PROJECT

Multi-tenant cybersecurity operations centre designed as an open cloud service with machine learning elements

V16 Functional PoC

KPB5 – Implementation of the solution

type – documentation

month of submission – M22

Funded by the European Union
NextGenerationEU

RECOVERY AND RESILIENCE PLAN

GOVERNMENT OFFICE OF THE SLOVAK REPUBLIC

Contents

1 Introduction 4
1.1 Purpose of the document 4
1.2 Starting points and inputs to the PoC 4
1.3 Scope of the PoC 4
1.4 PoC operating environment 4
1.5 Limitations of the PoC 5
2 Components implemented in the INNOSOC PoC 6
2.1 SIEM – ELK Stack 6
2.1.1 ELK Stack deployment model 6
2.1.2 Elasticsearch deployment 6
2.1.3 Logstash deployment 8
2.2 Shuffle 17
2.2.1 Shuffle deployment model 17
2.2.2 Basic integration links of the Shuffle component in the PoC 18
2.3 Wazuh 19
2.3.1 Deployment model of the Wazuh component 19
2.3.2 Environments monitored by Wazuh 19
2.3.3 Deployed and activated modules of the Wazuh system 22
2.3.4 Basic integration links – sending notifications 23
2.4 Greenbone Community edition 30
2.4.1 Role of Greenbone Community Edition in the PoC 30
2.4.2 Deployment and basic configuration of Greenbone 30
2.4.3 Use of Greenbone within the PoC 30
2.4.4 Current status and limitations of Greenbone in the SOC 33
2.5 OpenCTI 34
2.5.1 Deployment of OpenCTI in the OpenStack environment 34
2.5.2 Basic integration links for the OpenCTI component 35
2.5.3 Note on the MISP platform 36
2.6 GLPI 36
2.6.1 Role of GLPI in the PoC 36
2.6.2 Deployment and basic configuration of GLPI 36
2.6.3 Use of GLPI within the PoC 37
2.6.4 Current status and limitations of GLPI in the SoC 40
2.7 NetBox 41
2.7.1 Role of NetBox in the PoC 41
2.7.2 Deployment and basic configuration of NetBox 41
2.7.3 Use of NetBox within the PoC 43
2.7.4 Current status and limitations of NetBox in the PoC 47
2.8 DFIR-IRIS 47
2.8.1 Deployment model of DFIR IRIS in the OpenStack environment 47

2.8.2 Basic integration links of the DFIR-IRIS component in INNOSOC 48
2.9 Kibana (ELK) 49
2.9.1 Dashboards created 49
2.9.2 Technical parameters and functionality verification 52
2.9.3 Use of Kibana within the PoC 53
2.10 Wiki.js 53
2.10.1 Role of Wiki.js in the PoC 53
2.10.2 Deployment and basic configuration of Wiki.js 54
2.10.3 Use of Wiki.js within the PoC 54
2.10.4 Current status and limitations of Wiki.js in the SOC 58
3 Conclusion 60
3.1 Summary of activities carried out 60
3.2 Nature and scope of the PoC 61
3.3 Verification and testing 61
3.4 Final evaluation 61

1 INTRODUCTION

1.1 Purpose of the document

The purpose of this document is to document the functional PoC within work package KPB5 – Implementation of the solution of the INNOSOC project. The document follows on from the project's previous architectural and implementation outputs and summarizes the state in which the proposed solution is implemented as a functional experimental concept.

The document captures the functional state of the implemented solution as an integrated whole, i.e. as a multi-tenant SOC-type cloud service that is deployed in experimental mode and ready for formal verification. Its main aim is to demonstrate that the proposed solution has been implemented in a functional and operable form within the scope of the PoC, and it constitutes the technical and operational documentation of the experimentally created solution.

1.2 Starting points and inputs to the PoC

The PoC builds on the results of the previous work packages and on the follow-up outputs of the project. The basic framework for implementation was created in KPB4 – Architecture of the solution, the aim of which was to create a coherent technological concept on the basis of the previous analytical and design outputs. Within KPB4, the functional and non-functional requirements, the business architecture, the technological architecture and the final DNR – Detailed design of the solution were defined, which represent a direct input to the KPB5 implementation stage.

The results of the project's previous design and analytical activities must also be regarded as substantive inputs to the implementation of the PoC, in particular the design of the virtualization solution, the design of the SOCaaS service and the design of the integration architecture of the individual components. The project plan lists among the main implementation tasks the configuration of the infrastructure, the installation of software applications, their integration and the testing of the proposed solution. The PoC is the result of building step by step on the project's previous architectural, technological and infrastructure outputs.

1.3 Scope of the PoC

The PoC described here captures the implemented parts of the proposed solution to the extent necessary to verify the basic functionality of the service. The subject of this output is mainly a description of the deployed components, their basic links and their operational arrangement in the project environment. The document focuses on what was actually deployed, configured and verified within KPB5. The PoC does not represent a final production solution or a complete commercial service. It is an experimental concept whose purpose is to demonstrate the technical feasibility of the proposed architecture and the basic operability of the solution as a whole. The scope of the PoC is therefore deliberately limited to the functions that it was possible to implement and verify under laboratory conditions.

1.4 PoC operating environment

The PoC is operated on a private OpenStack cloud environment, which forms the basic infrastructure layer of the solution. The individual components are deployed in virtual

machines, with containerization by means of Docker and Docker Compose also used depending on the nature of the specific tool. This model makes it possible to separate the individual functional blocks of the solution, simplify their management and at the same time retain flexibility in experimental deployment. The PoC operating environment has the character of a test and verification environment. Its purpose is not to provide a fully fledged production service, but to create a functional platform for the integration, configuration and verification of the proposed components and their mutual cooperation. The environment is therefore designed to enable repeated deployment, configuration changes and the execution of test scenarios.

1.5 Limitations of the PoC

The PoC provides an experimental proof of concept. Not all planned components and integration links had been brought to the final level of functionality at the time this output was prepared. In some cases, components were successfully deployed on their own, but their integration remained only partially implemented. The document thus captures the actual state of implementation, not the final state of the architecture. The limitations concern mainly the degree of completion of the individual integrations, the range of verified scenarios and the overall level of operational maturity of the solution. However, the PoC provides a sufficient basis for the technical verification of the concept and for the subsequent evaluation of the results achieved in a further output of the project.

2 COMPONENTS IMPLEMENTED IN THE INNOSOC POC

2.1 SIEM – ELK Stack

2.1.1 ELK Stack deployment model

Within the INNOSOC PoC solution, ELK is used as a SIEM consisting of the Elasticsearch index database, the Logstash parsing tool, which pre-processes the data format before storage in the database, and Kibana, which serves as the WEB GUI for the entire stack. Kibana displays the collected data and provides user and role management, creation of security rules, alert handling, index management, node management and more.

The entire ELK stack is deployed in LXC Linux containers. A separate container is created for each service. The operating system deployed in the containers is Ubuntu 24.04.2.

All LXC containers are deployed on a single physical server. The server has an Intel(R) Xeon(R) CPU E5-2690 processor, 2 physical processors, 8 cores per processor with a frequency of 2.90GHz. The server has 251GB of RAM. The storage space consists of a RAID of HDD disks with a size of 2.7T and a 1.8T WD Red SSD.

2.1.2 Elasticsearch deployment

The Elasticsearch container has the IP address 192.168.1.178. It runs an Elasticsearch cluster consisting of 7 elasticsearch nodes. The nodes have the roles master, hot, warm and cold. The master node is the main node; the other data nodes connect to it and join the cluster. Hot nodes are the nodes where the most recent data are kept. The data are stored on an SSD so that they can be written within the shortest possible time interval. The data are kept there for 5 days. After this period, they are moved to the warm nodes. There are 2 warm nodes and they use HDDs. The data are stored on the warm nodes for 5 to 14 days. The data are then moved to the cold node and deleted after 21 days.

Node	Disks	Retention period	Allocated RAM
master1	HDD	---	16GB
hot1	SSD	<5 days	31GB
hot2	SSD	<5 days	31GB
hot3	SSD	<5 days	31GB
warm1	HDD	<14 days	31GB
warm2	HDD	<14 days	31GB
cold1	HDD	<21 days	16GB

In the project, emphasis is placed on security. Since version 7, Elasticsearch uses by default the x-pack module, which provides several security functions, including authentication, authorization, encryption of communication and user management.

Communication between the nodes is encrypted using TLS certificates. Each node has its own TLS certificate. The communication between Logstash and Elastich search as well as between Kibana and Elasticsearch is also encrypted. Each Elasticsearch node has its own configuration file.

The certification authority (CA) used to issue the individual certificates is our own. Elastic provides the option of generating a CA using the *elasticsearch-certutil* tool, which is

part of the *Elasticsearch* distribution. This tool makes it possible to create a certification authority certificate as well as to issue further certificates signed by this certification authority.

In addition, *Elasticsearch* uses a so-called keystore. The keystore serves as storage for sensitive data such as login credentials, tokens or passwords of system users. In the solution, we use the keystore to store the credentials of predefined system accounts. This mechanism increases the security of the system because sensitive data are not stored directly in configuration files.

In addition to protecting sensitive data by means of the keystore, it is also necessary to secure the network communication of the *elk-elasticsearch* container itself. For this purpose, we used the iptables tool, with which we defined firewall rules. Access to the Secure Shell (SSH) service is permitted from the university range, and access to the *Elasticsearch* application is permitted only from the *Kibana*, *Logstash* and *Arkime* machines, defined on the basis of IP addresses.

As part of availability monitoring, in addition to the already mentioned monitoring of the ELK stack using metricbeat, monitoring using the libreNMS tool and the zabbix tool was introduced.

Example of a hot node configuration file

```
cluster.name: elastic-cluster
node.name: hot1
node.roles: [data_hot]
path.data: /mnt/hot-nodes/elasticsearch-hot1
path.logs: /var/log/elasticsearch-hot1
network.host: 192.168.1.178
http.port: 9201
transport.port: 9301
discovery.seed_hosts: ["192.168.1.178:9300", "192.168.1.178:9302", "192.168.1.178:9303",
"192.168.1.178:9304", "192.168.1.178:9305", "192.168.1.178:9306"]
cluster.initial_master_nodes: ["master1"]
xpack.security.enabled: true
xpack.security.enrollment.enabled: true
xpack.security.http.ssl:
  enabled: true
  keystore.path: certs/elasticsearch/hot1/http.p12
xpack.security.transport.ssl:
  enabled: true
  verification_mode: certificate
  client_authentication: required
  keystore.path: certs/elastic-certificates.p12
  truststore.path: certs/elastic-certificates.p12
http.host: 0.0.0.0
transport.host: 0.0.0.0
xpack.security.audit.enabled: true
xpack.security.audit.logfile.events.include: access_denied, anonymous_access_denied,
authentication_failed, connection_denied, tampered_request, run_as_denied,
security_config_change
xpack.security.audit.logfile.emit_node_host_address: true
xpack.security.audit.logfile.emit_node_name: true
xpack.security.audit.logfile.emit_node_host_name: true
```

2.1.3 Logstash deployment

Logstash processes the data received from the auditbeat and filebeat agents, which are deployed on the other servers at FRI UNIZA. Communication between the agents and Logstash is also encrypted. The Logstash pipeline is divided into three configuration files. For the input, filtering and output of the collected data.

For the input, filtering and output of the collected data:

- input.conf – defines the sources of the received data,
- filters.conf – processes the received data, applies filters to the data, ensures data normalization
- output.conf – handles sending of the data.

As with the Elasticsearch tool, we also performed a clean installation for the Logstash tool. On the machine ucs-lxc-1 with the IP address 192.168.2.11, an LXC container with the Ubuntu 24.04 LTS operating system was created for the deployment of the Logstash tool. The container is named elk-logstash and has the assigned IP address 192.168.1.177.

Logstash is deployed in version 8.17.2. Its configuration defines the size of the main memory, similarly to the Elasticsearch nodes, with 16 GB allocated. This value provides a sufficient reserve also for future expansion of the infrastructure and the connection of further monitored devices.

Communication between Logstash and Elasticsearch is secured by a certificate on the Elasticsearch side. Logstash authenticates to Elasticsearch using a username and password. Since we sought to adhere to the principle of the lowest necessary permissions, the so-called "least privilege", it was necessary to create a user who was assigned a role with permissions. The created role is named logstash_writer and the created user logstash_internal. We assigned the following cluster permissions to this role:

- monitor - monitoring the state of the cluster and nodes
- manage_index_template - management of index templates
- manage_ilm - index life cycle management
- manage_pipeline - management of ingest pipelines

And permissions for individual indices:

- write - writing data to an index
- create - creating new documents
- create_index - creating new indices
- manage - full management of an index
- read - reading data from an index
- index - adding or updating documents

The indices to which the privileges are to be assigned were defined using a so-called index pattern. An index pattern is a template or rule that defines a specific index. So-called wildcards are used in them, in the form of the character " * ", in combination with text that forms part of the index name. For our role, we defined the following index patterns:

1. gvm-greenbone*
2. elasticsearch-elk-elasticsearch*
3. syslog-fortigate*

4. waf-filebeat*
5. auditbeat*
6. suricata*
7. threatintel-filebeat*

Logstash uses a keystore to store the user password for the Logstash API and the password for the logstash_internal user. Network communication is secured using the iptables tool, with which we defined firewall rules. Access to the SSH service and for receiving data via Logstash is permitted from a specific IPv4 range.

As part of availability monitoring, Logstash is monitored using metricbeat and also using the zabbix and libreNMS tools.

2.1.3.1 Example of the input.conf configuration file for the deployed Logstash

The configuration file sets up data reception using the beat module, which receives data on port 5044. Furthermore, Secure Sockets Layer (SSL)/Transport Layer Security (TLS) encryption is set and the paths to the certificates are defined. The certificates were signed by our own certification authority. Logstash requires clients to present a valid certificate, otherwise the connection will not be established, and it also provides its own TLS certificate with which clients verify the identity of Logstash. This is mutual TLS authentication.

```
input {
        beats {
                port => 5044
                ssl_enabled => true
                ssl_certificate => "/etc/logstash/certs/logstash/logstash.crt"
                ssl_key => "/etc/logstash/certs/logstash/logstash.key"
                ssl_certificate_authorities => ["/etc/logstash/certs/ca.crt"]
                ssl_client_authentication => "required"
        }
}
```

