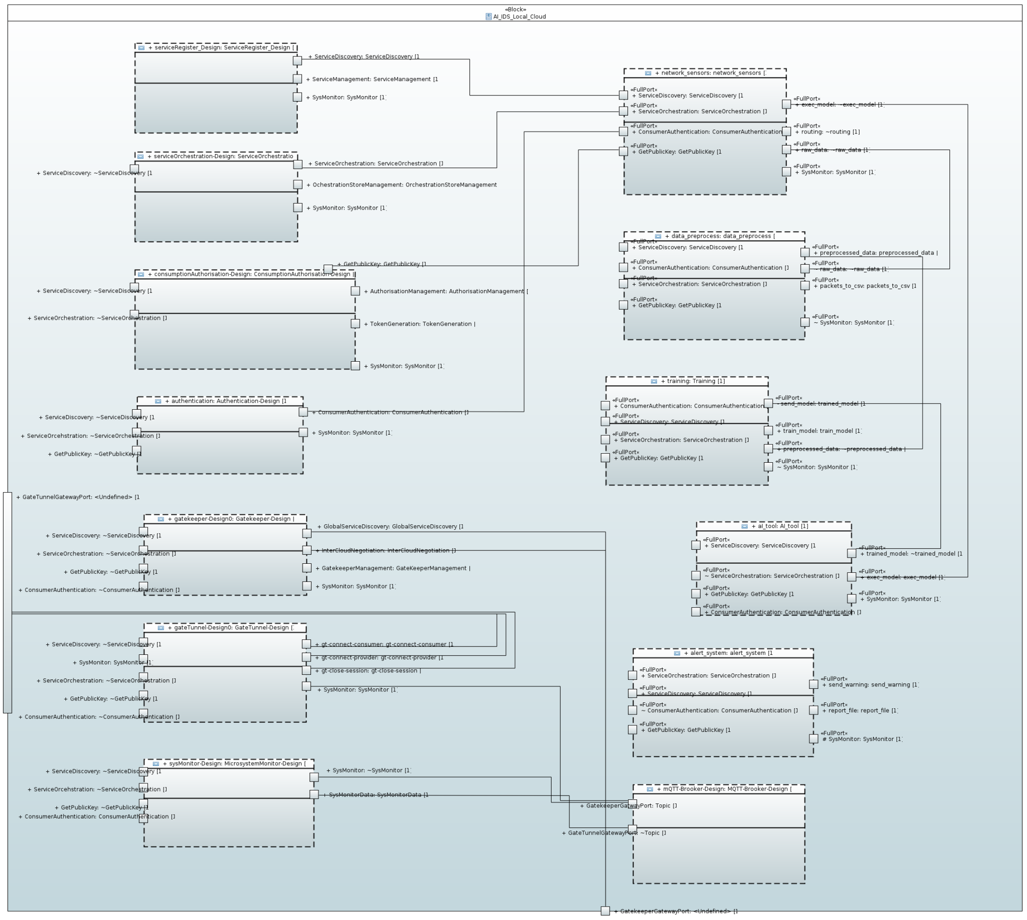
«Block»
AI_IDS_Local_Cloud
+ serviceRegister_Design: ServiceRegister_Design [
+ ServiceDiscovery: ServiceDiscovery [1
+ ServiceManagement: ServiceManagement [1
+ SysMonitor: SysMonitor [1]
+ serviceOrchestration-Design: ServiceOrchestratio
+ ServiceDiscovery: ~ServiceDiscovery [1
+ ServiceOrchestration: ServiceOrchestration [1]
+ OchestrationStoreManagement: OrchestrationStoreManagement
+ SysMonitor: SysMonitor [1]
+ GetPublicKey: GetPublicKey [1
+ consumptionAuthorisation-Design: ConsumptionAuthorisation-Design [
+ ServiceDiscovery: ~ServiceDiscovery [1
+ ServiceOrchestration: ~ServiceOrchestration [1]
+ AuthorisationManagement: AuthorisationManagement [
+ TokenGeneration: TokenGeneration |
+ SysMonitor: SysMonitor [1]
+ authentication: Authentication-Design [1
+ ServiceDiscovery: ~ServiceDiscovery [1
+ ServiceOrcehstration: ~ServiceOrchestration [1]
+ GetPublicKey: ~GetPublicKey [1
+ ConsumerAuthentication: ConsumerAuthentication [1]
+ SysMonitor: SysMonitor [1]
+ GateTunnelGatewayPort: <Undefined> [1
+ gatekeeper-Design0: Gatekeeper-Design |
+ ServiceDiscovery: ~ServiceDiscovery [1
+ ServiceOrchestration: ~ServiceOrchestration [1]
+ GetPublicKey: ~GetPublicKey [1
+ ConsumerAuthentication: ~ConsumerAuthentication [1]
+ GlobalServiceDiscovery: GlobalServiceDiscovery [1
+ InterCloudNegotiation: InterCloudNegotiation [1]
+ GatekeeperManagement: GateKeeperManagement |
+ SysMonitor: SysMonitor [1]
+ gateTunnel-Design0: GateTunnel-Design [1]
+ ServiceDiscovery: ~ServiceDiscovery [1
+ SysMonitor: SysMonitor [1]
+ ServiceOrchestration: ~ServiceOrchestration [1]
+ GetPublicKey: ~GetPublicKey [1
+ ConsumerAuthentication: ~ConsumerAuthentication [1]
+ gt-connect-consumer: gt-connect-consumer [1
+ gt-connect-provider: gt-connect-provider [1
+ gt-close-session: gt-close-session |
+ SysMonitor: SysMonitor [1]
+ sysMonitor-Design: MicrosystemMonitor-Design [
+ ServiceDiscovery: ~ServiceDiscovery [1
+ ServiceOrcehstration: ~ServiceOrchestration [1]
+ GetPublicKey: ~GetPublicKey [1
+ ConsumerAuthentication: ConsumerAuthentication [1]
+ SysMonitor: ~SysMonitor [1]
+ SysMonitorData: SysMonitorData [1