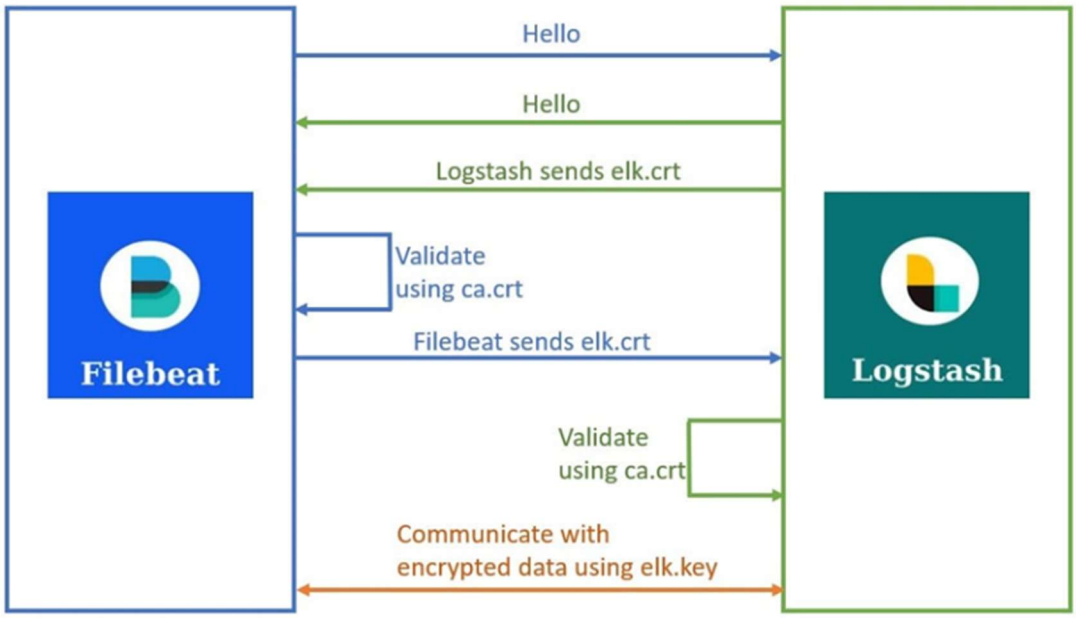


Illustration of mutual TLS

2.1.3.2 Example of the filters.conf configuration file for the deployed Logstash

```
filter { if ("suricata-moloch" in [tags]) or ("suricata-suricata2" in [tags]) { json { source =>
"[event][original]" target => "suricata_parsed" } date { match =>
["[suricata_parsed][timestamp]", "yyyy-MM-dd'T'HH:mm:ss.SSSSSSZ"] target =>
"@timestamp" } mutate { rename => { "[suricata_parsed][src_ip]" => "source.ip"
"[suricata_parsed][src_port]" => "source.port" "[suricata_parsed][dest_ip]" => "destination.ip"
"[suricata_parsed][dest_port]" => "destination.port" "[suricata_parsed][proto]" =>
"network.protocol" "[suricata_parsed][event_type]" => "event.type"
"[suricata_parsed][flow_id]" => "network.flow_id" "[suricata_parsed][app_proto]" =>
"network.application_protocol" "[suricata_parsed][in_iface]" => "network.interface"
"[suricata_parsed][alert][signature]" => "rule.name" "[suricata_parsed][alert][signature_id]" =>
"rule.id" "[suricata_parsed][alert][category]" => "rule.category"
"[suricata_parsed][alert][severity]" => "rule.severity" "[suricata_parsed][alert][action]" =>
"rule.action" "[suricata_parsed][alert][rev]" => "rule.revision" "[suricata_parsed][alert][gid]" =>
"rule.gid" "[suricata_parsed][flow][pkts_toserver]" => "network.packets.to_server"
"[suricata_parsed][flow][pkts_toclient]" => "network.packets.to_client"
"[suricata_parsed][flow][bytes_toserver]" => "network.bytes.to_server"
"[suricata_parsed][flow][bytes_toclient]" => "network.bytes.to_client"
"[suricata_parsed][flow][start]" => "network.flow_start"
"[suricata_parsed][alert][metadata][signature_severity]" => "rule.signature_severity"
"[suricata_parsed][alert][metadata][created_at]" => "rule.created_at"
"[suricata_parsed][alert][metadata][updated_at]" => "rule.updated_at"
"[suricata_parsed][alert][metadata][cve]" => "rule.cve_list"
"[suricata_parsed][alert][metadata][deployment]" => "rule.deployment"
"[suricata_parsed][alert][metadata][attack_target]" => "rule.attack_target"
"[suricata_parsed][alert][metadata][affected_product]" => "rule.affected_product"
"[suricata_parsed][alert][metadata][tag]" => "rule.tag" } convert => { "network.flow_id" =>
```

```
"integer" "source.port" => "integer" "destination.port" => "integer" "network.packets.to_server"
=> "integer" "network.packets.to_client" => "integer" "network.bytes.to_server" => "integer"
"network.bytes.to_client" => "integer" "rule.id" => "integer" "rule.severity" => "integer"
"rule.revision" => "integer" "rule.gid" => "integer" } } geoip { source => "source.ip" target =>
"source" database => "/etc/logstash/geoip/GeoLite2-City.mmdb" } geoip { source =>
"destination.ip" target => "destination" database => "/etc/logstash/geoip/GeoLite2-
City.mmdb" } mutate { remove_field => [ "source.ip", "destination.ip"] rename => {
"[suricata_parsed][src_ip]" => "source.ip" "[suricata_parsed][dest_ip]" => "destination.ip" } }
mutate { remove_field => [ "message", "suricata_parsed"] } }

else if ("auditbeat-studenti" in [tags]) or ("auditbeat-admin" in [tags]) or ("auditbeat-testing" in
[tags]) or ("auditbeat-elk" in [tags]){ mutate { add_field => { "[@metadata][index]" =>
"auditbeat-logs" } add_field => { "event.ingested" => "%{+YYYY-MM-dd'T'HH:mm:ss.SSSZ}"
} rename => { "[process][parent][process][name]" => "process.parent.name"
"[process][parent][process][executable]" => "process.parent.executable"
"[process][parent][process][args]" => "process.parent.args" "[process][parent][process][pid]"
=> "process.parent.pid" "[process][parent][process][entity_id]" => "process.parent.entity_id"
"[process][parent][process][title]" => "process.parent.title"
"[process][parent][process][start_time]" => "process.parent.start_time"
"[process][parent][process][owner]" => "process.parent.owner"
"[process][parent][process][group]" => "process.parent.group"
"[process][parent][process][thread]" => "process.parent.thread" } } mutate { replace => {
"process.parent.pid" => "%{[process][parent][pid][0]}" } }

mutate {
  remove_field => ["process.parent.pid"]
}
mutate {
  convert => { "process.parent.pid" => "integer" }
}
if [process][args] and ( [process][args] != "" ) {

  mutate {
    add_field => { "[process][command_line]" => "%{[process][args]}" }
  }
  mutate {
    gsub => [
      "[process][command_line]", '\"', '',
      "[process][command_line]", '\[', '',
      "[process][command_line]", '\]', '',
      "[process][command_line]", ',', ' '
    ]
  }
}


} else if "syslog-fortigate" in [log_type] { grok { match => { "message" =>
"%{SYSLOGTIMESTAMP:syslog_timestamp} %{IPORHOST:syslog_host}
%{GREEDYDATA:event.original}" } overwrite => ["event.original"] } kv { source =>
"event.original" field_split => " " value_split => "=" trim_key => """ trim_value => """ } mutate {
rename => { "srcip" => "source.ip" "dstip" => "destination.ip" } convert => { "srcport" =>
"integer" "dstport" => "integer" "sentbyte" => "integer" "rcvdbyte" => "integer" "sentpkt" =>
```

```
"integer" "rcvdpkt" => "integer" "duration" => "integer" } add_field => { "datetime" => "%{date} %{time}" } } date { match => ["datetime", "yyyy-MM-dd HH:mm:ss"] target => "@timestamp" timezone => "+0100" } mutate { remove_field => ["datetime", "message", "date"] } if [agent] { mutate { add_field => { "agent_name" => "%{[agent][name]}" "agent_id" => "%{[agent][id]}" "agent_type" => "%{[agent][type]}" "agent_version" => "%{[agent][version]}" "agent_ephemeral_id" => "%{[agent][ephemeral_id]}" } } mutate { remove_field => ["agent"] }

 } } else if "CTF-win" in [tags]{ mutate { add_field => { "[@metadata][index]" => "ctf-win-logs" } add_field => { "event.ingested" => "%{+YYYY-MM-dd'T'HH:mm:ss.SSSZ}" } } if [winlog][provider_name] == "Microsoft-Windows-Sysmon" { mutate { rename => { "[winlog][event_id]" => "event.sysmon_id" "[winlog][event_data][Image]" => "process.name" "[winlog][event_data][CommandLine]" => "process.command_line" "[winlog][event_data][ParentImage]" => "process.parent.name" "[winlog][event_data][ParentCommandLine]" => "process.parent.command_line" "[winlog][event_data][User]" => "user.name" "[winlog][event_data][Hashes]" => "file.hashes" "[winlog][event_data][TargetFilename]" => "file.target_name" "[winlog][event_data][CreationUtcTime]" => "file.creation_time" "[winlog][event_data][ProcessId]" => "process.id" "[winlog][event_data][ParentProcessId]" => "process.parent.id" } } if [file][hashes] { ruby { code => ' hashes = event.get("[file][hashes]") if hashes hashes.split(",").each do |hash| key_value = hash.split("=") if key_value.length == 2 end end end ' } } } if [winlog][provider_name] == "PowerShell" { mutate { rename => { "[winlog][event_id]" => "event.powershell_id" "[winlog][event_data][param1]" => "powershell.command" "[winlog][event_data][param2]" => "powershell.details" "[winlog][event_data][param3]" => "powershell.output" } } } if [host][name] { mutate { rename => { "[host][name]" => "system.hostname" "[host][os][name]" => "system.os_name" "[host][os][version]" => "system.os_version" } } } if [agent][version] { mutate { rename => { "[agent][version]" => "logstash.agent_version" } } } mutate { remove_field => [ "[winlog][provider_guid]", "[winlog][record_id]", "[winlog][opcode]", "[winlog][api]" ]

 } } else if "filebeat-gvm" in [tags] {

if [message] =~ "^\{.*\}$" {
  json {
    source => "message"
  }
} else {
  grok {
    match => { "message" => "%{IPORHOST:client_ip} - - \[%{HTTPDATE:log_timestamp}\] \"%{WORD:http_method} %{URIPATHPARAM:request} HTTP/%{NUMBER:http_version}\" %{NUMBER:response_code} (?:%{NUMBER:bytes_sent}|-) \"%{DATA:referrer}\" \"%{DATA:user_agent}\"" }
  }
}
date {
  match => [ "log_timestamp", "dd/MMM/yyyy:HH:mm:ss Z" ]
  target => "@timestamp"
}
mutate {
  remove_field => ["log_timestamp"]
}
```

```
} else if "filebeat-elasticsearch" in [tags] { json { source => "message" target => "audit"
skip_on_invalid_json => true } mutate { add_field => { "user" => "%{[audit][user.name]}"
"action" => "%{[audit][event.action]}" "timestamp" => "%{[audit][timestamp]}" } } date { match
=> ["[audit][@timestamp]", "yyyy-MM-dd'T'HH:mm:ss,SSSZ"] target => "@timestamp" } } else
if "filebeat-WAF" in [tags] { if [log][file][path] == "/var/log/nginx/access.log" { mutate { add_tag
=> [ "nginx-access" ] } grok { match => { "message" => '%{IPORHOST:remote_ip} -
%{DATA:remote_user} [%{HTTPDATE:timestamp}] "(?:%{WORD:method}
%{URIPATHPARAM:request}(?: HTTP/%{NUMBER:http_version})?)" %{NUMBER:status}
(?:%{NUMBER:bytes}|-) "(?:%{URI:referrer}|-)" "(?:%{DATA:user_agent}|-)"' } remove_field
=> [ "event.original" ] } date { match => [ "timestamp", "dd/MMM/yyyy:HH:mm:ss Z" ] target
=> "@timestamp" } } else if [log][file][path] == "/var/log/nginx/error.log" { mutate { add_tag => [
"nginx-error" ] } grok { match => { "message" => '[%{DAY} %{MONTH} %{MONTHDAY}
%{TIME} %{YEAR}] [%{LOGLEVEL:log_level}]( [%{DATA:module}])?
%{GREEDYDATA:error_message}' } remove_field => [ "event.original" ] } date { match => [
"timestamp", "dd/MMM/yyyy:HH:mm:ss Z" ] target => "@timestamp" } } else if [log][file][path]
== "/var/log/modsec/kibana.fri.uniza.sk_audit.log" { mutate { add_tag => [ "nginx-audit" ] }
grok { match => { "message" => "(?m)---%{DATA:modsec_id}---A--
\[%{HTTPDATE:timestamp}\] %{NUMBER:unique_id} %{IP:src_ip} %{NUMBER:src_port}
%{IP:dst_ip} %{NUMBER:dst_port}.*?---%{DATA:modsec_id}---B--
%{GREEDYDATA:http_request}---%{DATA:modsec_id}---F--
%{GREEDYDATA:http_response}---%{DATA:modsec_id}---H--
%{GREEDYDATA:modsec_alerts}" } tag_on_failure => [ "_modsec_grok_fail" ] } date { match
=> [ "timestamp", "dd/MMM/yyyy:HH:mm:ss Z" ] target => "@timestamp" } mutate {
remove_field => [ "event.original" ] } } } else if "filebeat-threatintel" in [tags] { date { match =>
["[audit][@timestamp]", "yyyy-MM-dd'T'HH:mm:ss,SSSZ"] target => "@timestamp" } if
[fileset][name] == "abuseurl" { json { source => "message" target => "abuseurl" } mutate {
add_field => { "threat_type" => "%{[abuseurl][threat]}" "url" => "%{[abuseurl][url]}" "ip" =>
"%{[abuseurl][host]}" "reporter" => "%{[abuseurl][reporter]}" "malware_tag" =>
"%{[abuseurl][tags][0]}" "urlhaus_ref" => "%{[abuseurl][urlhaus_reference]}" } } } else if
[fileset][name] == "abusemalware" { json { source => "message" target => "abusemalware" }
mutate { add_field => { "sha256" => "%{[abusemalware][sha256_hash]}" "md5" =>
"%{[abusemalware][md5_hash]}" "signature" => "%{[abusemalware][signature]}" "vt_result"
=> "%{[abusemalware][virustotal][result]}" "vt_percent" =>
"%{[abusemalware][virustotal][percent]}" "download_url" =>
"%{[abusemalware][urlhaus_download]}" } } } else if [fileset][name] == "otx" { json { source =>
"message" target => "otx" } mutate { add_field => { "indicator" => "%{[otx][indicator]}"
"indicator_type" => "%{[otx][type]}" "description" => "%{[otx][description]}" } } }

 } else if "arkimecapture-moloch" in [tags] { grok { match => { "message" =>
"%{MONTH:log_month} %{MONTHDAY:log_day} %{TIME:log_time}
%{GREEDYDATA:rest_of_message}" } } mutate { add_field => { "log_timestamp" =>
"%{log_month} %{log_day} %{log_time}" } } date { match => [ "log_timestamp", "MMM dd
HH:mm:ss" ] target => "@timestamp" timezone => "Europe/Bratislava" } grok { match => {
"message" => "drop: %{NUMBER:drop_count:int} (%{NUMBER:drop_percent:float})" } } grok
{ match => { "message" => "recv: %{NUMBER:recv_count:int}" } } } }
```

2.1.3.3 Example of the output.conf configuration file for the deployed Logstash

```
output {
  if "auditbeat-studenti" in [tags] {
    elasticsearch {
      hosts => ["https://192.168.1.178:9200"]
```

```
    user => "logstash_internal"
    password => "${ES_psswd}"
    ssl_enabled => true
    ssl_certificate_authorities => "/etc/logstash/certs/ca.crt"
    ilm_enabled => true
    ilm_rollover_alias => "auditbeat-logs-studenti"
    ilm_pattern => "{now/d}-000001"
    ilm_policy => "auditbeat-logs-ILM"
    manage_template => false
  }
}
else if "suricata-moloch" in [tags] {
  elasticsearch {
    hosts => ["https://192.168.1.178:9200"]
    user => "logstash_internal"
    password => "${ES_psswd}"
    ssl_enabled => true
    ssl_certificate_authorities => "/etc/logstash/certs/ca.crt"
    ilm_enabled => true
    ilm_rollover_alias => "suricata-moloch"
    ilm_pattern => "{now/d}-000001"
    ilm_policy => "suricata-moloch-ILM"
    manage_template => false
  }
}
else if "syslog-fortigate" in [log_type] {
  elasticsearch {
    hosts => ["https://192.168.1.178:9200"]
    user => "logstash_internal"
    password => "${ES_psswd}"
    ssl_enabled => true
    ssl_certificate_authorities => "/etc/logstash/certs/ca.crt"
    ilm_enabled => true
    ilm_rollover_alias => "syslog-fortigate"
    ilm_pattern => "{now/d}-000001"
    ilm_policy => "syslog-fortigate-ILM"
    manage_template => false
  }
}
else if "auditbeat-admin" in [tags] {
  elasticsearch {
    hosts => ["https://192.168.1.178:9200"]
    user => "logstash_internal"
    password => "${ES_psswd}"
    ssl_enabled => true
    ssl_certificate_authorities => "/etc/logstash/certs/ca.crt"
    ilm_enabled => true
    ilm_rollover_alias => "auditbeat-logs-admin"
    ilm_pattern => "{now/d}-000001"
    ilm_policy => "auditbeat-logs-ILM"
    manage_template => false
```

```
  }
}
else if "suricata-suricata2" in [tags] {
  elasticsearch {
    hosts => ["https://192.168.1.178:9200"]
    user => "logstash_internal"
    password => "${ES_psswd}"
    ssl_enabled => true
    ssl_certificate_authorities => "/etc/logstash/certs/ca.crt"
    ilm_enabled => true
    ilm_rollover_alias => "suricata-suricata2"
    ilm_pattern => "{now/d}-000001"
    ilm_policy => "suricata-suricata2-ILM"
    manage_template => false
  }
}
else if "auditbeat-testing" in [tags] {
  elasticsearch {
    hosts => ["https://192.168.1.178:9200"]
    user => "logstash_internal"
    password => "${ES_psswd}"
    ssl_enabled => true
    ssl_certificate_authorities => "/etc/logstash/certs/ca.crt"
    index => "auditbeat-testing"
    manage_template => false
  }
}
else if "filebeat-gvm" in [tags] {
  elasticsearch {
    hosts => ["https://192.168.1.178:9200"]
    user => "logstash_internal"
    password => "${ES_psswd}"
    ssl_enabled => true
    ssl_certificate_authorities => "/etc/logstash/certs/ca.crt"
    ilm_enabled => true
    ilm_rollover_alias => "gvm-greenbone"
    ilm_pattern => "{now/d}-000001"
    ilm_policy => "gvm-greenbone-ILM"
    manage_template => false
  }
}
else if "auditbeat-elk" in [tags] {
  elasticsearch {
    hosts => ["https://192.168.1.178:9200"]
    user => "logstash_internal"
    password => "${ES_psswd}"
    ssl_enabled => true
    ssl_certificate_authorities => "/etc/logstash/certs/ca.crt"
    ilm_enabled => true
    ilm_rollover_alias => "auditbeat-logs-elk"
    ilm_pattern => "{now/d}-000001"
```

```
    ilm_policy => "auditbeat-logs-ILM"
    manage_template => false
  }
}
else if "filebeat-elasticsearch" in [tags] {
  elasticsearch {
    hosts => ["https://192.168.1.178:9200"]
    user => "logstash_internal"
    password => "${ES_psswd}"
    ssl_enabled => true
    ssl_certificate_authorities => "/etc/logstash/certs/ca.crt"
    ilm_enabled => true
    ilm_rollover_alias => "elasticsearch-elk-elasticsearch"
    ilm_pattern => "{now/d}-000001"
    ilm_policy => "elasticsearch-elk-elasticsearch-ILM"
    manage_template => false
  }
}
else if "filebeat-WAF" in [tags] {
  elasticsearch {
    hosts => ["https://192.168.1.178:9200"]
    user => "logstash_internal"
    password => "${ES_psswd}"
    ssl_enabled => true
    ssl_certificate_authorities => "/etc/logstash/certs/ca.crt"
    ilm_enabled => true
    ilm_rollover_alias => "waf-filebeat"
    ilm_pattern => "{now/d}-000001"
    ilm_policy => "waf-filebeat-ILM"
    manage_template => false
  }
}
else if "filebeat-threatintel" in [tags] {
  elasticsearch {
    hosts => ["https://192.168.1.178:9200"]
    user => "logstash_internal"
    password => "${ES_psswd}"
    ssl_enabled => true
    ssl_certificate_authorities => "/etc/logstash/certs/ca.crt"
    ilm_enabled => true
    ilm_rollover_alias => "threatintel-filebeat"
    ilm_pattern => "{now/d}-000001"
    ilm_policy => "threatintel-filebeat-ILM"
    manage_template => false
  }
}
else if "arkimecapture-moloch" in [tags] {
  elasticsearch {
    hosts => ["https://192.168.1.178:9200"]
    user => "logstash_internal"
    password => "${ES_psswd}"
```

```
    ssl_enabled => true
    ssl_certificate_authorities => "/etc/logstash/certs/ca.crt"
    ilm_enabled => true
    ilm_rollover_alias => "arkimecapture-moloch"
    ilm_pattern => "{now/d}-000001"
    ilm_policy => "arkimecapture-moloch-ILM"
    manage_template => false
  }
 }
}
```

2.2 Shuffle

Shuffle Automation is used within the INNOSOC PoC solution for the automated execution of processes and the transformation of data between various security tools and systems (SOAR, Security Orchestration, Automation and Response). Within the solution, this involves the orchestration of data flows, which ensures that data from one system are automatically processed, transformed and redirected to other systems without human intervention.

2.2.1 Shuffle deployment model

The solution runs in Docker containers, with the backend built on OpenSearch, which serves as storage for workflows and Python applications. The web interface is accessible on port 3001.

Technical specification of the Shuffle deployment:

- Environment – OpenStack
- Instance name – shuffle
- IP address – 192.168.4.188
- vCPU – 4
- RAM – 16 GB
- Disk – 80 GB
- OS – Ubuntu 24.04

The Shuffle software is built on Docker technology. This means that, in order for it to be fully functional, it is advisable to have the system on which it will be deployed running the latest version of Docker. The Shuffle installation comes with 2 files, docker-compose and a .env file. Both files are used to configure ports, port forwarding, the backend and the frontend. For Shuffle to be fully functional, it is sufficient to start it with the command sudo docker-compose up –d.

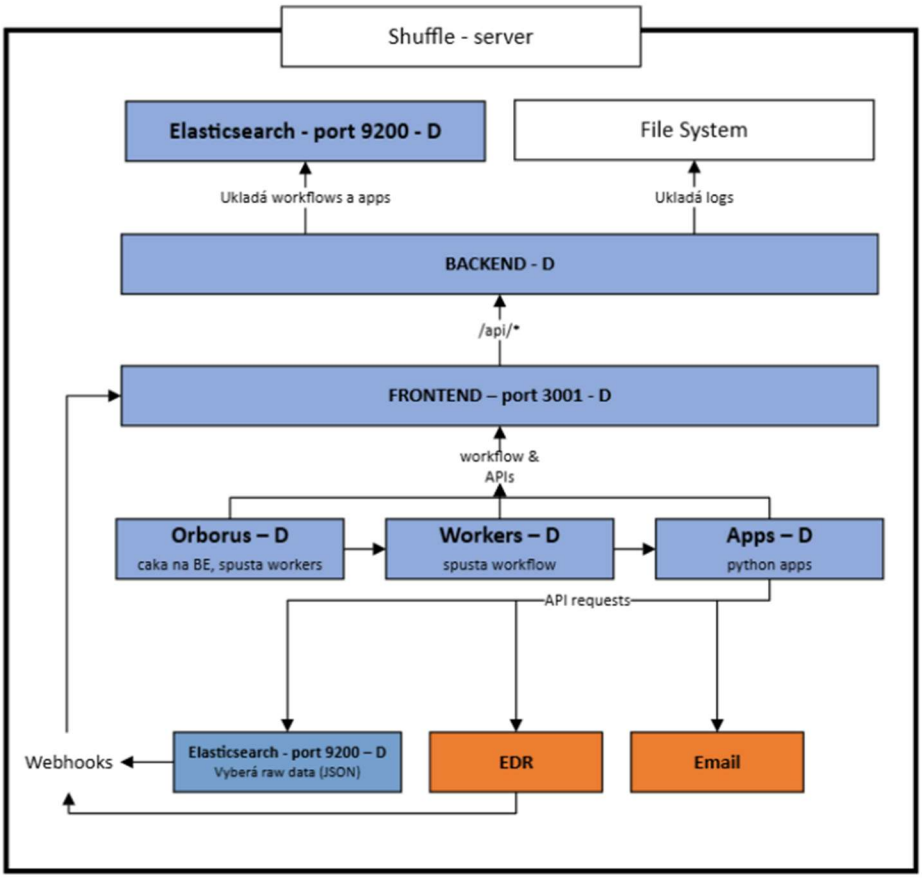


Figure 1 Architecture of the Shuffle components

The figure illustrates the architecture of the Shuffle SOAR platform on the server and the data processing flow. The frontend runs on port 3001 and serves as the web interface through which the user works with workflows and the API. Requests from the frontend are routed to the backend, which provides the application logic and communication with the Elasticsearch database, where workflows and applications are stored, and with the file system, in which logs are stored. The runtime environment contains the Orborus, Workers and Apps components, with Orborus overseeing the backend and launching workers, which execute the individual workflow steps. Apps are scripts, most often written in Python, that communicate with external services. Through API requests and webhooks, Shuffle is connected to external systems such as Elasticsearch, EDR or e-mail services.

2.2.2 Basic integration links of the Shuffle component in the PoC

Services currently connected and tested using Shuffle:

- ELK – data can be retrieved via API calls, and indices can be searched and new indices created. However, the IP address of the device on which Shuffle runs must be allowed in iptables on the device where ELK runs.
- Discord – notifications can be sent in JSON format using a webhook that is created in a group on Discord.
- OpenCTI – new indicators can be created via the API; we create these as JSON and send them via an API call.

- DFIR-IRIS – new events can be created via the API, which can likewise be sent as JSON.

2.3 Wazuh

Wazuh is used within the INNOSOC PoC solution as the central platform for the collection, correlation and analysis of security events, with EDR functionality.

2.3.1 Deployment model of the Wazuh component

The architecture of our solution is based on deploying the universal Wazuh agent on target systems and on three key central components:

- Wazuh-manager (server): Provides data analysis, agent management and threat detection.
- Wazuh-indexer (database): A high-performance full-text search and analytics engine used for indexing and storing security records.
- Wazuh-dashboard: A web user interface designed for data visualization, incident analysis and system configuration

The central components are deployed on a virtual machine (VM) in the XenOrchestra environment. The VM has the following available: 8 virtual CPUs, 8GB of RAM and 250GB of storage.

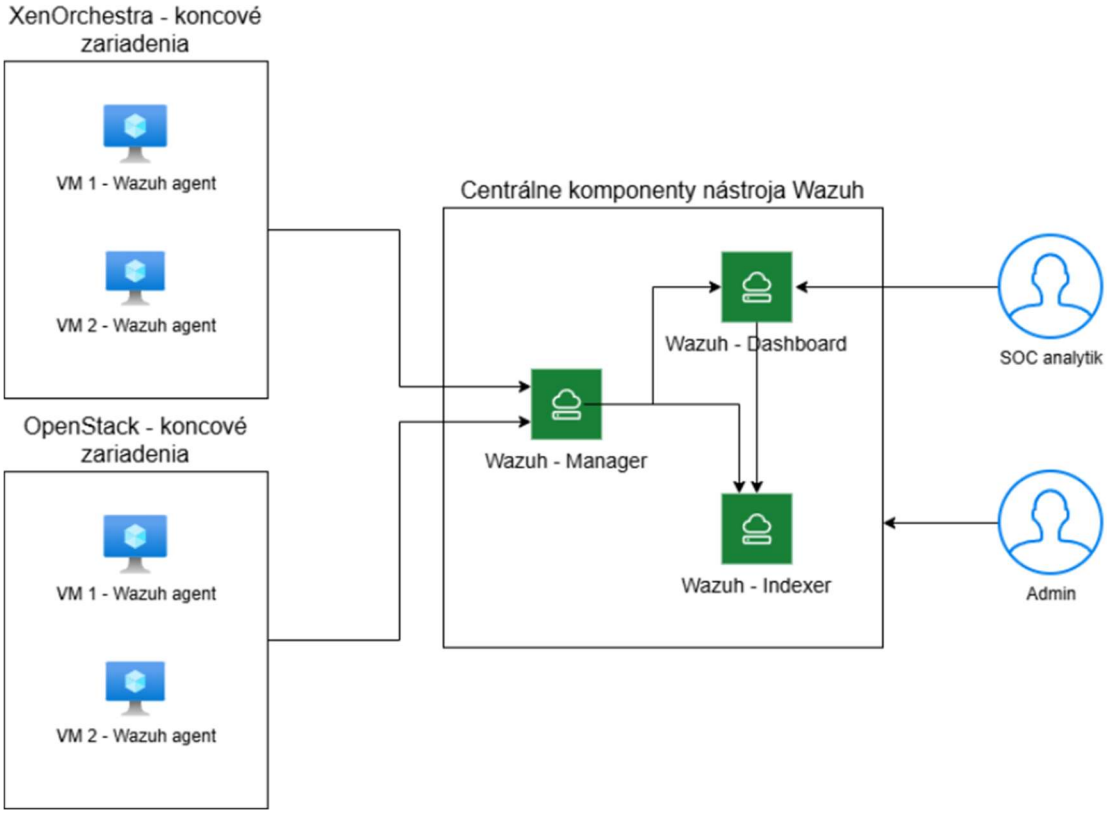
2.3.2 Environments monitored by Wazuh

We monitor the environments of two tenants. The first are the XenOrchestra and OpenStack cloud environments of the Department of Information Networks at the Faculty of Management Science and Informatics. The second is the environment of a selected primary school (ZŠ) in Žilina.

2.3.2.1 Environment of the Department of Information Networks, FRI UNIZA

At KIS, the system is configured as an All-in-one installation, with all key components of the Wazuh platform integrated into a single virtual machine (VM). This solution provides centralized collection and analysis of data from the monitored nodes located in the XenOrchestra and OpenStack environments. A graphical representation of the resulting network architecture and component hierarchy is illustrated in the figure below.

Since this is an All-in-one deployment, all central components have the same IP address: 192.168.1.184. In the case of a Single or multi-node installation, each component needs to have its own IP address (since each component will be on a separate server or VM).



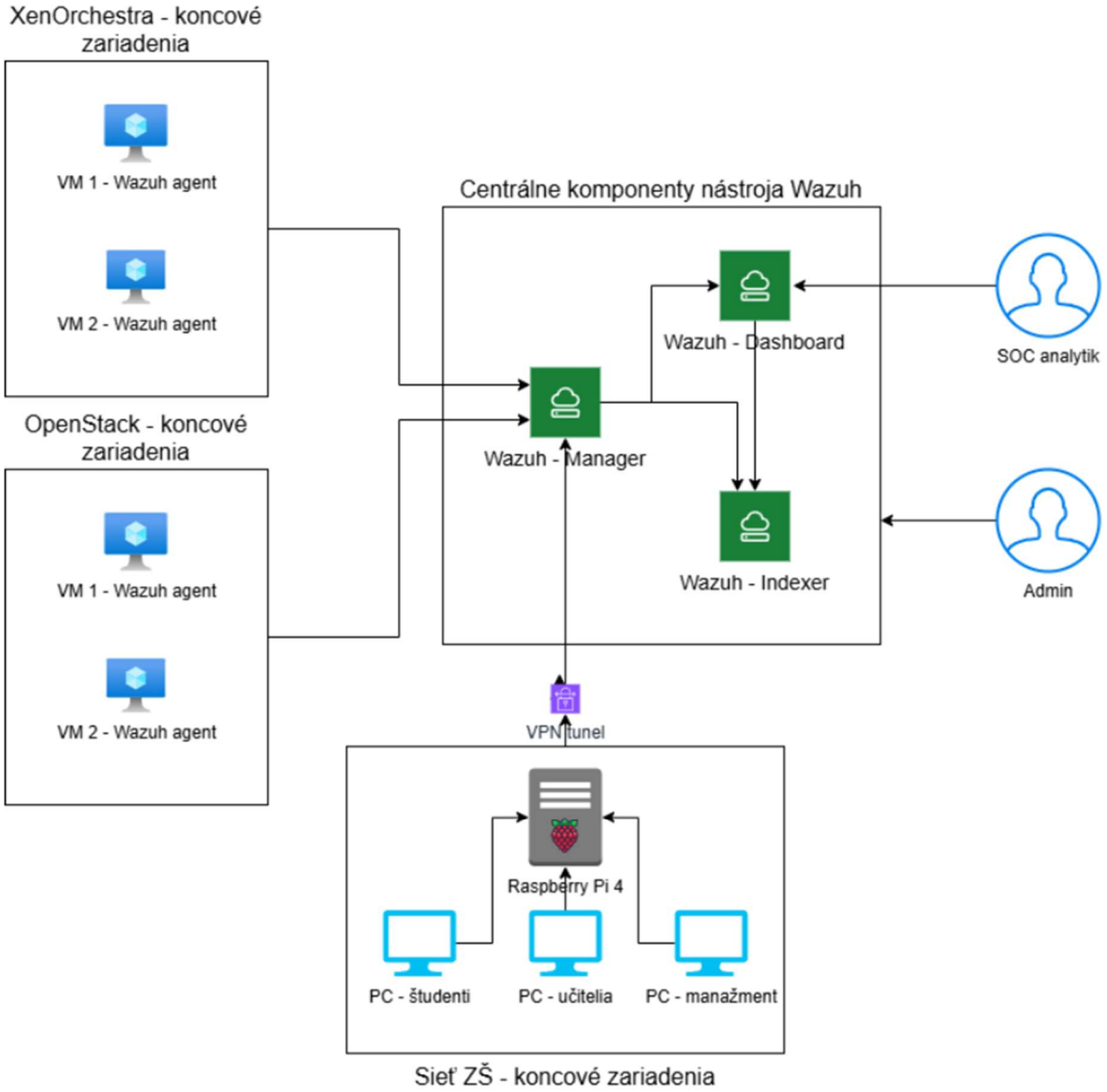
2.3.2.2 Monitoring of a selected primary school (ZŠ) in Žilina

The second instance of the monitoring system is implemented in the infrastructure of a selected primary school (ZŠ) in Žilina. The original architecture in this environment consisted of a local server on which the basic components of the Wazuh platform were deployed. This server provided the collection and storage of data from endpoint devices within the school's local network, while remote access to the Wazuh Dashboard interface was provided via the ZeroTier VPN technology.

Owing to inefficient administration and the need for redundant configuration of two separate managers, the architecture was revised. It was decided to centralize the main components in the XenOrchestra environment at KIS FRI. Data from endpoint devices located at the school are now redirected to and processed centrally at KIS.

The interconnection of the two sites is technically implemented as follows:

- **KIS side:** In addition to the Wazuh manager, the server components of the WireGuard VPN are also operated on the central server.
- **School side:** A Raspberry Pi 4 device acting as the VPN client is deployed in the local network.
- **Network connectivity:** A functional VPN tunnel with the address range 10.0.0.0/24 is established between the sites.
- **Routing:** A static route is defined on the local router at the school, which routes all traffic destined for the 10.0.0.0/24 network through the Raspberry Pi gateway.



- **Wazuh agents** installed on endpoint devices at the school send the collected data directly to the Wazuh manager in the KIS network via the secured tunnel.

2.3.3 Deployed and activated Wazuh modules

In the initial phase, native Wazuh modules were activated and configured, forming the basic framework for security monitoring and compliance. Selected functions are specified in detail below:

2.3.3.1 File Integrity Monitoring (FIM)

This module monitors changes in critical system files and directories. It works on the principle of comparing checksums and file attributes over time.

- **Mechanism:** On initialization, the agent creates a database of hashes of selected files. On every change (creation, modification, deletion), it sends an alert to the manager.
- **Use:** Detection of unauthorized configuration changes or malware injection.

2.3.3.2 Rootcheck (Rootkit Detection)

A module focused on detecting hidden threats such as rootkits, Trojan horses or unauthorized modifications of the operating system kernel.

- **Mechanism:** It performs an in-depth check for anomalies in the system, compares the outputs of system calls with a direct reading of the original (so-called raw) data and searches for hidden processes or ports.

2.3.3.3 Vulnerability Detection

This module automatically identifies weaknesses in the operating system and installed applications.

- **Mechanism:** The Wazuh agent collects a list of installed packages, which the manager then compares against global vulnerability databases (**CVE** – Common Vulnerabilities and Exposures) in the NVD - National Vulnerability Database.

2.3.3.4 Policy monitoring & SCA (Security Configuration Assesment)

It is used to verify whether the system is configured in compliance with security standards (e.g. CIS Benchmarks).

- **Mechanism:** The module runs scripts that check system settings (e.g. password strength, disabled protocols). The result is a compliance score and a list of remediation recommendations.

2.3.3.5 System Inventory

It provides a comprehensive overview of the hardware and software properties of the monitored endpoint.

Mechanism: It collects data on network interfaces, running processes, installed packages and OS versions. These data are available in real time via wazuh-dashboard.

2.3.4 Basic integration links – sending notifications

The Wazuh ecosystem offers a wide range of integration options for the automated sending of alerts via external communication channels and third-party tools.

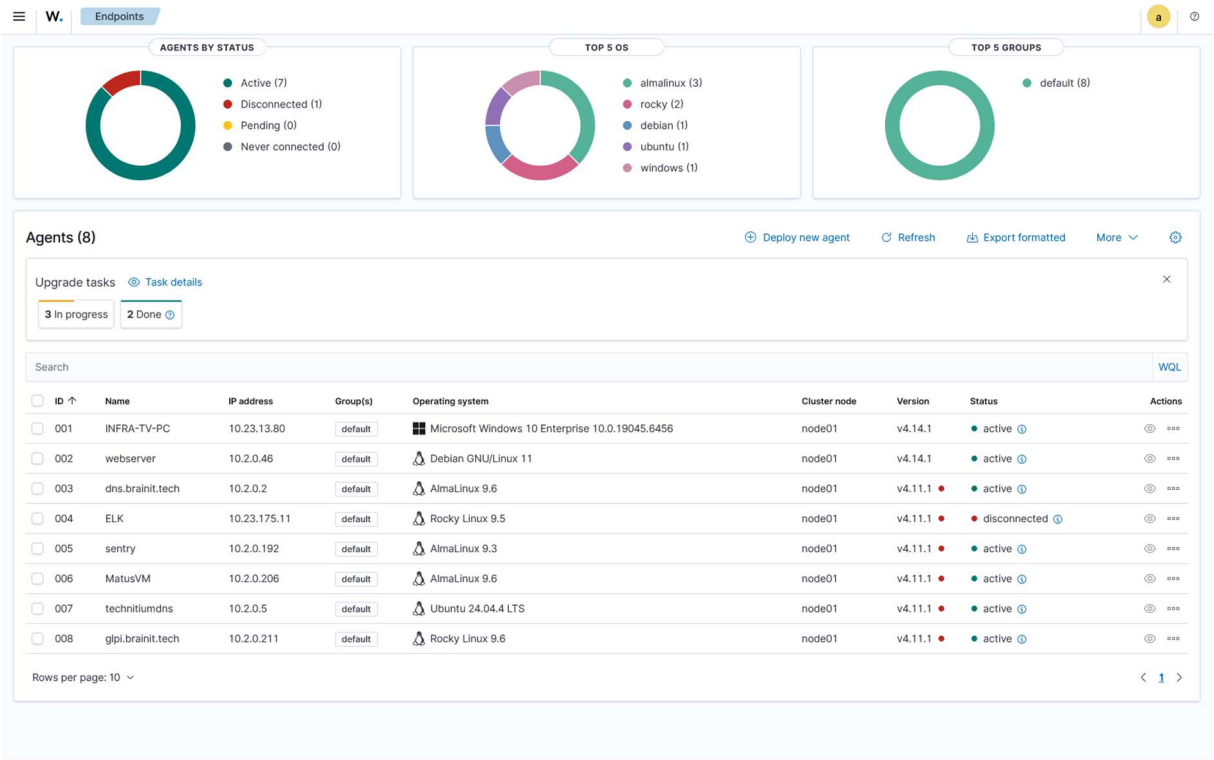
Sending notifications to external platforms (Slack, MS Teams, PagerDuty, etc.) is handled in Wazuh through the Integrations module. The integration is implemented via the Webhook application interface, with the system enabling granular filtering of the events sent based on their severity. This process makes it possible to transform internal alerts into immediate messages for the security team, thereby significantly shortening the response time (MTTR – Mean Time to Respond).

We analysed various tools for sending notifications, such as Slack, PagerDuty and MS Teams. For monitoring purposes, we decided to use MS Teams. The integration is built on the use of an Incoming Webhook (HTTP endpoint). The Wazuh platform sends data in JSON format, which are transformed in Teams into visually structured cards (Adaptive Cards). This format enables a clear display of key metrics such as the severity level, agent identification, rule ID and a detailed description of the event.

The main reason for this choice was the fact that this platform is fully integrated into the university's existing digital infrastructure. Using the Microsoft 365 ecosystem brings us several advantages:

- **Deployment efficiency:** There is no need to build a new communication infrastructure or to manage additional user accounts.
- **Connectivity:** The integration takes place within the university's verified and secured environment.
- **Availability:** It provides immediate access to notifications for all team members without the need to install additional third-party software.

Example from the Wazuh dashboard:

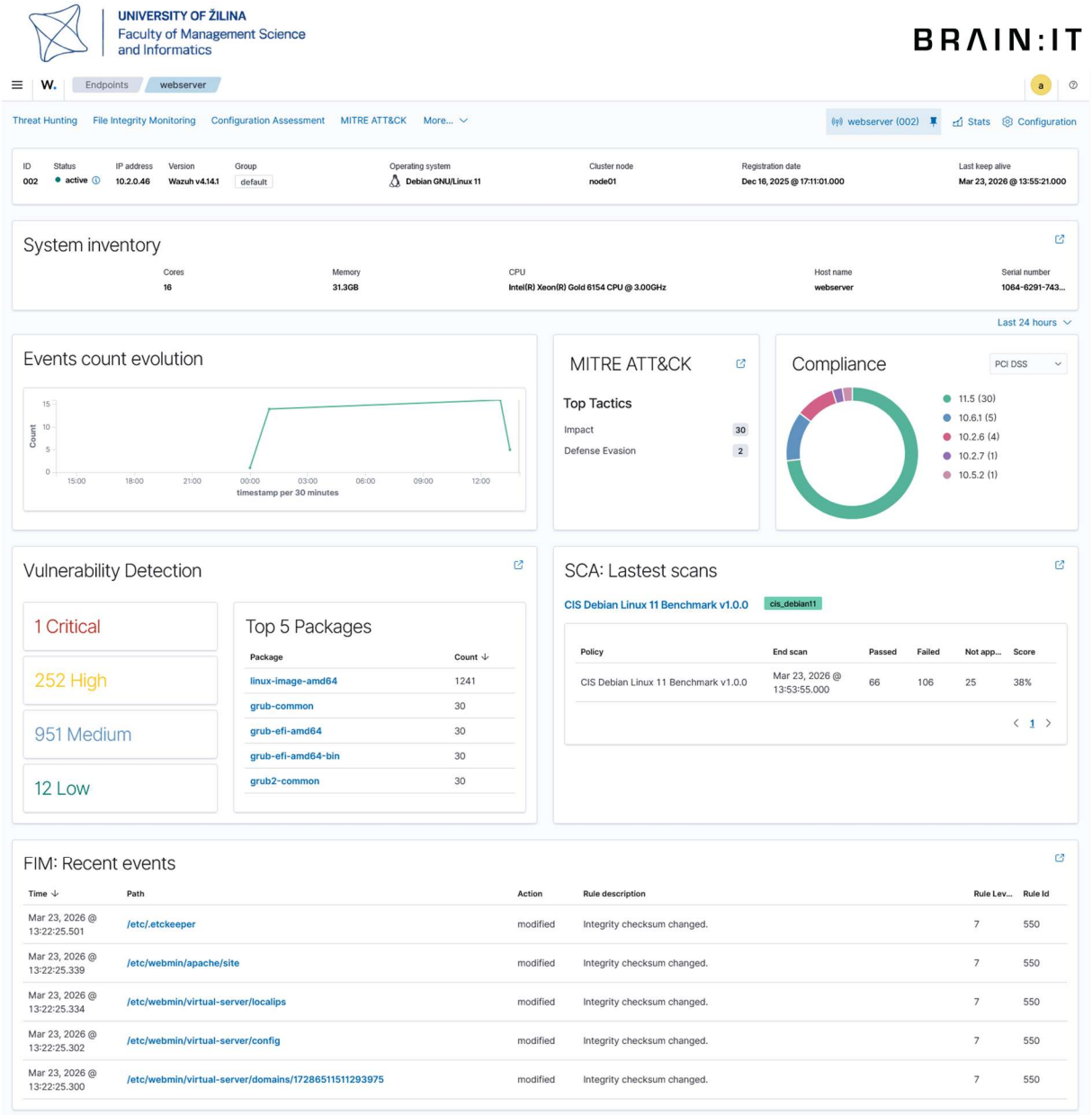


The interface shown complements the notification-sending integration scenario by providing an overview of the source endpoints from which security events are generated. The dashboard displays the status of the agents, their availability and basic characteristics that feed into the process of detection and subsequent sending of alerts.

The agent status overview makes it possible to identify whether all relevant endpoints are active and correctly connected, which is a prerequisite for reliable generation of notifications. Inactive or disconnected devices may represent a gap in detection coverage and thus also an absence of notifications for the security team.

The tabular overview of agents also provides contextual information that is also carried over into the notifications, such as agent identification, operating system or version. These data then form part of the alerts sent via the webhook to MS Teams and enable faster evaluation of the event.

The interface thus represents the input layer of the entire notification process, where the quality and availability of data from the agents directly affect the accuracy and reliability of the security alerts sent.



The figure shows a detailed view of a specific endpoint within the **Threat Hunting / Endpoint Detail** module in the Wazuh environment. The interface provides a consolidated overview of the device's telemetry, security events and configuration state.

The upper part shows the identification and operational metadata of the endpoint, including the IP address, agent version, operating system, cluster node and the timestamps of the last communication. This information defines the context of the data source and its availability within monitoring.

The *System inventory* section provides basic hardware and system parameters (CPU, memory, hostname) that can be used in correlating performance and security events.

The *Events count evolution* time series shows the trend in the number of events over time, which makes it possible to identify anomalies in system behaviour or sudden spikes in

Funded by the European Union NextGenerationEU
RECOVERY AND RESILIENCE PLAN
GOVERNMENT OFFICE OF THE SLOVAK REPUBLIC

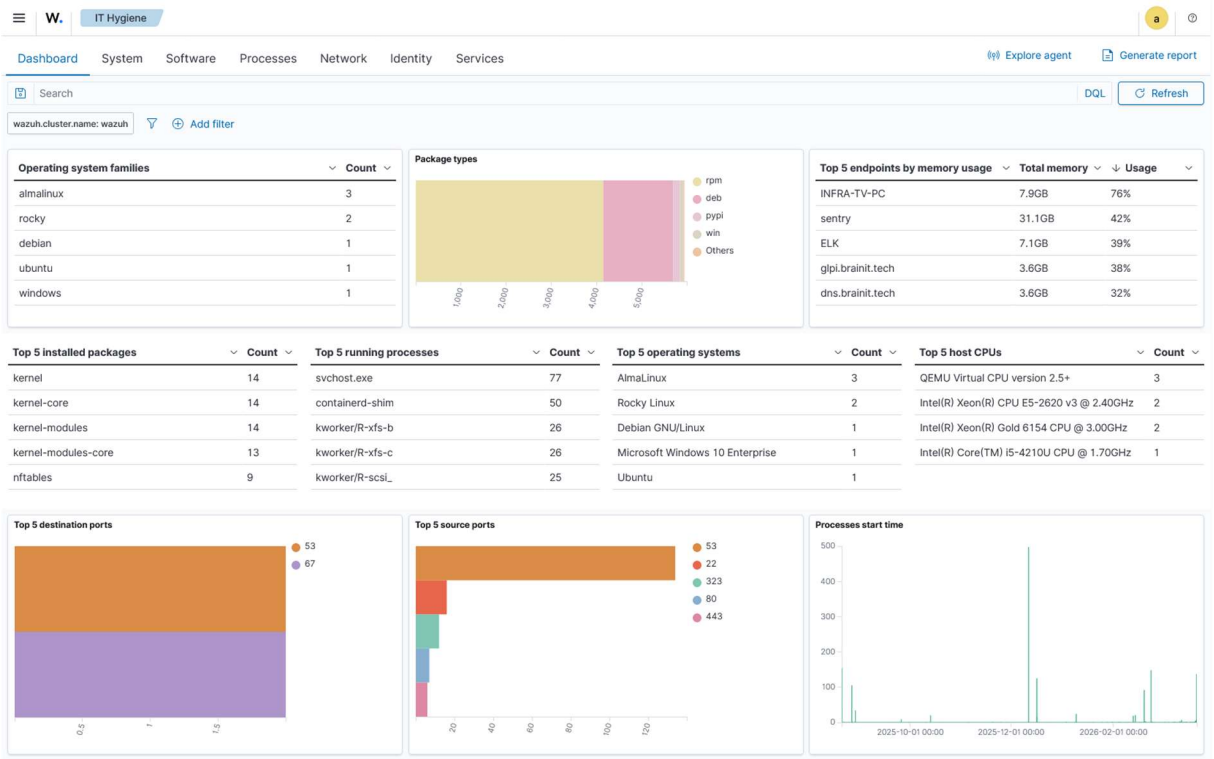
activity. The *MITRE ATT&CK* section maps detected activities to specific tactics (e.g. Defense Evasion), thereby providing a standardized framework for interpreting threats.

The *Compliance* module displays the status of the system's compliance with defined policies and benchmarks, while the *Vulnerability Detection* section quantifies the detected vulnerabilities by severity. The overview of the most frequently occurring packages makes it possible to identify potentially risky components in the system.

The *SCA (Security Configuration Assessment)* part provides the results of the most recent configuration checks according to defined standards (e.g. CIS benchmark), including the pass rate of individual checks and the overall score.

The bottom *FIM (File Integrity Monitoring)* section records recent changes in the file system, including modified files, the time of change and the assigned detection rules. These events are key to detecting unauthorized interventions in the system.

From an architectural point of view, this interface represents a correlation point over endpoint telemetry, where outputs from several detection mechanisms (FIM, vulnerability scanning, SCA, log analysis) are aggregated into a unified context of a specific device. This approach enables efficient triage, incident analysis and decision-making on response measures within the SOC.



The figure shows the dashboard of the **IT Hygiene** module in the Wazuh environment, which provides an aggregated view of the state of endpoints in terms of software and operational context. This module extends the basic telemetry with information usable for security analysis and vulnerability management.