+ network_sensors: network_sensors [
«FullPort» + ServiceDiscovery: ServiceDiscovery [1
«FullPort» + ServiceOrchestration: ServiceOrchestration [1]
«FullPort» + ConsumerAuthentication: ConsumerAuthentication
«FullPort» + GetPublicKey: GetPublicKey [1
«FullPort» + exec_model: ~exec_model [1]
«FullPort» + routing: ~routing [1]
«FullPort» + raw_data: ~raw_data [1]
«FullPort» + SysMonitor: SysMonitor [1]
+ data_preprocess: data_preprocess [
«FullPort» + ServiceDiscovery: ServiceDiscovery [1
«FullPort» + ConsumerAuthentication: ConsumerAuthentication [1]
«FullPort» + ServiceOrchestration: ServiceOrchestration [1]
«FullPort» + GetPublicKey: GetPublicKey [1
«FullPort» + preprocessed_data: preprocessed_data |
«FullPort» ~ raw_data: ~raw_data [1]
«FullPort» + packets_to_csv: packets_to_csv [1
«FullPort» ~ SysMonitor: SysMonitor [1]
+ training: Training [1]
«FullPort» + ConsumerAuthentication: ConsumerAuthentication
«FullPort» + ServiceDiscovery: ServiceDiscovery [1]
«FullPort» + ServiceOrchestration: ServiceOrchestration [1]
«FullPort» + GetPublicKey: GetPublicKey [1
«FullPort» - send_model: trained_model [1
«FullPort» + train_model: train_model [1]
«FullPort» + preprocessed_data: ~preprocessed_data |
«FullPort» ~ SysMonitor: SysMonitor [1]
+ ai_tool: AI_tool [1]
«FullPort» + ServiceDiscovery: ServiceDiscovery [1
«FullPort» ~ ServiceOrchestration: ServiceOrchestration [1]
«FullPort» + GetPublicKey: GetPublicKey [1
«FullPort» + ConsumerAuthentication: ConsumerAuthentication [1]
«FullPort» + trained_model: ~trained_model [1
«FullPort» + exec_model: exec_model [1]
«FullPort» + SysMonitor: SysMonitor [1]
+ alert_system: alert_system [1
«FullPort» + ServiceOrchestration: ServiceOrchestration [1]
«FullPort» + ServiceDiscovery: ServiceDiscovery [1]
«FullPort» ~ ConsumerAuthentication: ConsumerAuthentication [1]
«FullPort» + GetPublicKey: GetPublicKey [1
«FullPort» + send_warning: send_warning [1]
«FullPort» + report_file: report_file [1]
«FullPort» # SysMonitor: SysMonitor [1]
+ mQTT-Brooker-Design: MQTT-Brooker-Design [
+ GatekeeperGatwayPort: Topic [1]
+ GateTunnelGatewayPort: ~Topic [1]
+ GatekeeperGatewayPort: <Undefined> [1

Figure 2: SoS block diagram