The upper part of the dashboard contains summary metrics over the system inventory. It displays the distributions of operating systems, package types and memory usage on

individual endpoints. These data are derived from the collected agent telemetry and serve as input for exposure assessment and the identification of potentially risky systems (e.g. outdated OS or non-standard configurations).

The middle part provides a more detailed view of the software state of the endpoints, specifically:

- the most frequently installed packages (e.g. kernel components),
- actively running processes (e.g. container runtimes or system services),
- the distribution of endpoints by operating system,
- identification of the most heavily loaded systems in terms of CPU.

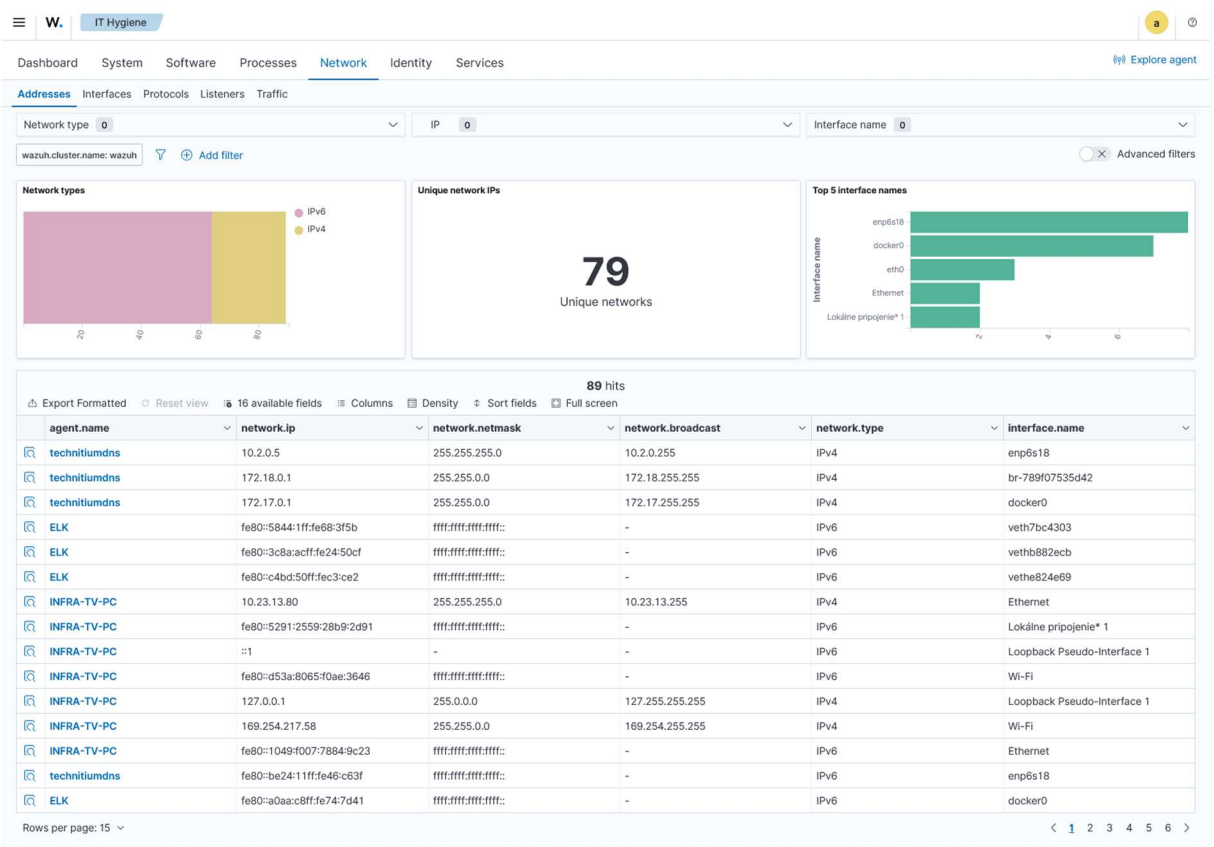
This information enables correlation between system behaviour and security events, for example the identification of anomalous processes or non-standard resource usage.

The bottom part of the dashboard focuses on the network and temporal context:

- an overview of the most frequently used source and destination ports,
- the temporal distribution of process launches.

These metrics provide a basis for detecting deviations in network communication or process behaviour, which is relevant, for example, in identifying lateral movement or unauthorized services.

From an architectural point of view, this dashboard represents an analytical layer over endpoint telemetry, where the data obtained from agents are aggregated and visualized. The outputs serve as a basis for detection rules, threat hunting and the prioritization of security measures within the SOC.



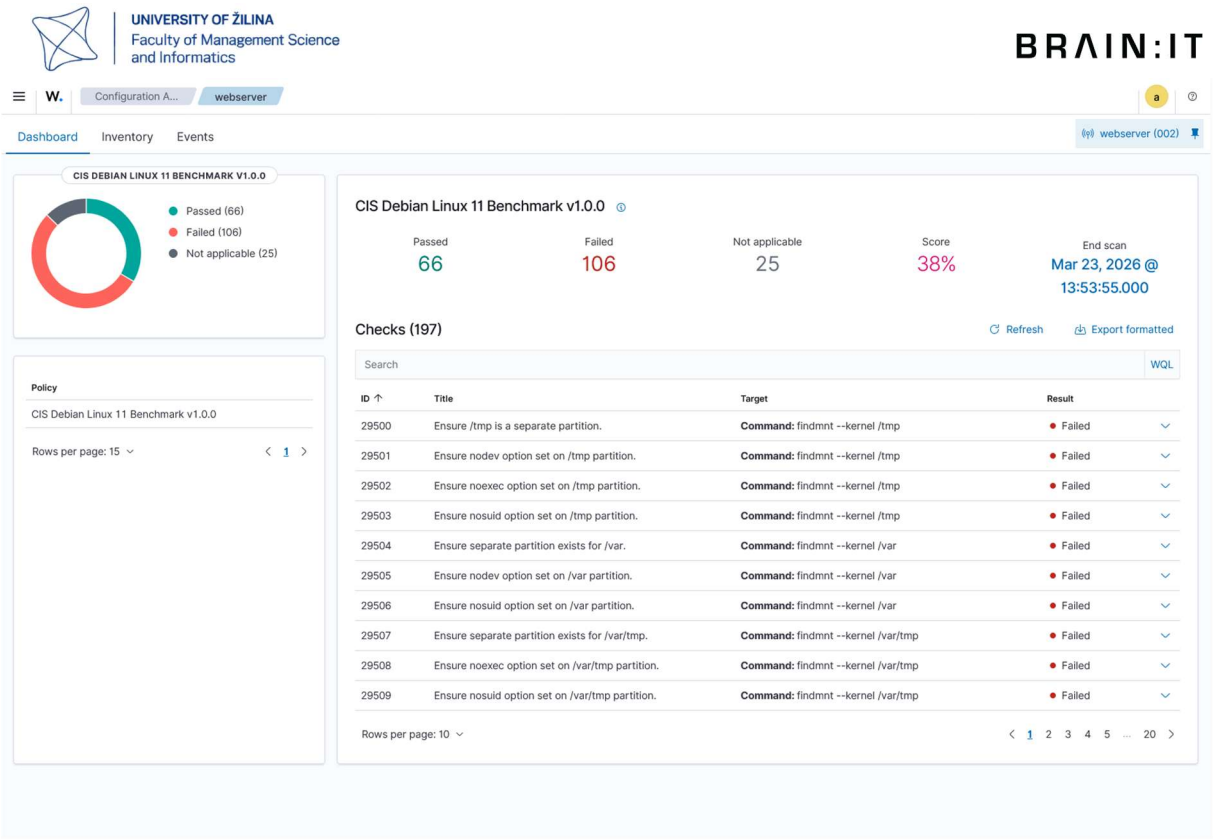
The figure shows the dashboard of the **Network** module within Wazuh IT Hygiene, which provides an overview of the network configuration of the monitored endpoints.

The upper part contains aggregated metrics over network properties, specifically the breakdown by network protocol type (IPv4/IPv6), the number of unique networks and the most frequently used network interfaces. These data enable rapid identification of the structure of the network environment and the distribution of interfaces across devices.

The lower part contains a detailed tabular overview of the network configurations of individual agents. For each record, attributes such as the IP address, netmask, broadcast address, network type and network interface name are listed. The data are collected through the agents and represent the current state of the endpoints' network configuration.

The interface enables filtering and analysis of network data, thereby supporting the identification of non-standard configurations, duplicate addresses or deviations in the network topology. From a security perspective, this information serves as input for detecting anomalies in communication and assessing the network exposure of devices.

Funded by the European Union
NextGenerationEU
RECOVERY AND RESILIENCE PLAN
GOVERNMENT OFFICE OF THE SLOVAK REPUBLIC



The figure shows the output of the **Security Configuration Assessment (SCA)** module in the Wazuh environment, specifically the assessment of an endpoint according to the **CIS Debian Linux 11 Benchmark v1.0.0** standard.

The upper part of the interface provides a summary overview of the configuration check results. It displays the number of successful (Passed), unsuccessful (Failed) and non-applicable (Not applicable) checks, together with the overall compliance score (Score). These metrics quantify the level of the system's security configuration against the defined benchmark and enable a rapid assessment of its security posture.

The main part contains a detailed list of the individual checks defined within the CIS benchmark. For each check, the identifier, a description of the requirement, the target component (e.g. a specific configuration file or system parameter) and the evaluation result are given. Most of the checks are in the "Failed" state, which indicates non-compliance of the system configuration with recommended security practices.

The checks focus mainly on operating system hardening, for example:

- the configuration of disk partitions (e.g. /tmp, /var),
- the setting of mount parameters,
- system security policies.

From an architectural point of view, this module represents a configuration assessment layer over endpoint telemetry, where system settings are compared against defined security standards. The outputs serve as a basis for identifying weaknesses, prioritizing corrective measures and demonstrating compliance with security requirements within the SOC or audit processes.

2.4 Greenbone Community edition

2.4.1 Role of Greenbone Community Edition in the PoC

Greenbone Community Edition is used within the PoC solution as a tool for vulnerability management and the identification of security weaknesses in the infrastructure. In the proposed SOC solution, it does not act as a real-time detection or response component, but represents a preventive layer that provides a systematic overview of the vulnerabilities of assets. Its importance lies mainly in the fact that it enables regular scanning of the infrastructure, the identification of weaknesses and the provision of supporting information for risk management and the planning of corrective measures.

Within the PoC, Greenbone is understood as a tool that complements the detection and analytical components (e.g. SIEM, EDR) with preventive security context. The scan outputs make it possible to identify vulnerable systems, assign them to specific assets and support decision-making on remediation priorities. Such an approach extends the SOC's perspective from reactive event processing to proactive security management.

Greenbone in the SOC provides a view of the vulnerability level of the environment over time. In this sense, it represents a complementary yet important component of the solution, which supports the prevention of security incidents and creates a basis for further integration scenarios, for example connection to ticketing systems or risk management tools.

2.4.2 Deployment and basic configuration of Greenbone

Within the PoC solution, Greenbone Community Edition was deployed as a standalone service intended for vulnerability scanning and results management. From an implementation perspective, the solution is based on a modular architecture comprising a scanning component (OpenVAS Scanner), a management layer (gvmd) and database storage (PostgreSQL). Such an approach allows flexible deployment and scaling according to the scope of the infrastructure.

The basic configuration focused on making the web interface (GSA) accessible and on setting up basic scan profiles, targets and scan schedules. The system was prepared to use the updated vulnerability feed (NVT), which ensures that the detected weaknesses are up to date.

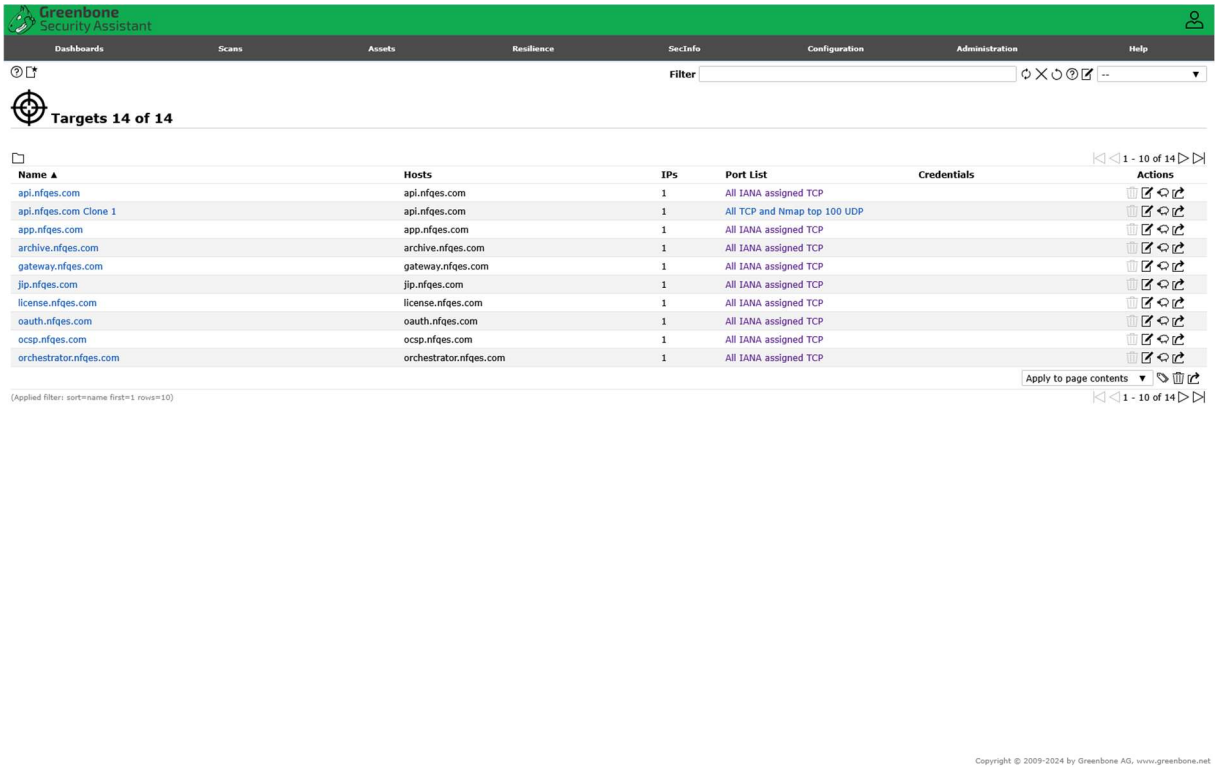
Within the PoC, basic scanned ranges were defined and scenarios of scheduled and manual scanning were tested. The results are stored in the database and made accessible via the web interface, where they can be filtered, analysed and exported.

2.4.3 Use of Greenbone within the PoC

Within the PoC, Greenbone was used as a tool for identifying vulnerabilities and providing an overview of the security state of the infrastructure. Its main contribution lay in the fact that it made it possible to systematically uncover weaknesses in systems and to provide supporting information for their subsequent remediation.

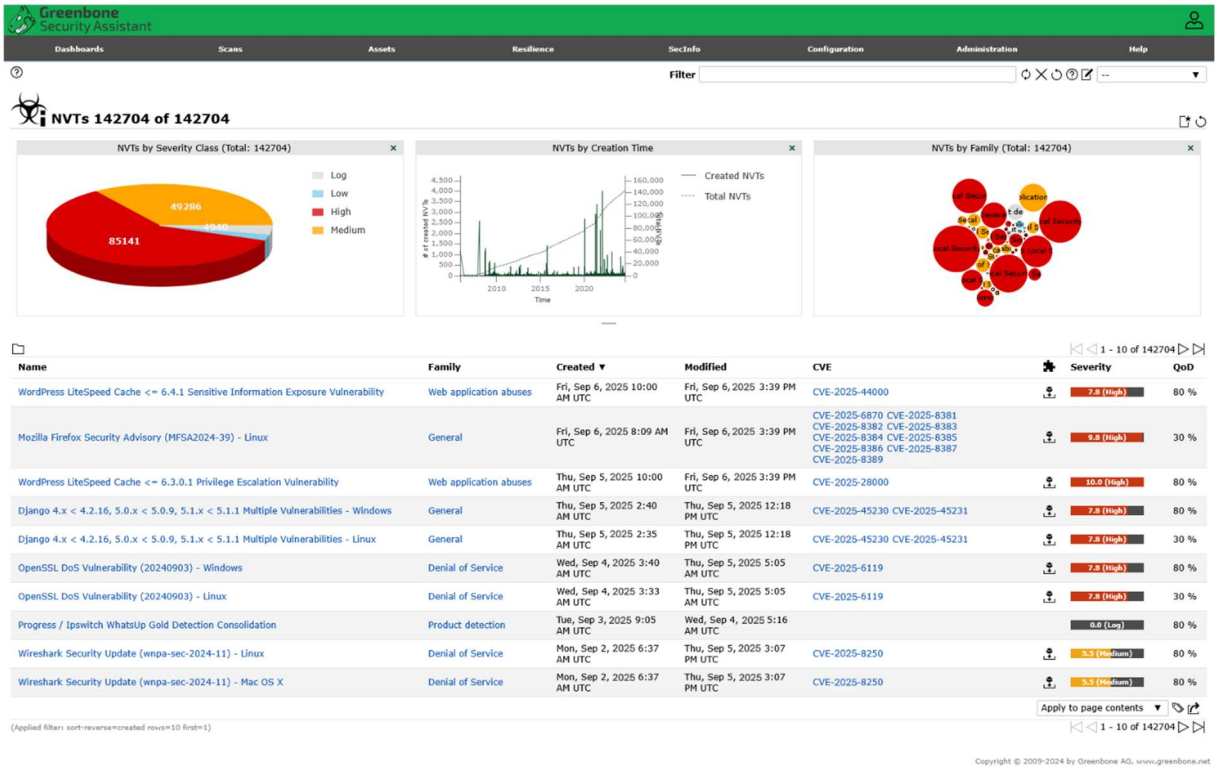
Practical use at this stage was oriented mainly towards performing basic scans, analysing the results and identifying critical vulnerabilities. Greenbone thus did not perform active detection or response, but complemented the solution with a preventive view of the security of the environment.

To this extent, its deployment confirmed that it is a suitable complement to the PoC solution and a usable basis for further extension, particularly in the area of integration with vulnerability and incident management processes.



The figure shows the **Targets** section in Greenbone Security Assistant, where the target systems for vulnerability scanning are defined.

The table contains a list of domains (hosts), their IP addresses, the range of scanned ports and the available management actions. Each record represents one scan target, which determines which systems will be subject to security scanning and to what extent.



The figure shows the web interface of **Greenbone Security Assistant (GSA)**, specifically an overview of the **NVT (Network Vulnerability Tests)** database, which are the individual detection tests used in vulnerability scanning.

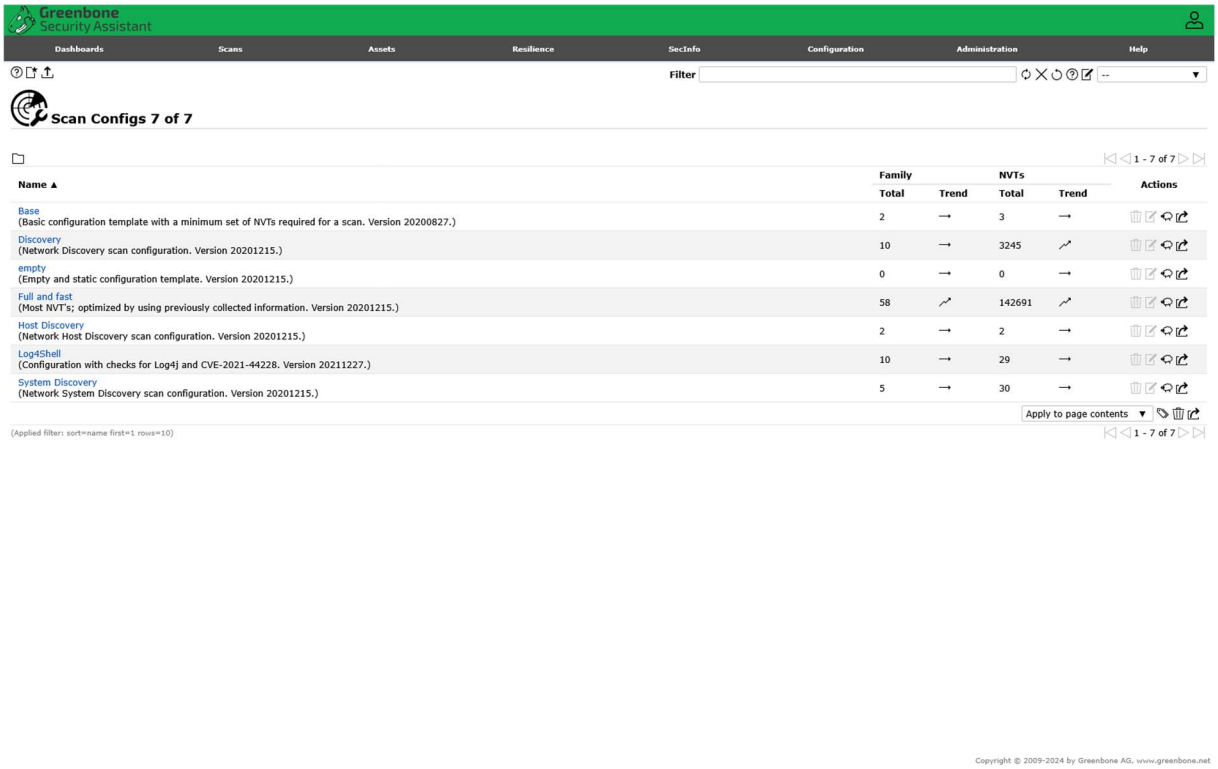
The upper part contains visualizations providing an aggregated view of the vulnerability database:

- a pie chart of the distribution of NVTs by severity (Low, Medium, High, Log),
- the trend in the creation of NVT tests over time,
- the breakdown of NVTs by category (family).

The central part contains a tabular overview of specific vulnerabilities, where each record represents one detection test. For each test, the following are given:

- the name and type of the vulnerability,
- the category (family),
- the date of creation and of last modification,
- the link to standardized identifiers (CVE),
- the severity rating (CVSS score),
- QoD (Quality of Detection), i.e. the reliability of detection.

The interface enables filtering, sorting and detailed analysis of individual vulnerabilities, thereby providing analysts with an overview of the coverage of known threats within the scanning engine.



The figure shows the **Scan Configs** section in Greenbone Security Assistant, which is used to manage configuration profiles for vulnerability scanning.

The table contains a list of the available scan configurations (e.g. *Full and fast*, *Discovery*, *Host Discovery*, *System Discovery*), with each configuration defining the scope and type of the tests (NVTs) performed. For the individual configurations, the following are given:

- a brief description of the purpose of the scan,
- the number of test categories (Family),
- the total number of NVT tests used,
- the trend (an indication of changes or updates),
- actions for managing the configuration.

The configurations differ mainly in scope – from basic or quick scans (Discovery) to comprehensive scanning (Full and fast), which uses an extensive database of tests. This section makes it possible to adapt scanning to the required depth of analysis, performance demands and the specific purpose of the security assessment.

2.4.4 Current status and limitations of Greenbone in the SOC

Within the PoC, Greenbone Community Edition was successfully deployed and given a basic configuration as a functional tool for vulnerability scanning and results management. In this state, it provides a usable basis for identifying security weaknesses and supporting vulnerability management processes. From the PoC perspective, it was confirmed that it is a suitable component for complementing the SOC solution with preventive security context.

It should also be pointed out that NVT feeds and scanning largely work only on the basis of correlating package versions, which often results in distorted scan results, for example for backported packages that do carry a vulnerable version number but to which security patches have already been retroactively applied for this older version; in practice, this means

that a vulnerability found by the scan is merely heuristic and does not exist in real-world practice.

On the other hand, it should be noted that its use in the PoC remained at the level of basic scanning and manual analysis of the results. Not all of the planned integration links were fully implemented, for example automated connection to SIEM, SOAR or ticketing systems, and so at this stage the tool does not function as a fully integrated source for vulnerability management.

For this reason, Greenbone in the PoC should be understood as a functional, but completely separate, standalone component. Nevertheless, its deployment demonstrated its practical usability and confirmed the suitability of including it in the wider architecture of the SOC service.

2.5 OpenCTI

OpenCTI is used within the INNOSOC PoC solution as the central threat repository, where indicators of compromise (IoC), TTP techniques, campaigns and actors are collected. The platform makes it possible to analyse these data and subsequently distribute them to other security tools without the need for manual intervention.

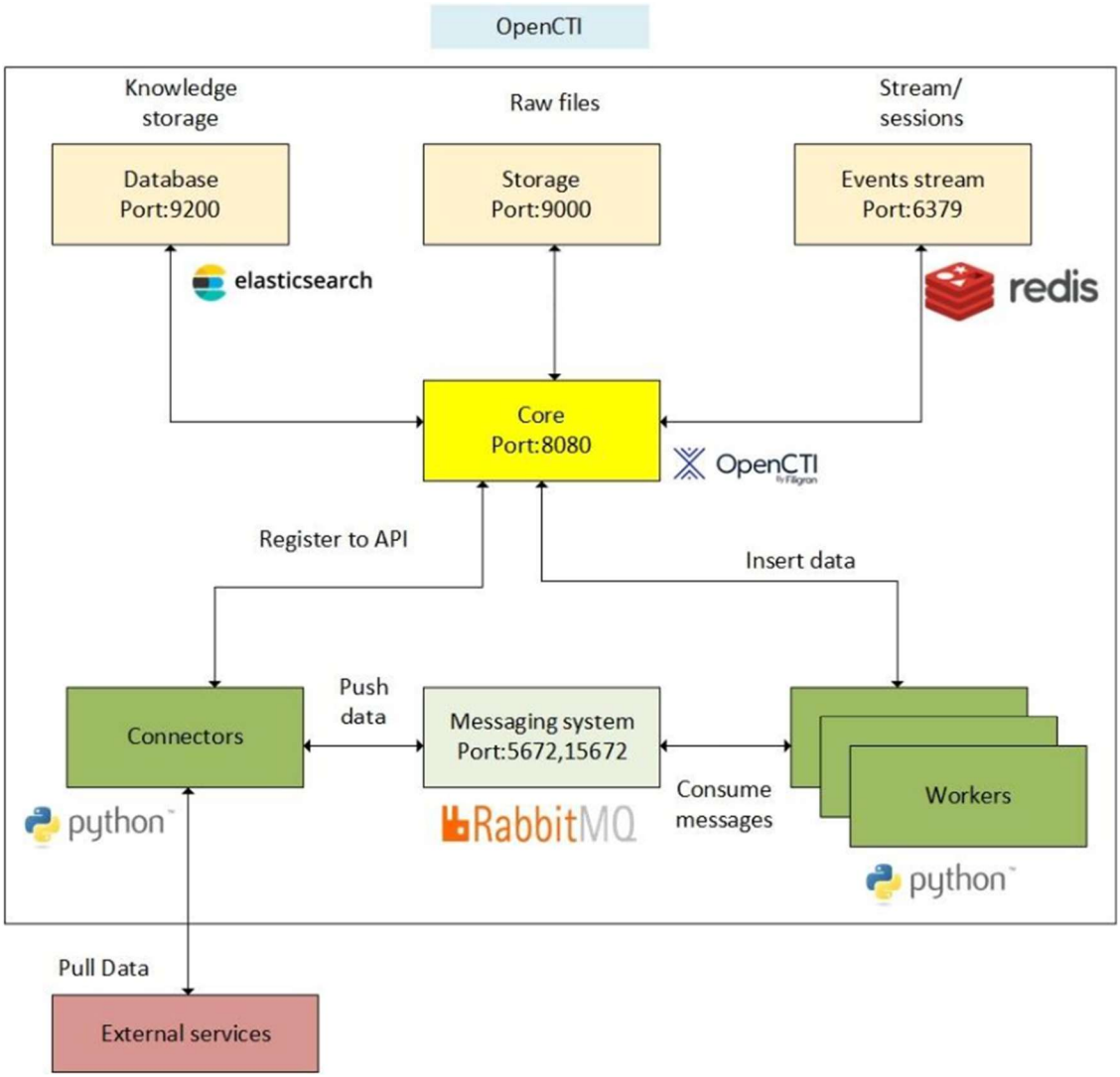
OpenCTI (Open Cyber Threat Intelligence Platform) is an open-source platform designed for managing, analysing and sharing cyber threat information (CTI). It makes it possible to aggregate data from various sources, correlate them and transform them into a unified model based on the STIX standard.

2.5.1 Deployment of OpenCTI in the OpenStack environment

Technical specification of the OpenCTI deployment:

- Environment – OpenStack
- Instance name – OpenCTI_s
- IP address – 192.168.4.121
- vCPU – 4
- RAM – 16 GB
- Disk – 80 GB
- OS – Ubuntu 22.04

The diagram in the figure illustrates the architecture of the OpenCTI platform on the server and the data processing flow between the individual components. The central element is OpenCTI Core running on port 8080, which provides the application logic and communication between services. Core communicates with the Elasticsearch database on port 9200, where all CTI data are stored, as well as with the file storage on port 9000, which is used to store raw data and attachments. Redis on port 6379 is used to manage events and the cache.



The runtime environment contains the RabbitMQ messaging system (ports 5672 and 15672), which provides asynchronous communication between components. Connectors retrieve data from external sources, register via the API with Core and then send the data to RabbitMQ. Workers process these messages, perform data transformation and enrichment, and then store them back in OpenCTI via Core.

2.5.2 Basic integration links for the OpenCTI component

Services currently connected and tested using OpenCTI:

- **Abuse.ch (ThreatFox, URLhaus, Malwarebazzar)** – automatic import of IoCs
- **Ransomwarelive** – import of strategic CTI
- **MITRE ATT&CK** – mapping of techniques and tactics
- **SOAR (Shuffle)** – automation of incident response

2.5.3 Note on the MISP platform

In the original architectural design, the MISP (Malware Information Sharing Platform) was also considered as a separate component for collecting and sharing threat information (Threat Intelligence). After evaluating the current capabilities and the scope of the PoC, it was decided that the collection, management and distribution of CTI data within the PoC would be handled exclusively through the OpenCTI platform.

OpenCTI natively supports integration with MISP as one of its external data sources via the MISP connector, which enables the automatic import of indicators of compromise (IoC) and other structured data from MISP instances. However, this integration was neither activated nor tested within the PoC, since a direct connection of OpenCTI to selected public threat sources (Abuse.ch, MITRE ATT&CK, Ransomwarelive) is sufficient for the proof of concept.

Connecting MISP as an external threat database remains an option for future extension of the solution in a production environment, where it could provide access to a wider CTI data-sharing community and to the organization's internal databases.

2.6 GLPI

2.6.1 Role of GLPI in the PoC

GLPI is used within the INNOSOC PoC solution as the central tool for IT asset inventory, configuration management (CMDB) and IT service management (ITSM). In the proposed SOC solution, it does not act as a detection or response component, but represents a process and record-keeping layer that links technical security events with the organizational context, asset owners and the services provided. Its importance lies mainly in the fact that it makes it possible to systematically record assets, their relationships and responsibilities, while also ensuring the management of incidents, requests and changes through a unified ticketing system.

Within the PoC, GLPI is understood as a tool that complements the technical data obtained from the detection and analytical components (e.g. SIEM, EDR) with process and organizational context. When processing security events, it thus makes it possible to assign an incident to a specific asset, identify its owner, determine responsibilities and track the entire handling life cycle from reporting through to closure. Such an approach increases the transparency, auditability and consistency of security incident handling.

GLPI in the SOC provides a process framework within which security events are formalized and managed. In this sense, it represents a complementary yet key component of the solution, which ensures the linking of the technical and organizational levels and creates the prerequisite for the effective management of security processes and further integration extensions.

2.6.2 Deployment and basic configuration of GLPI

Within the INNOSOC PoC solution, the GLPI tool was deployed as a standalone web application intended for IT asset inventory and the management of service processes. From an implementation perspective, the system was operated on a standard web stack (Apache/Nginx, PHP) using a relational database (MySQL/MariaDB), which corresponds to its

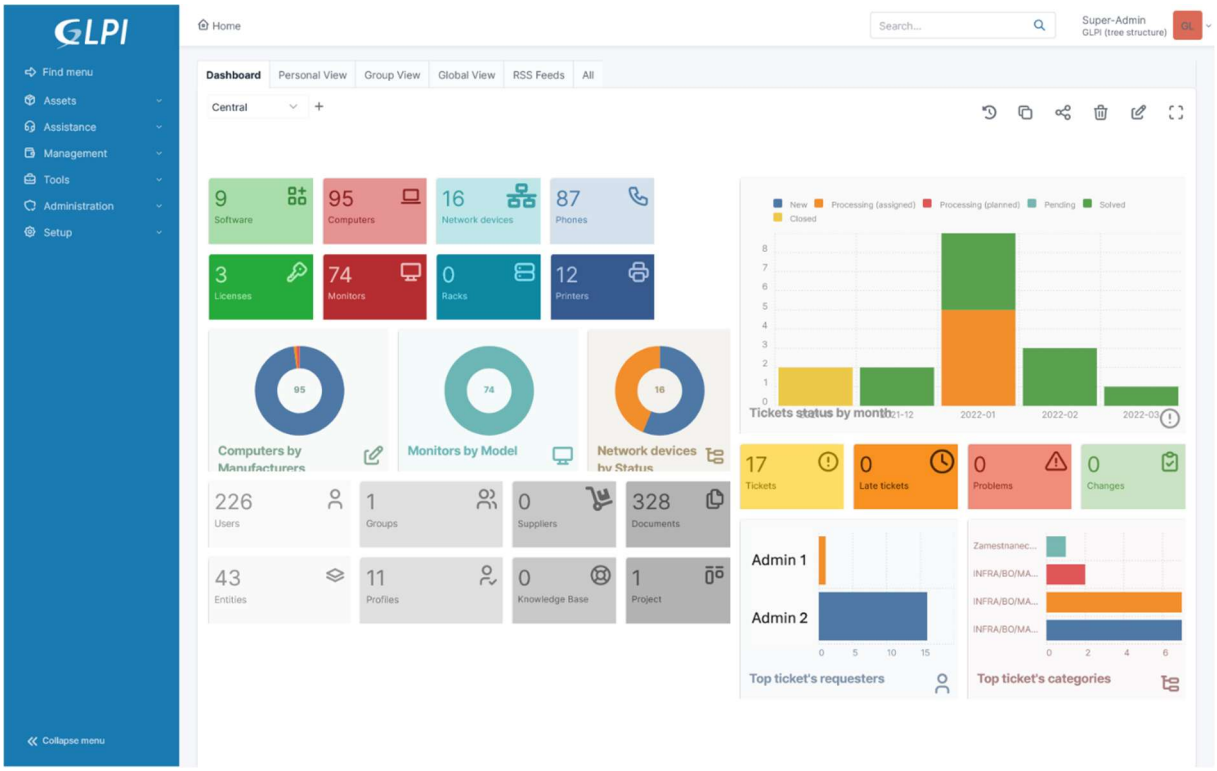
multi-tier architecture and allows both simple deployment and scaling. Such an implementation approach is suitable for both test and production environments and at the same time enables integration with other SOC components.

In the PoC, the basic configuration of GLPI focused on making the web interface accessible and activating key functionalities, in particular asset inventory (CMDB) and the ticketing system. The system was configured to be ready for managing users, roles and relationships between assets, with the database layer ensuring the storage of operational and historical data. Within the PoC, basic objects such as IT assets, users, organizational units and incident records were entered into the system, enabling technical events to be linked with the process and organizational context.

2.6.3 Use of GLPI within the PoC

Within the PoC, GLPI was used as a process and record-keeping layer for the management of incidents and IT assets in the context of the SOC solution. Its main contribution lay in the fact that it made it possible to assign technical security events to specific assets, users and organizational units, while also ensuring their formal processing through the ticketing system.

Practical use of GLPI at this stage was oriented mainly towards asset inventory, management of basic relationships in the CMDB and tracking the incident life cycle. GLPI thus did not act as a detection or response tool, but complemented the solution with the process and organizational context needed for incident management. To this extent, its deployment confirmed that it is a suitable complement to the PoC solution and a usable basis for further extension of integration and automation scenarios.

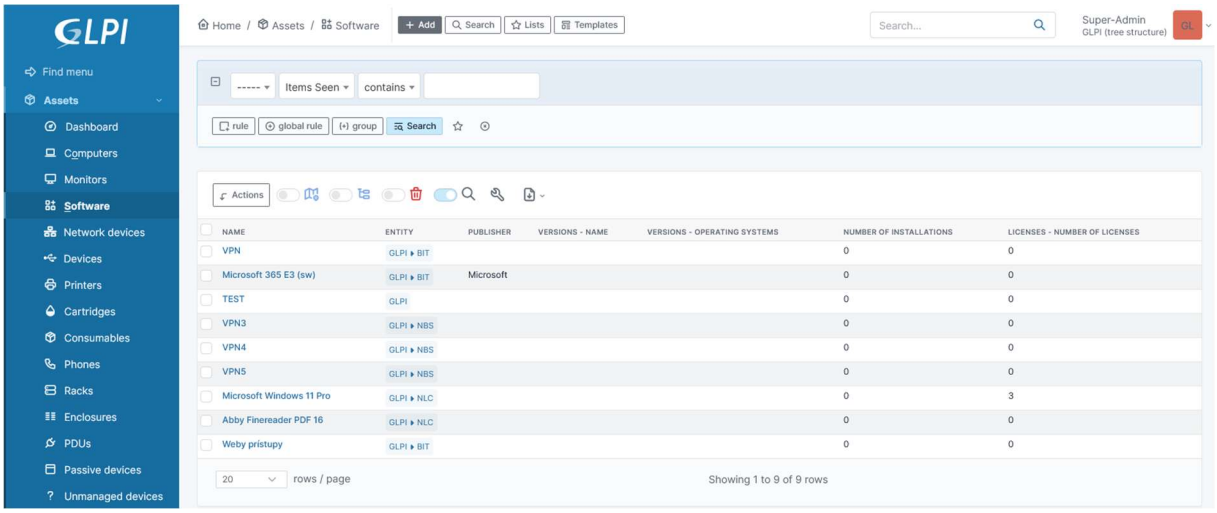


The figure shows the GLPI dashboard, which provides an overview of the registered IT assets and the status of ITSM processes.

The upper part summarizes the asset categories (e.g. computers, network devices, software), which makes the role of GLPI as a CMDB system visible. The middle part contains infrastructure statistics and overviews, supplemented by information on users and organizational entities.

On the right-hand side is the ticketing section, which displays the status of incidents and requests (new, in progress, closed), including basic metrics.

From the perspective of the SOC architecture, GLPI represents a process and record-keeping layer that links technical events with assets and ensures controlled and auditable incident handling.



The figure shows the GLPI interface in the software management section (Assets → Software), which is used to record and manage the applications installed in the organization's environment.

The central part contains a tabular overview of software items (e.g. VPN, Microsoft 365, Windows 11), with each record containing:

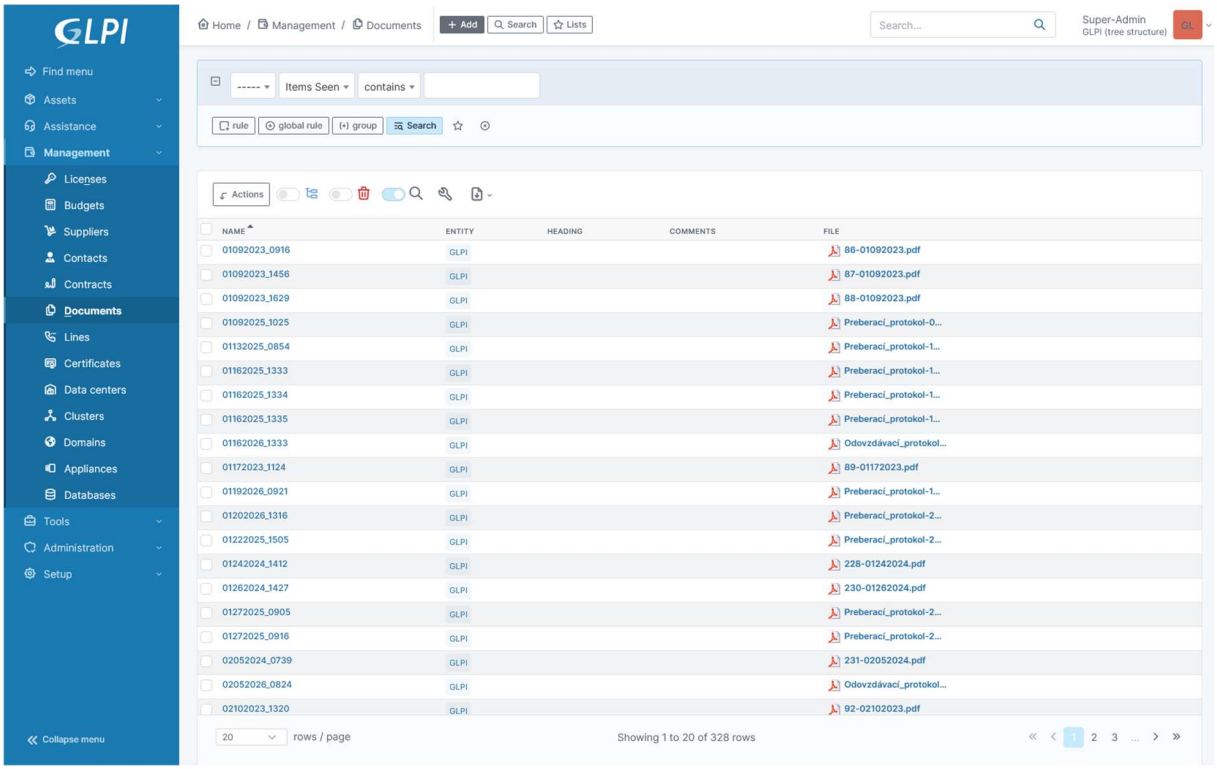
- the software name,
- the organizational entity (Entity),
- the publisher (Publisher),
- versions and supported operating systems,
- the number of installations,
- licence information.

The upper part of the interface enables filtering and searching of records, while the side panel provides navigation between the various asset types (computers, networks, devices, etc.).

From the perspective of the SOC architecture, this section represents the inventory of software assets and licences, which is key for:

- identifying vulnerable or outdated software,

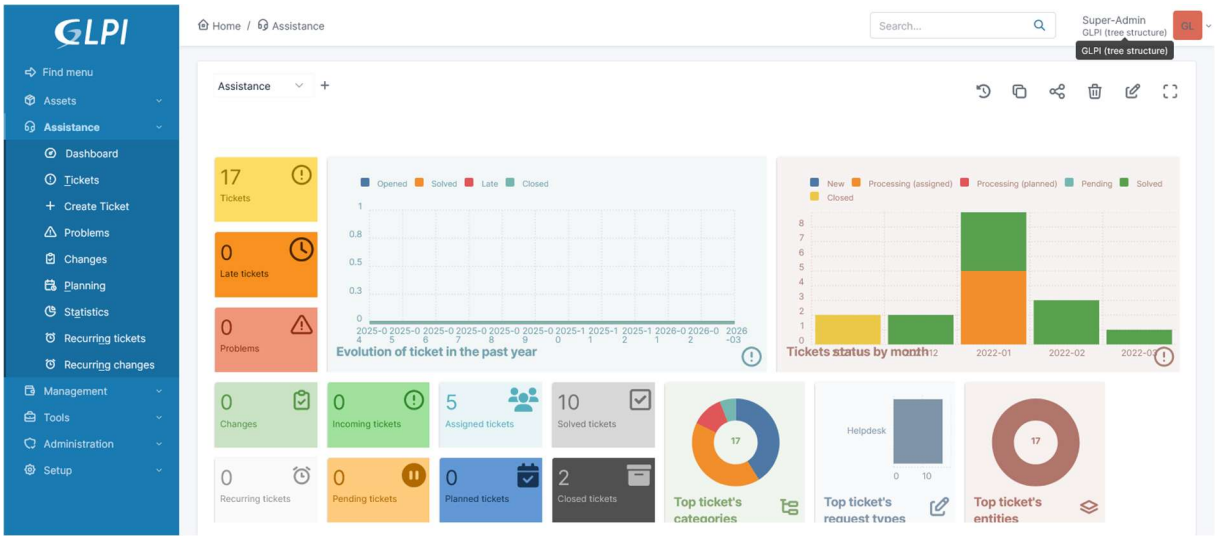
- linking to specific devices and owners,
- supporting vulnerability and incident management in the context of specific applications.



The figure shows the **Documents section in GLPI**, which is used to manage and record documentation within the IT environment.

The central part contains a tabular overview of documents with identifiers, the assigned entity and links to files (most often PDF). The interface enables filtering, searching and basic operations on documents.

This functionality provides **centralized storage and traceability of documents** such as reports, records or contractual documentation, which is important from the perspective of auditing and process management in the SOC environment.



The figure shows the **dashboard of the Assistance module in GLPI**, which is used for the operational management and monitoring of tickets (incidents and requests) within ITSM.

On the left-hand side is a navigation menu with sections such as Dashboard, Tickets, Problems, Changes and Planning, with the Assistance area active. The central part of the dashboard provides summary ticket metrics through colour-coded panels – for example, the total number of tickets (17) and the number of late tickets, problems and changes. These metrics provide a quick overview of the current status of service requests.

The dashboard also contains visualization elements:

- a time chart showing the trend in tickets over the monitored period,
- a bar chart representing the status of tickets by month (new, in progress, solved, etc.),
- additional statistics such as the number of assigned, solved or pending tickets.

The bottom part contains aggregated views of ticket categories, request types and organizational entities, which makes it possible to identify the most heavily loaded areas or the most frequent types of incidents.

From the perspective of the SOC architecture, this dashboard represents the **process and management layer of incident management**, where security events are transformed into ticket records, their life cycle is tracked and the effectiveness of their handling is evaluated.

2.6.4 Current status and limitations of GLPI in the SoC

Within the PoC, GLPI was successfully deployed and given a basic configuration as a functional tool for IT asset inventory and incident management. In this state, it provides a usable basis for managing CMDB data, recording relationships between assets and supporting ticketing processes. From the PoC perspective, it was confirmed that GLPI is suitable for complementing the SOC solution with process and organizational context and for supporting security incident management.

On the other hand, it should be noted that its use in the PoC remained at the level of basic record-keeping and ticket processing. Not all of the planned integration links to the other components of the solution were fully implemented (e.g. automated connection to SIEM/SOAR), and so at this stage GLPI does not function as a fully integrated incident

management tool with automatic data enrichment. For the time being, its role is rather a supporting one, laying the groundwork for further extension in future stages of the project.

For this reason, GLPI in the PoC should be understood as a functional, but not yet fully developed, component. Nevertheless, its deployment demonstrated its practical usability in the proposed solution and confirmed the suitability of including it in the wider architecture of the SOC service.

2.7 NetBox

2.7.1 Role of NetBox in the PoC

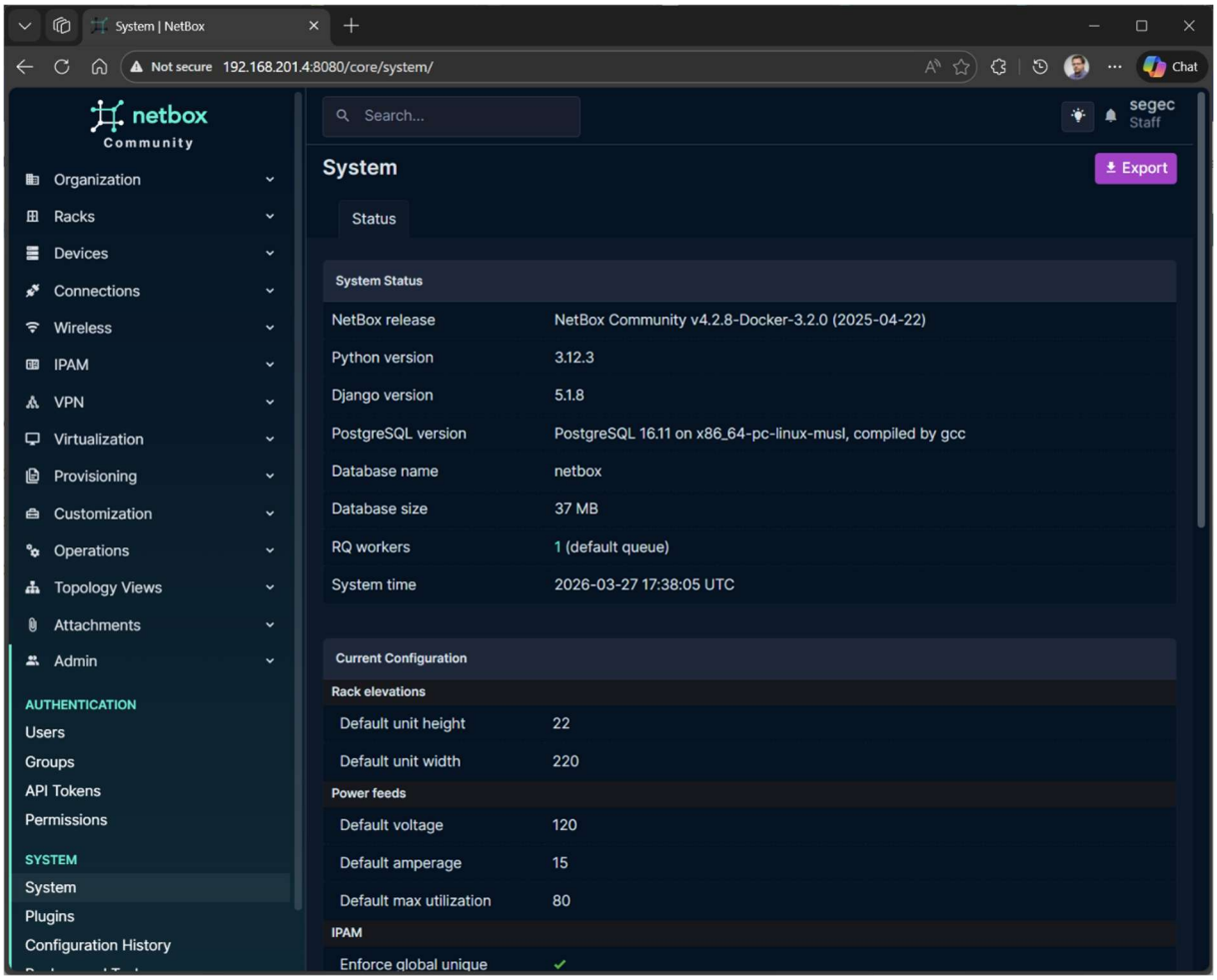
NetBox is used within the INNOSOC PoC solution as a supporting tool for recording and managing infrastructure context. In the proposed SOC solution, it does not act as a detection or response component, but serves as a source of information on assets, addressing, network segments and selected relationships between infrastructure elements. Its importance lies mainly in the fact that it makes it easier to understand which device, system or part of the network a particular event relates to. Within the PoC, NetBox is understood as a context layer that complements the technical data obtained from the other components of the solution. When analysing security events, it can thus be used to look up basic information about devices, IP ranges, sites or the logical placement of infrastructure elements. Such an approach increases the clarity of the solution and creates the prerequisite for a more accurate assessment of security situations.

NetBox in the SOC offers the context of the environment in which security logs and alerts originate. In this sense, it represents a ***complementary*** but important component of the solution, which supports orientation within the infrastructure and lays the groundwork for further integration use in extended scenarios.

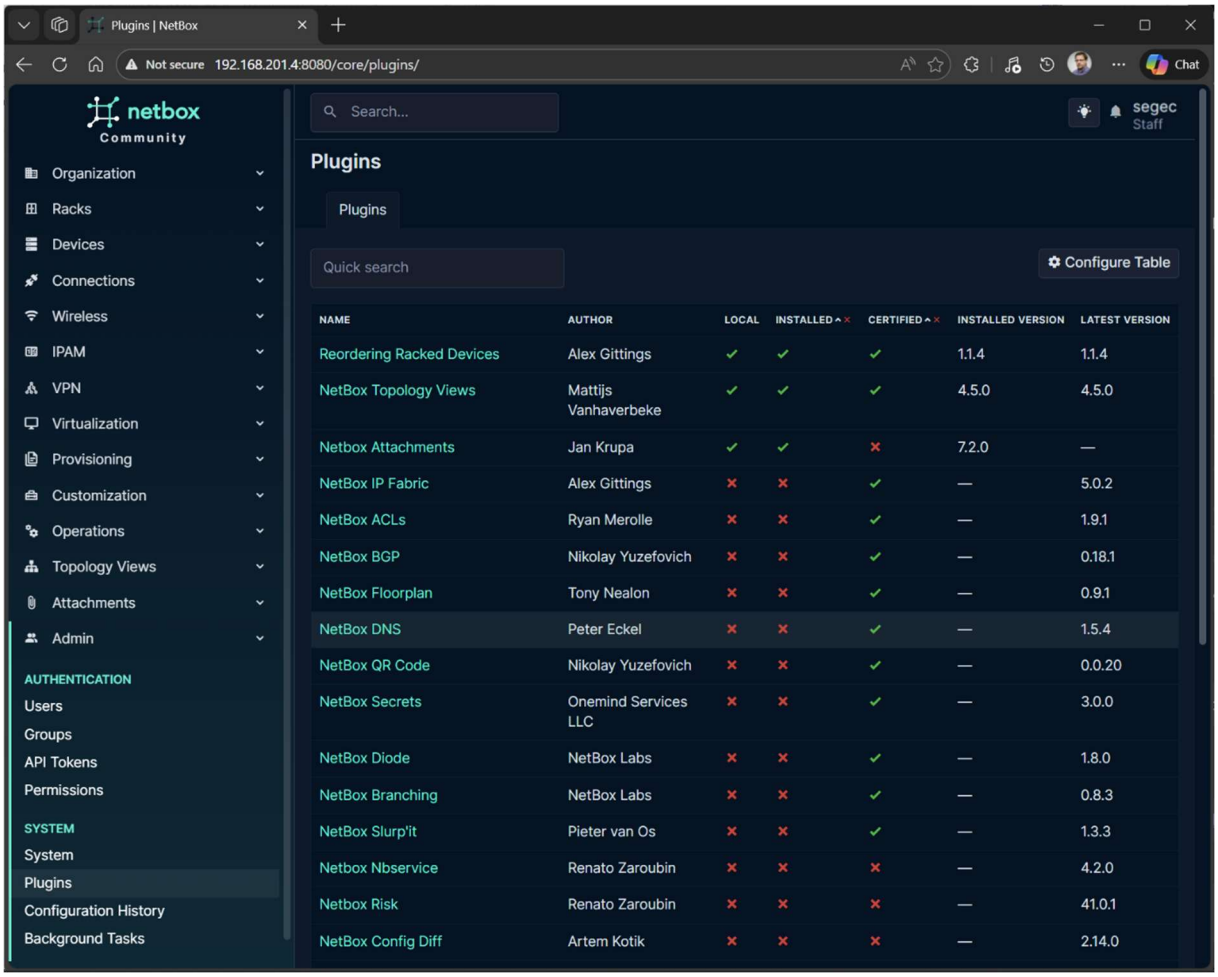
2.7.2 Deployment and basic configuration of NetBox

Within the PoC, NetBox was deployed in the OpenStack environment as a standalone service intended for recording and managing infrastructure data. From an implementation perspective, it was containerized, using the application and supporting components required for it to run, in particular the database layer and cache/queue services. This deployment approach corresponds to the nature of an experimental deployment and at the same time allows easier administration, updating and repeatable deployment.

In the PoC, the basic configuration of NetBox focused on making the web interface accessible and on preparing the data model for recording selected infrastructure objects. In terms of configuration, the system was prepared to work with a PostgreSQL database and with a separate Redis layer for tasks and cache. Within the PoC, basic objects were entered into the system, in particular sites, devices, address spaces and the related infrastructure context.



System information



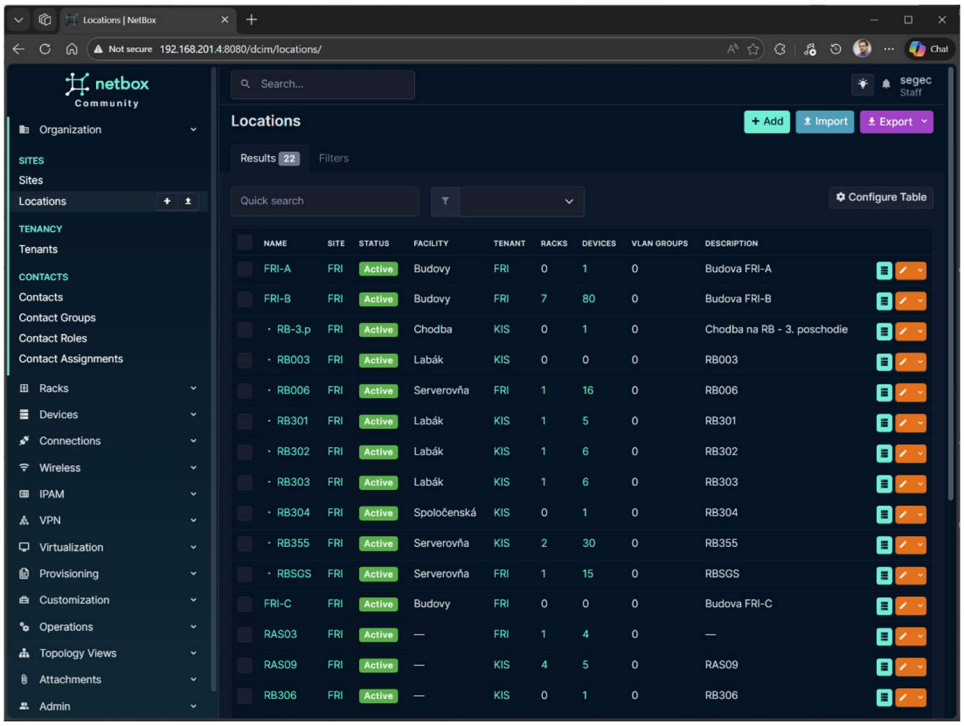
Plugins

The aim was not to create a complete production CMDB or a detailed model of the entire network, but to prepare a representative data set on which the use of NetBox in the proposed solution can be demonstrated.

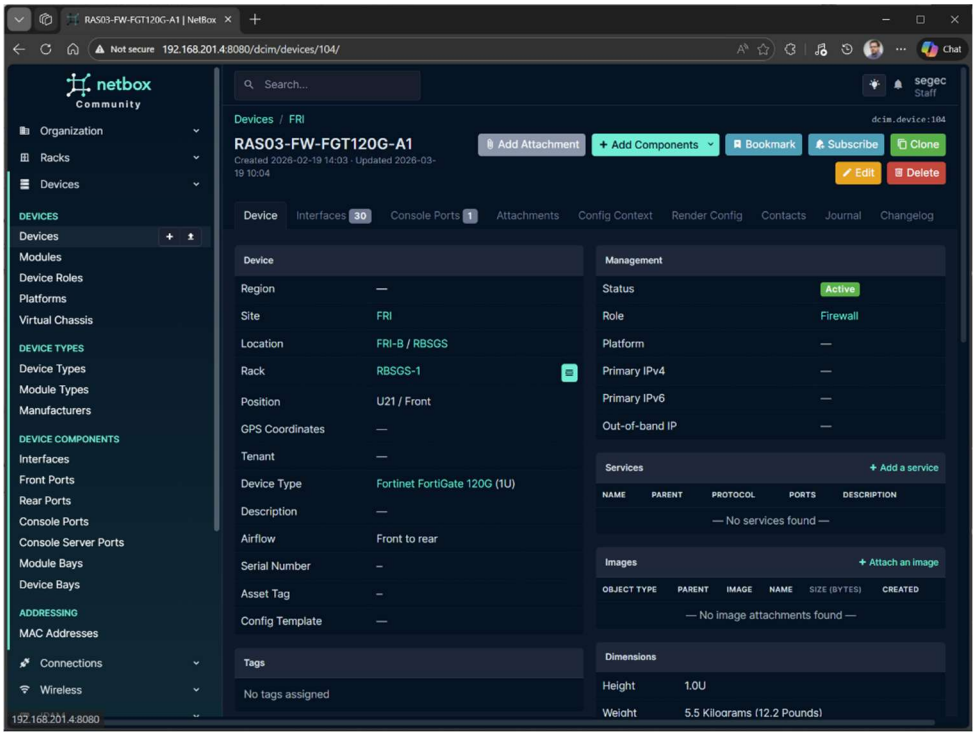
2.7.3 Use of NetBox within the PoC

Within the PoC, NetBox was used as a supporting source of infrastructure context for the other parts of the solution. Its main contribution lay in the fact that it made it possible to look up basic information about devices, IP addressing, sites and network segments to which events or findings from other tools could be assigned.

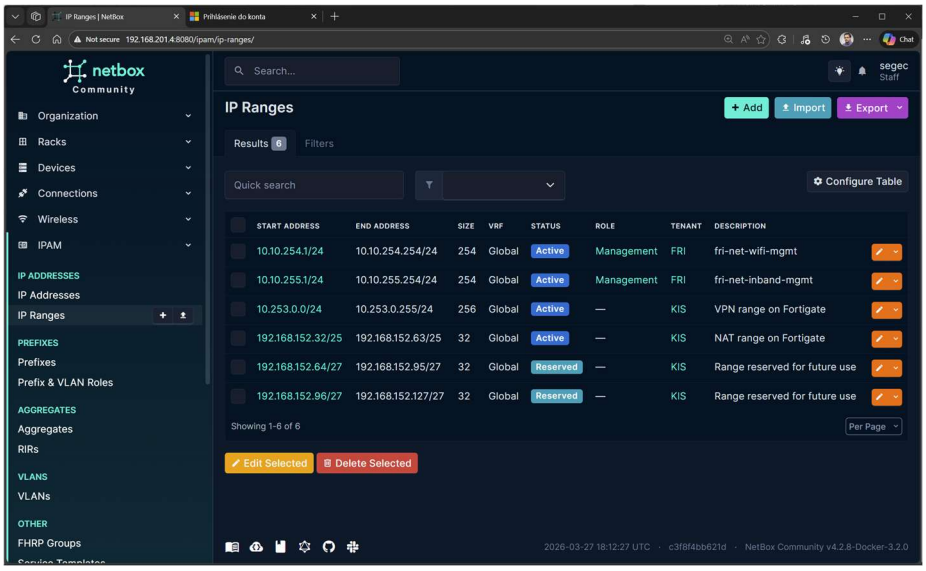
Practical use of NetBox at this stage was oriented mainly towards record-keeping and manual or partially assisted context lookup. NetBox thus did not act as a detection or response tool, but complemented the solution with the data needed for a better understanding of the technical environment. To this extent, its deployment confirmed that it is a suitable complement to the PoC solution and a usable basis for further extension of integration scenarios.



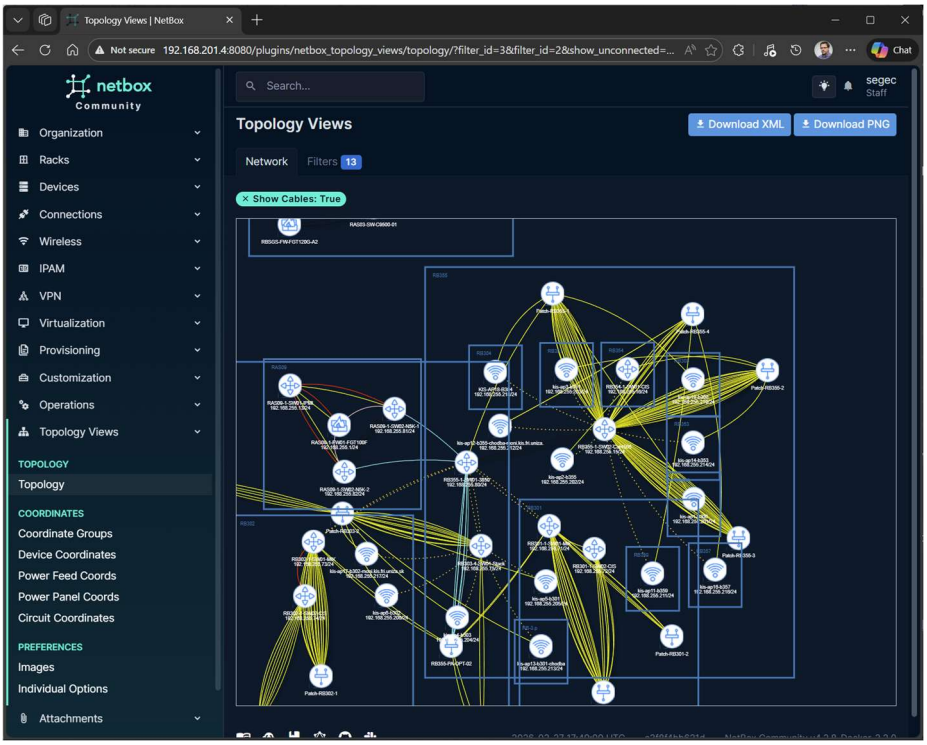
Modelling the organization

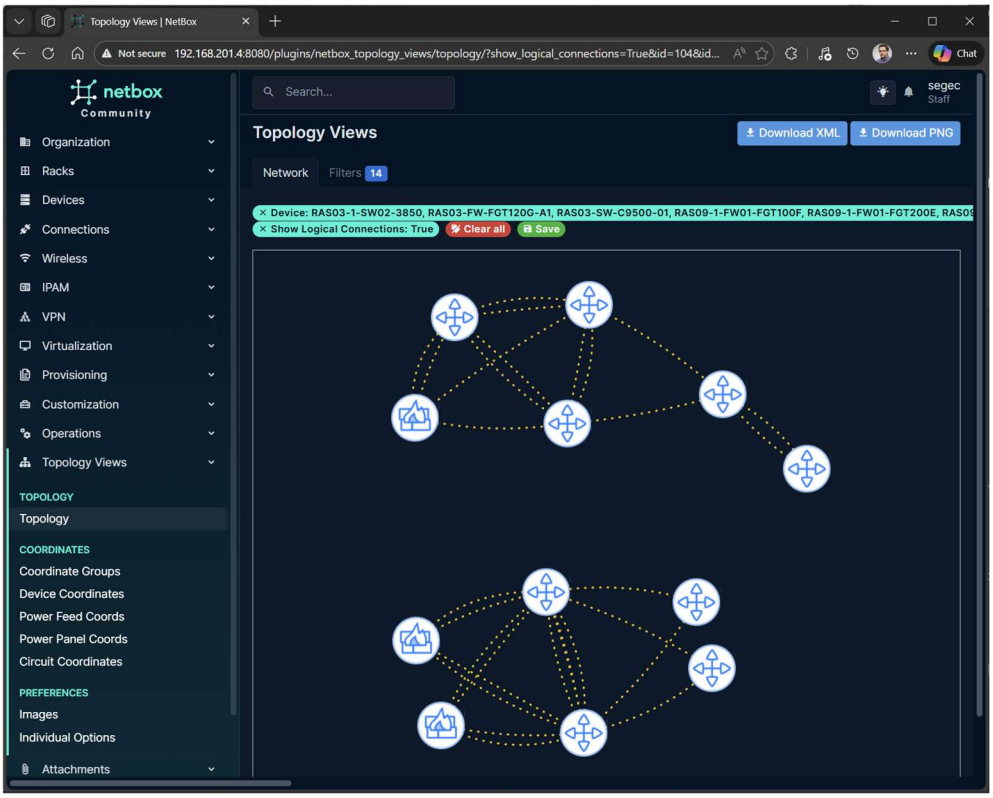


Modelling a device – the firewall

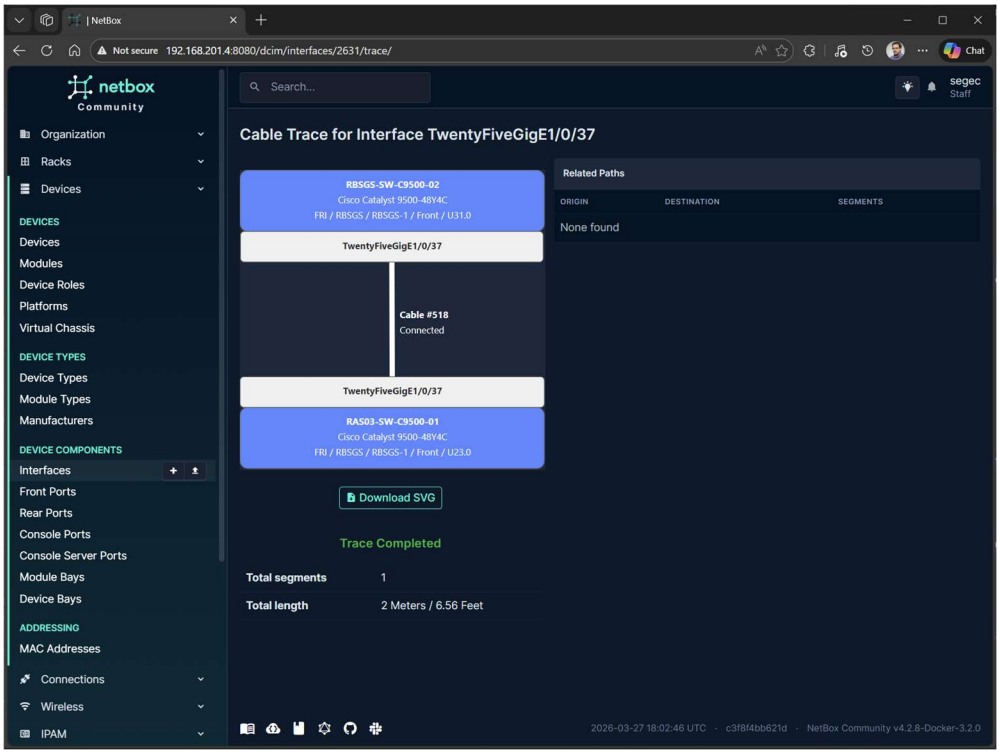


Address management





Generating topologies



Cable tracing

2.7.4 Current status and limitations of NetBox in the PoC

Within the PoC, NetBox was successfully deployed and given a basic configuration as a functional tool for recording infrastructure data. In this state, it provides a usable basis for managing selected objects and for orientation within the environment of the proposed solution. From the PoC perspective, it was confirmed that NetBox is suitable for complementing the SOC solution with infrastructure context and for supporting work with assets and addressing.

On the other hand, it should be noted that its use in the PoC remained at the level of basic record-keeping and partial context enrichment. Not all of the planned integration links to the other components of the solution were completed, and so at this stage NetBox does not function as a fully automated source of enrichment data. For the time being, its role is rather a supporting one, laying the groundwork for further extension in future stages of the project.

For this reason, NetBox in the PoC should be understood as a functional, but not yet fully developed, component. Nevertheless, its deployment demonstrated its practical usability in the proposed solution and confirmed the suitability of including it in the wider architecture of the SOC service.

2.8 DFIR-IRIS

DFIR-IRIS is used within the INNOSOC PoC solution as an open-source tool designed for the management and recording of security incidents (case management). Within our PoC, it serves as the central place where alerts from the Wazuh system are received and subsequently processed as incidents (cases). It enables the recording of artefacts, notes and tasks, the changing of incident statuses and the coordination of analysts' work.

2.8.1 DFIR IRIS deployment model in the OpenStack environment

The system is deployed in the OpenStack environment on a virtual machine running the Ubuntu 24.04 operating system. DFIR-IRIS runs in containers using Docker Compose, which allows simple deployment, administration and scaling of the solution.

Technical specification of the DFIR-IRIS deployment:

- Environment – OpenStack
- Instance name – ubuntu-24.04-kis
- IP address – 192.168.20.45 (internal IP in OpenStack), 192.168.4.97 (external IP)
- vCPU – 4
- RAM – 8 GB
- Disk – 40 GB
- OS – Ubuntu 24.04

The solution is built on a container architecture in which the individual components (web interface, backend and database layer) run separately in Docker containers. The entire application is started using the command:

docker compose up-d

DFIR-IRIS provides a web interface through which the analyst works with incidents, while the backend handles the application logic and communication with the database, where all incidents, artefacts and the change history are stored

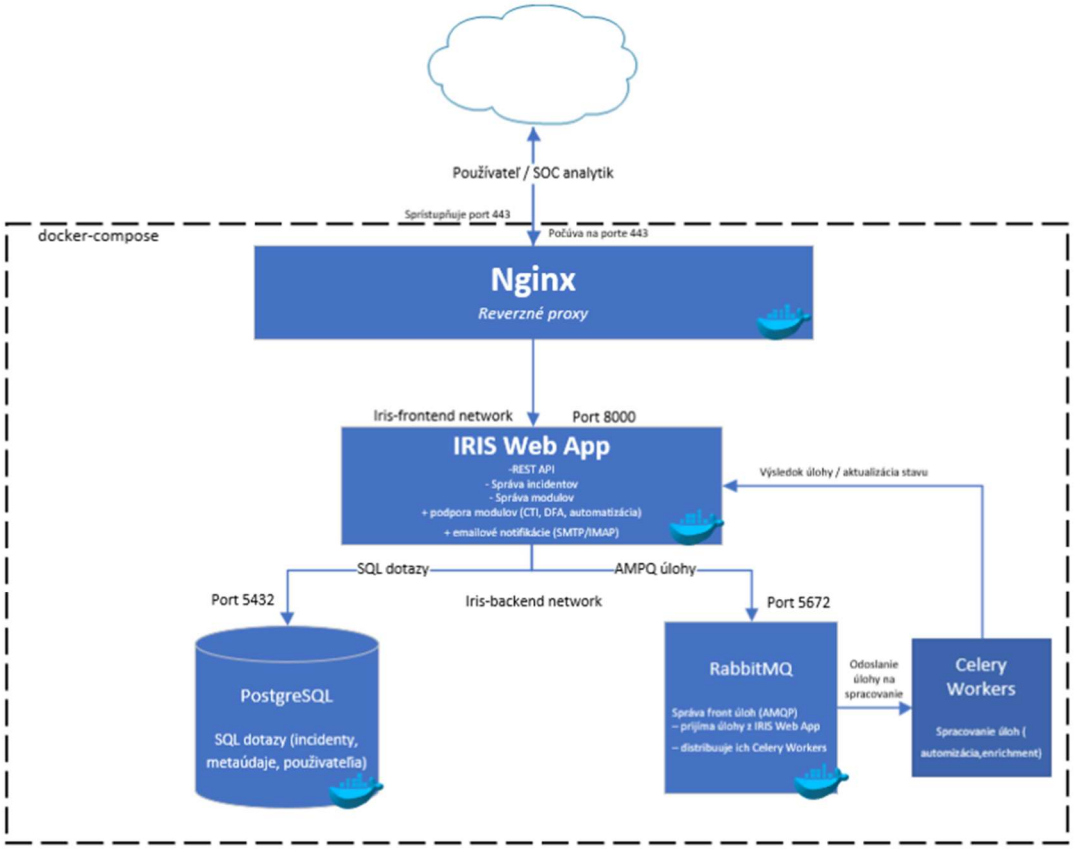


Figure 2 DFIR-IRIS architecture

The figure illustrates the architecture of DFIR-IRIS and the processing of incidents within the INNOSOC environment. The solution is based on a containerized architecture in which the individual components communicate via internal services.

The frontend is the web interface for analysts, which is used to work with incidents (cases), artefacts and tasks. Requests from the interface are processed by the backend, which provides the application logic and the management of the incident workflow. The backend communicates with the database layer, where all information about incidents is stored, including status, users, artefacts and the timeline. This architecture enables consistent record-keeping and retrospective tracing of the operations performed.

2.8.2 Basic integration links of the DFIR-IRIS component in INNOSOC

DFIR-IRIS supports integration via a REST API and webhooks. Within the PoC, it is connected to the Wazuh system, from which it receives alerts in JSON format and processes them as incidents or as updates to existing cases. The Shuffle system is connected to DFIR-IRIS and will serve as the automation layer for operations on incidents, such as enrichment, case updates or triggering responses. This connection is implemented and its functionality has been verified through testing.

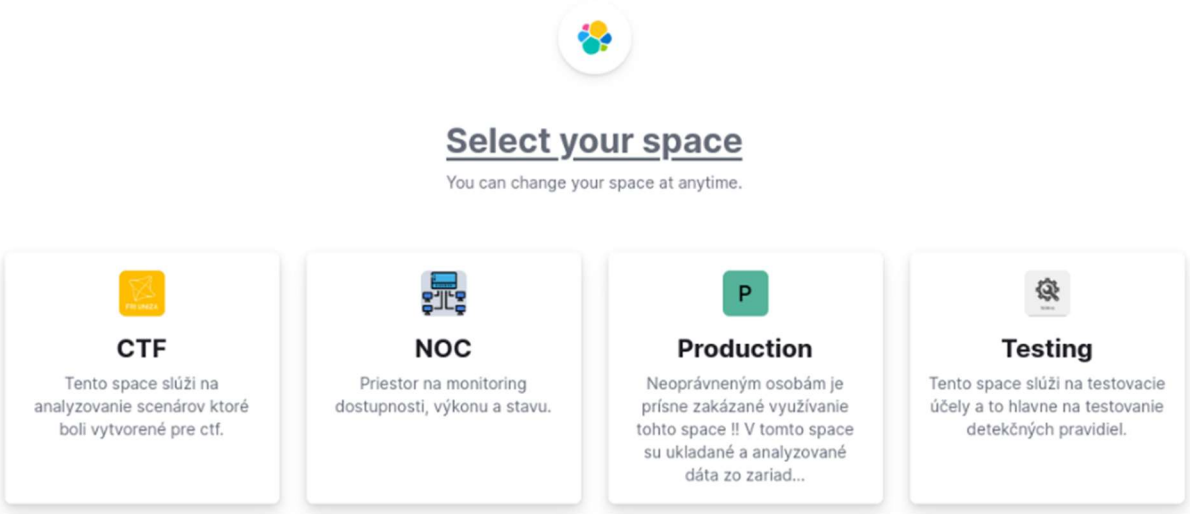
Services currently connected and tested using DFIR-IRIS:

- Wazuh – receiving alerts and creating incidents (JSON / webhook)

- Shuffle – automation and updating of incidents via the API

2.9 Kibana (ELK)

Kibana is used within the INNOSOC PoC solution as the graphical interface of the ELK stack. Kibana is deployed in the FRI UNIZA environment and is accessible via an internal URL in the faculty's domain. The logical division of rules, access and other aspects is implemented using Spaces. We have deployed the following Spaces. Their role / use can also be seen in the figure.



Main Kibana functions used in the deployed component:

- Discover – viewing records,
- Dashboards (Analytics) – displaying records in clear charts,
- Dashboards (Security) - displaying the outputs of detection rules,
- Rules - management of detection rules,
- Alerts,
- Dev Tools - administration console,
- Stack Monitoring,
- Stack Management.

The entire stack is managed in Stack Management.

2.9.1 Dashboards created

Dashboards in Kibana are used to visualize data from Elasticsearch by means of charts, maps and filters, thereby providing an overview of the monitored data.

Dashboards created:

- Suricata UNIZA
- Arkime UNIZA
- F_kibana_dashboards
- Researchers' Night
- IDS DMZ: Visualizes data from IDS sensors in the demilitarized zones and makes it possible to compare detections between Suricata and Fortigate.

- Login Activity Monitoring: Focuses on successful and failed logins
- Stack Load: Used to monitor RAM usage by the individual Elasticsearch, Logstash and Kibana nodes



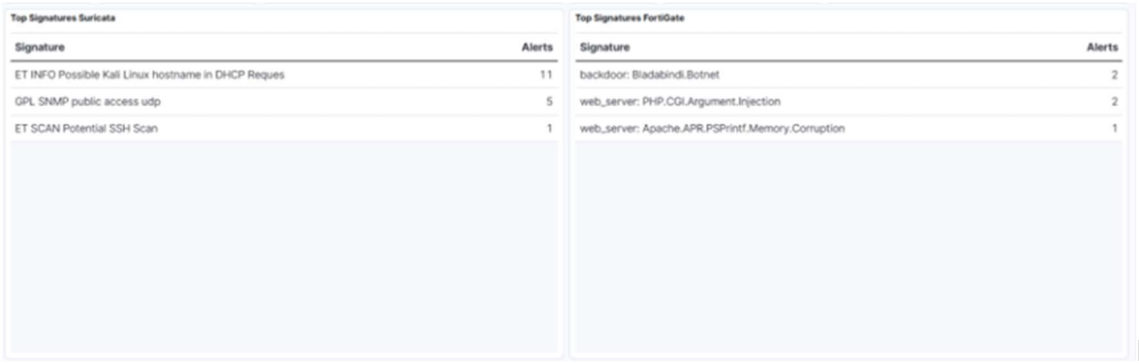
IDS DMZ dashboard, alerts

The figure shows a dashboard in Kibana that is used to visualize security events detected by IDS systems, specifically Suricata and FortiGate.

The left-hand part of the dashboard presents data from Suricata IDS, showing the total number of alerts (17), their breakdown by severity and the time course of detections over the monitored period. The graphical elements enable rapid identification of an increase or decrease in security events.

The right-hand part of the dashboard shows similar metrics for FortiGate IDS, including the total number of alerts (5), their classification by severity and their trend over time.

From the perspective of the SOC architecture, this dashboard represents a visualization and analytical layer that aggregates the outputs of detection tools and enables their clear evaluation. Kibana does not perform a detection function here, but provides analysts with a tool for interpreting events, tracking trends and supporting decision-making in security incident handling.



IDS DMZ dashboard, top signatures

The figure shows a dashboard in Kibana focused on an overview of the most frequently detected signatures (Top Signatures) from the Suricata and FortiGate IDS systems.

The left-hand part contains a list of the most frequent signatures from Suricata IDS together with the number of alerts captured. Visible are, for example, events related to DHCP communication, SNMP access or SSH scanning attempts. This overview makes it possible to identify the most frequent types of activity detected in the network.

The right-hand part shows a similar overview for FortiGate IDS, listing signatures related, for example, to botnet activity or attacks on web applications (e.g. PHP injection or Apache server vulnerabilities).

From the perspective of the SOC architecture, this dashboard represents an analytical layer that enables rapid identification of the most significant threats and trends in detections. It helps analysts determine which types of attacks or anomalies occur most frequently in the environment, thereby supporting the prioritization of further analysis and response.



IDS DMZ dashboard, top source IPs

The figure shows a dashboard in Kibana focused on the analysis of source IP addresses (Top Source IPs) detected by the Suricata and FortiGate IDS systems.

The left-hand part presents data from Suricata IDS, visualizing the most frequent source IP addresses generating security events together with their occurrence over time. The chart makes it possible to identify dominant sources of communication or potentially suspicious activities.

The right-hand part shows a similar overview for FortiGate IDS, with the individual IP addresses colour-coded and displayed on a timeline. This makes it possible to track when and how often individual sources generated alerts.

From the perspective of the SOC architecture, this dashboard represents an analytical tool for identifying the most active or potentially risky sources of communication. It enables analysts to quickly uncover suspicious IP addresses and track their behaviour over time, and it supports the process of investigating security incidents.



IDS DMZ dashboard, map

The figure shows a dashboard in Kibana in the form of a geographical map that visualizes security events detected by the IDS system (e.g. Suricata) based on the geographical origin of the IP addresses.

The map covers the area of Europe, and the individual points represent locations from which suspicious activity was recorded. The colour coding of the points expresses the severity level, which enables quick orientation in the riskiness of individual events.

On the right-hand side is the legend (Layers), which enables filtering by data source and severity. From the perspective of the SOC architecture, this dashboard represents a visualization layer that complements the analytical data with geographical context and supports the identification of the origin of security incidents.

2.9.2 Technical parameters and verification of functionality

During deployment, we defined for Kibana the maximum amount of RAM it can work with. We set it to 8 GB. Kibana does not have a RAM limit defined by default; by assigning a limit, we ensured that Kibana does not allocate all of the memory for its own use. We chose 8 GB to provide sufficient memory resources when displaying complex visualizations.

Kibana is able to verify the TLS certificate of Elasticsearch, which ensures encrypted communication. Kibana authenticates to Elasticsearch when connecting using a username and password; the default user accounts are used. Authentication is configured in the main Kibana configuration file. To increase security, the default user's password is not stored in so-called "plaintext" form; instead, we use a keystore to store the password, as with the other tools. Network communication is secured using iptables, with which we defined firewall rules. User accounts with roles designated for this purpose are used to log in to the Kibana UI, i.e. so-called role-based access control (RBAC).

Kibana is monitored by Zabbix, LibreNMS and also by Metricbeat. Through such monitoring, we track the availability and stability of the service.

The Kibana service also runs in its own LXC container on the same physical server as the rest of the ELK stack. The container has the IP address 192.168.2.188, which is accessible only from a specific IP range. The Kibana directory contains the configuration file, as well as the certificates with the Elasticsearch certificate authority for securing encrypted communication. Example of the kibana.yml configuration file:

```
server.port: 5601
server.host: "0.0.0.0"
server.publicBaseUrl: "https://[internal domain]"
server.ssl.enabled: false
elasticsearch.hosts: ["https://192.168.2.178:9200"]
elasticsearch.username: "kibana_system"
elasticsearch.requestTimeout: 180000
elasticsearch.ssl.certificateAuthorities: [ "/etc/kibana/certs/kibana/elasticsearch-ca.pem" ]
logging:
  appenders:
    file:
      type: file
      fileName: /var/log/kibana/kibana.log
      layout:
        type: json
  root:
```

```
  appenders:
    - default
    - file
pid.file: /run/kibana/kibana.pid
xpack.encryptedSavedObjects.encryptionKey: [key1]
xpack.reporting.encryptionKey: [key2]
xpack.security.encryptionKey: [key3]
monitoring.kibana.collection.enabled: false
monitoring.ui.enabled: true
monitoring.ui.elasticsearch.hosts: ["https://192.168.2.178:9200"]
        monitoring.ui.ccs.enabled: false
```

2.9.3 Use of Kibana within the PoC

Within the PoC, Kibana was used as the main tool for visualizing and analysing security data. Its main contribution lay in the fact that it made it possible to clearly display events from various sources and transform them into a form usable by SOC analysts.

Practical use was oriented mainly towards:

- viewing logs using the Discover module,
- creating analytical dashboards,
- managing detection rules and alerts,
- monitoring the status of the ELK stack.

Within the deployment, several dashboards were created, for example:

- IDS DMZ – correlation of detections from Suricata and Fortigate,
- Login Activity Monitoring – tracking login events,
- Stack Load – monitoring the performance of ELK components.

These dashboards make it possible to identify anomalies, track user behaviour and evaluate the security state of the infrastructure.

At this stage, Kibana thus did not perform detection, but provided the analytical and visualization tools needed for effective interpretation of the data.

2.10 Wiki.js

2.10.1 Role of Wiki.js in the PoC

Wiki.js is used within the PoC solution as the central tool for managing the knowledge base and internal documentation. In the proposed SOC solution, it does not act as a detection or response component, but represents a knowledge and methodological layer that supports a unified approach to documentation, procedures and internal guidelines. Its importance lies mainly in the fact that it makes it possible to systematically collect and manage the runbooks, playbooks and operating procedures of the SOC team.

Within the PoC, Wiki.js is understood as a tool that complements the technical components of the solution with documentation context. When handling security events, it thus

gives analysts quick access to defined procedures, recommendations and internal rules. At the same time, it supports the onboarding of new team members and reduces dependence on individual knowledge.

Wiki.js in the SOC provides a single place for recording and sharing knowledge. In this sense, it represents a complementary yet important component of the solution, which increases the consistency of procedures and creates a basis for further extension of documentation and integration scenarios.

2.10.2 Deployment and basic configuration of Wiki.js

Within the PoC solution, Wiki.js was deployed as a standalone web application intended for managing documentation and knowledge. From an implementation perspective, the system was operated in containers using a Docker environment and a relational database (PostgreSQL), which allows simple deployment, administration and repeatability of the installation.

The basic configuration focused on making the web interface accessible and creating the basic structure of the knowledge base. The system was prepared for managing users, roles and access rights, with the database layer ensuring the storage of content and its versions.

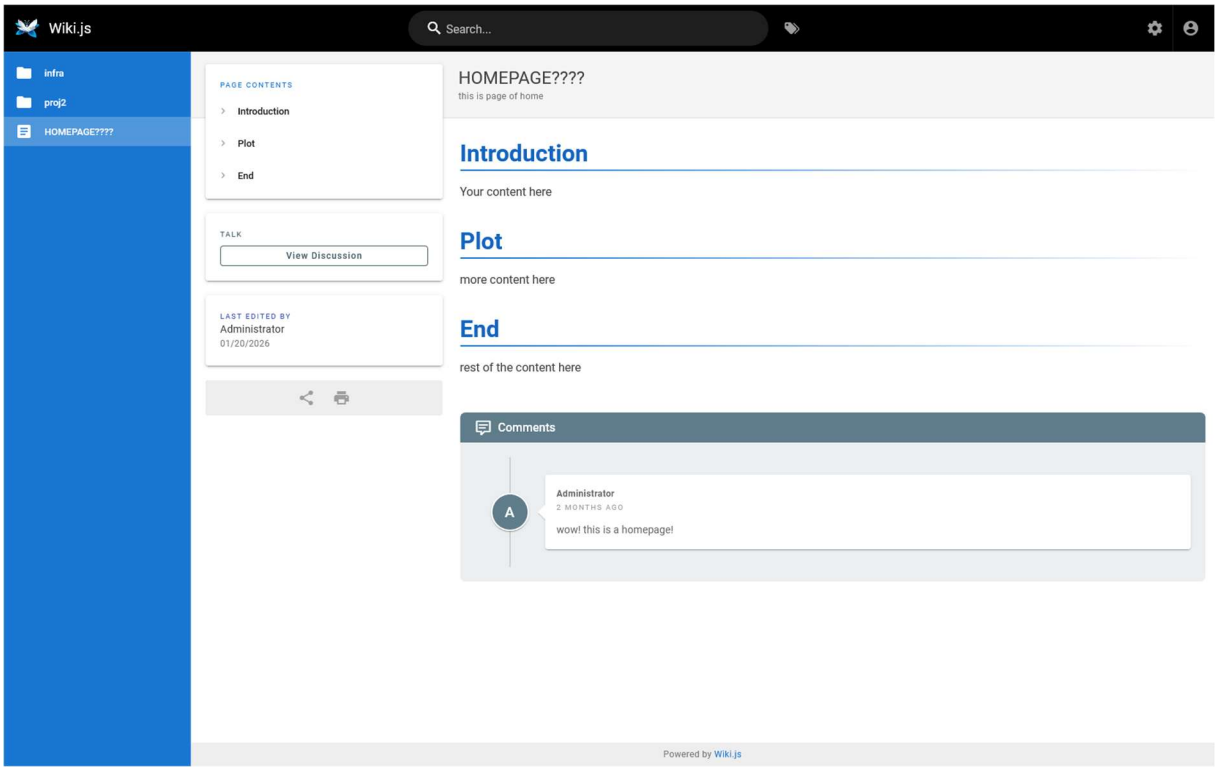
Within the PoC, basic documentation objects were entered into the system, in particular runbooks, playbooks and internal procedures, which serve as reference material in SOC operations.

2.10.3 Use of Wiki.js within the PoC

Within the PoC, Wiki.js was used as a knowledge and documentation layer to support SOC operations. Its main contribution lay in the fact that it enabled centralized storage of, and access to, the documentation needed when handling incidents and operational tasks.

Practical use at this stage was oriented mainly towards creating and managing basic documents such as operating procedures and methodologies. Wiki.js thus did not act as a technical security tool, but complemented the solution with the knowledge context needed for the effective work of analysts.

To this extent, its deployment confirmed that it is a suitable complement to the PoC solution and a usable basis for further extension, particularly in the area of linking to incident response and automation processes.

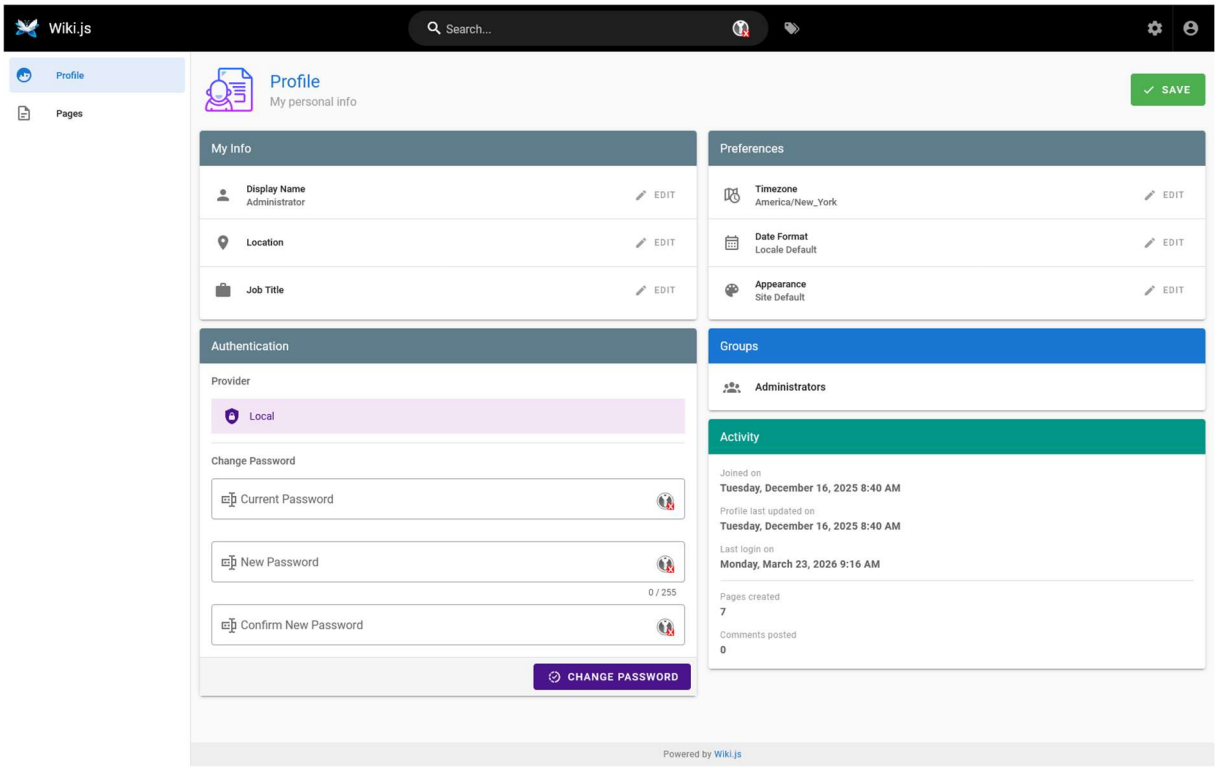


The figure shows the web interface of Wiki.js, which is used within the SOC as the central platform for managing documentation and knowledge.

On the left-hand side is a navigation panel with the page structure and the table of contents of the document, which enables quick movement between the individual sections. The central part displays a specific page divided into logical parts (e.g. Introduction, Plot, End), which illustrates the way runbooks and operating procedures are created.

The bottom part contains a comments section, which supports collaboration and the ongoing addition of information. From the perspective of the SOC architecture, Wiki.js represents a knowledge layer that ensures centralized storage of procedures and supports consistent incident handling.

Funded by the European Union
NextGenerationEU
RECOVERY AND RESILIENCE PLAN
GOVERNMENT OFFICE OF THE SLOVAK REPUBLIC

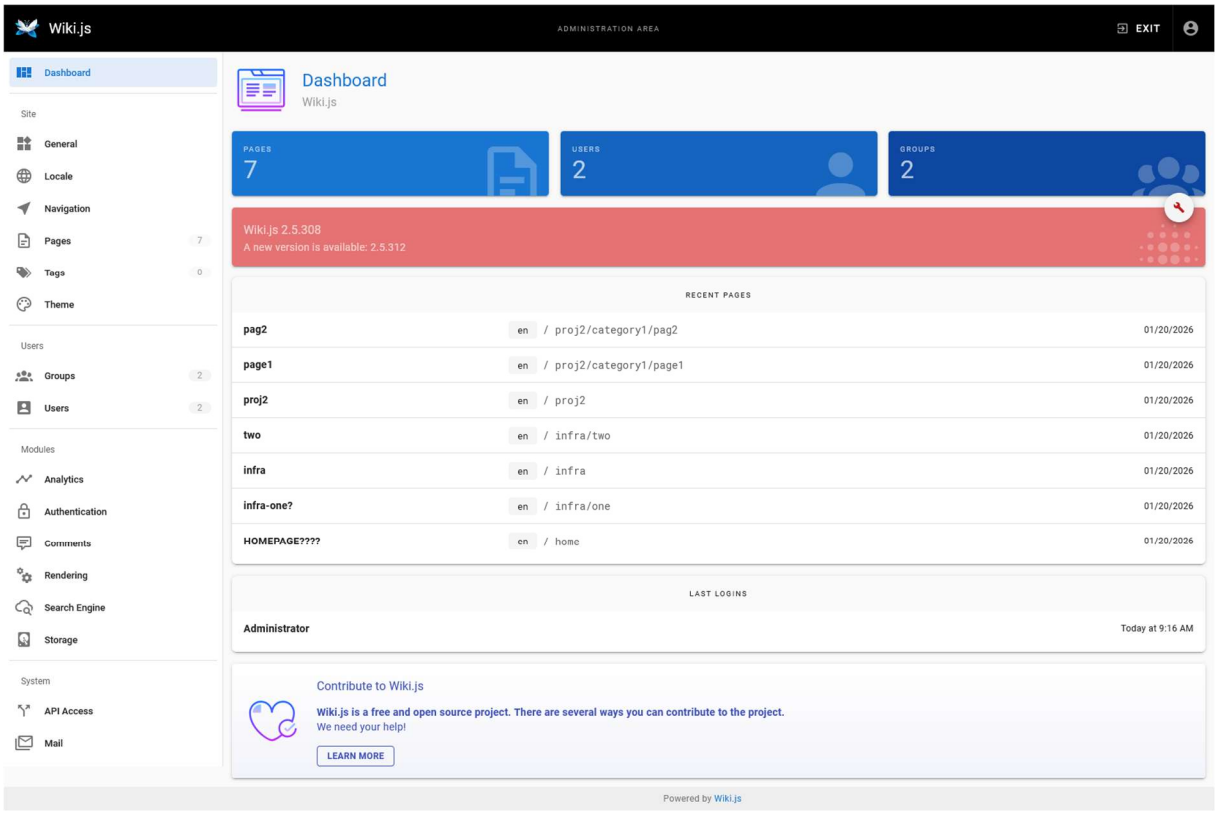


The figure shows the Wiki.js user interface in the user profile management section.

The central part contains information about the user (e.g. display name, location, job title) and configuration settings such as time zone, date format or interface appearance. It also includes an authentication section, which enables the management of credentials, including changing the password.

The right-hand side displays information about user groups (e.g. Administrators) and activity, including data on logins and the number of pages created.

From the perspective of the SOC architecture, the figure illustrates the management of users and access rights within Wiki.js, which is important for controlling access to internal documentation and ensuring controlled sharing of knowledge within the team.

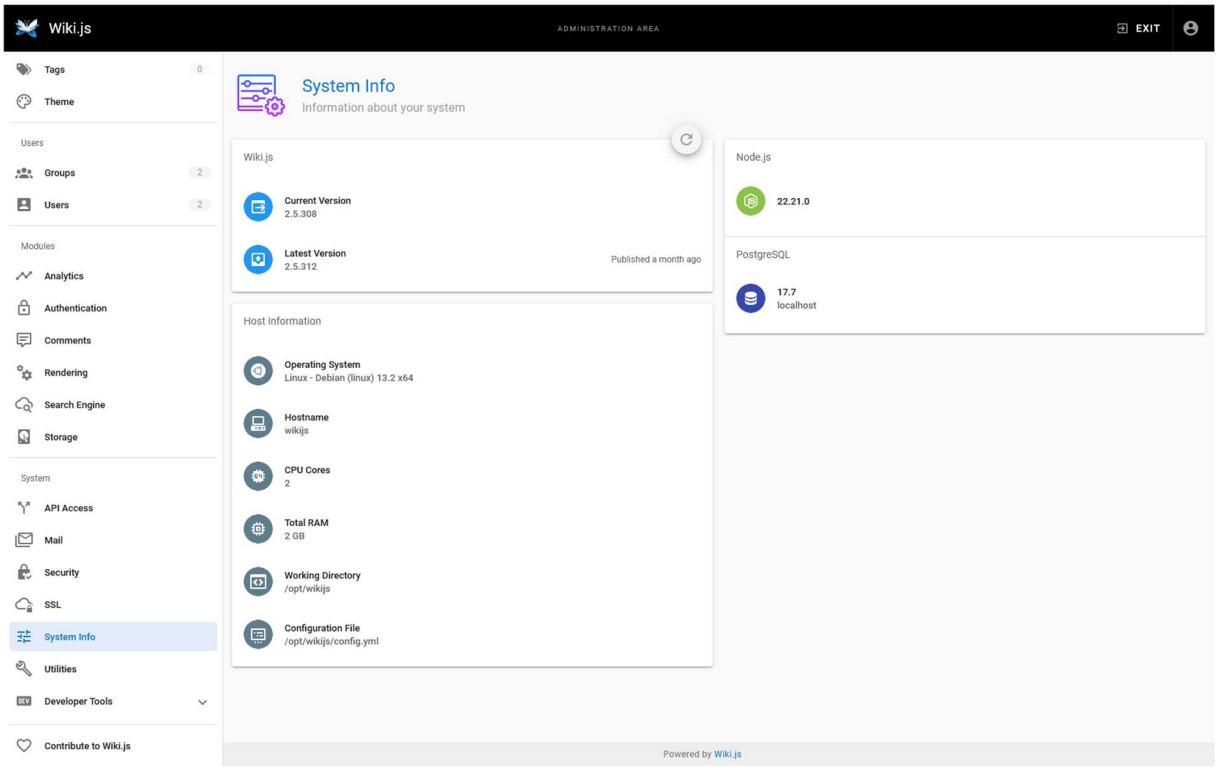


The figure shows the administration dashboard of Wiki.js, which provides an overview of the state of the knowledge base and basic system metrics.

The upper part contains summary panels with information on the number of pages, users and groups, which illustrates the extent of the recorded documentation and the user base. The central part contains a list of the most recently edited or created pages, which represent individual documentation records (e.g. runbooks or internal procedures).

On the left-hand side, an administration menu is available, enabling the management of content, users, authentication and other system settings.

From the perspective of the SOC architecture, the figure illustrates Wiki.js as a knowledge layer that centralizes documentation and provides an overview of its state and use, thereby supporting the effective management and availability of knowledge for the SOC team.



The figure shows the Wiki.js administration interface in the "System Info" section, which provides an overview of system and operational information about the deployment.

The central part displays information about the Wiki.js application version (current and available version), as well as data on the host environment, including the operating system, hostname, number of CPU cores, memory size and the location of the configuration files. The right-hand side lists the technologies on which the system is built, specifically the versions of Node.js and the PostgreSQL database.

On the left-hand side is the administration menu, which enables the management of users, authentication, search, storage and other system settings.

From the perspective of the SOC architecture, the figure illustrates the technical background and administration of the Wiki.js platform, which ensures reliable operation of the knowledge base and the availability of documentation for SOC team members.

2.10.4 Current status and limitations of Wiki.js in the SOC

Within the PoC, Wiki.js was successfully deployed and given a basic configuration as a functional tool for managing internal documentation. In this state, it provides a usable basis for recording knowledge, managing runbooks and supporting SOC operational processes. From the PoC perspective, it was confirmed that it is a suitable component for complementing the solution with a knowledge and methodological layer.

On the other hand, it should be noted that its use in the PoC remained at the level of basic documentation without wider integration links. Advanced scenarios of connection to the other SOC components (e.g. automatic referencing of incidents or integration with SIEM/SOAR) were not implemented, and so at this stage the tool functions primarily as a standalone documentation system.

For this reason, Wiki.js in the PoC should be understood as a functional, but not yet fully developed, component. Nevertheless, its deployment demonstrated its practical usability and confirmed the suitability of including it in the wider architecture of the SOC service.

3 CONCLUSION

The subject of this document was to document the functional PoC (Proof of Concept) of the INNOSOC SOCaaS platform solution, created within work package KPB5 – Implementation of the solution. The document captures the actual state of implementation of the proposed components, their deployment, configuration and basic verification of functionality in an experimental environment.

3.1 Summary of activities carried out

Within the PoC, the following components of the proposed SOCaaS architecture were implemented:

ELK Stack (SIEM) – deployed as the central system for the collection, storage, analysis and visualization of security events. The Elasticsearch cluster consists of seven nodes divided by data tier (hot, warm, cold), with Logstash handling the ingestion and normalization of logs and Kibana providing the analytical and visualization interface, including custom dashboards.

Wazuh – deployed as an EDR/SIEM agent for endpoint monitoring, detection of security events, configuration assessment (SCA), vulnerability detection and file integrity monitoring (FIM). The integration into the environment was supplemented with notification links to MS Teams.

Shuffle (SOAR) – deployed as a tool for the orchestration and automation of security responses. Within the PoC, basic integration links with the ELK, DFIR-IRIS and OpenCTI components were verified.

OpenCTI – deployed as the central platform for managing cyber threat information (CTI). The platform was connected to external data sources (Abuse.ch, MITRE ATT&CK, Ransomwarelive) and to the Shuffle component for automating the flow of CTI data.

DFIR-IRIS – deployed as a tool for recording and managing security incidents. Alert centralization, incident management, the notification system and the connection with Shuffle were verified.

Greenbone Community Edition – deployed as a tool for vulnerability scanning and management. Within the PoC, a basic scan and the sending of logs to the ELK Stack via Filebeat were verified.

GLPI – deployed as a CMDB and ITSM tool for IT asset inventory, incident management and mapping the organizational context to technical security events.

NetBox – deployed as a tool for recording the network infrastructure, IP address management and topological display of relationships between devices.

Wiki.js – deployed as a knowledge base for managing the documentation, runbooks and playbooks needed in SOC operations.

Kibana – within the ELK Stack, custom dashboards were created for visualizing security data (IDS DMZ, Login Activity, Stack Load), and the security configuration, including TLS, RBAC and the retention policy (ILM), was verified.

3.2 Nature and scope of the PoC

The functional PoC was created as a set of separately installed and configured instances of the individual components in the OpenStack environment. Each component was deployed, given a basic configuration and functionally verified as a standalone service. This approach made it possible to verify the technical feasibility of the proposed architecture and the basic functionality of each component in an isolated environment.

Within the PoC, the real possibilities of integration between the individual tools were also outlined and partially verified. The verified integration links include, in particular:

the flow of security events from monitored devices via Wazuh and Filebeat to the ELK Stack, the connection of the ELK Stack with Shuffle for the orchestration of responses to detected events, the import of alerts from Wazuh into DFIR-IRIS for centralized incident recording, the distribution of CTI data from OpenCTI to other components via Shuffle, and the sending of notifications from Wazuh to MS Teams via an Incoming Webhook.

Not all of the planned integration links had been brought to the final level of functionality at the time this output was prepared. In some cases, the components were successfully deployed on their own, but their deeper integration remained only partially implemented or was verified at a conceptual level. The PoC thus captures the actual state of implementation, not the final state of the production architecture.

3.3 Verification and testing

Functional verification of the implemented PoC is the subject of a separate output, V17 – Test results and evaluation (KPB5), which contains:

experimental tests of machine learning and artificial intelligence methods for the detection of security threats (spam, DoS/DDoS, malware, phishing, APT), verification of the detection capabilities of the deployed SIEM by means of a simulated attack based on the Cyber Kill Chain model, automated testing of detection rules using Atomic Red Team, manual testing of correlation rules in Kibana, load testing of the deployed ELK Stack, integration tests between components including DFIR-IRIS, Shuffle and OpenCTI, and functional tests of the individual components (Wazuh, Greenbone, GLPI, Wiki.js, ELK Stack, NetBox).

The test results demonstrated that the proposed solution, in its current scope, is functional and operationally capable. Detailed results, test identifiers and their evaluation are given in document V17.

3.4 Final assessment

The implemented PoC demonstrated the technical feasibility of the proposed architecture of a multi-tenant cybersecurity operations centre designed as an open cloud service. All key components were successfully deployed, configured and functionally verified in an experimental environment.

In terms of further development of the solution, the PoC provides a sufficient basis for moving to the phase of extended integration and pilot operation. The main areas of further development include, above all, deepening the integration links between components, extending the automation of security responses through the SOAR platform, deploying additional CTI data sources and optimizing detection and correlation rules based on the findings from testing.

The PoC as a whole confirms that the chosen architectural approach based on open technologies is viable and provides a flexible basis for building a commercial SOCaaS service that can be tailored to the needs of various organizations.