Web application for visualizing the security situation in a computer network

Lukáš Matta, Martin Husák

2020

Abstract

This document describes the output of the project of the same name ***"Research of tools for assessing the cyber situation and supporting decision-making of CSIRT teams in the protection of critical infrastructure"*** (VI20172020070) addressed in ***the Security Research program of the Czech Republic*** in the years 2017-2020 at Masaryk University. The document contains a description of the result, installation instructions, user documentation and programming documentation.

Content

Introduction

This document contains documentation on the result ***"Web application for visualizing the security situation in the computer network"*** (hereafter Orient), which is the output of the project ***"Research of tools for assessing the cyber situation and supporting decision-making by CSIRT teams in the protection of critical infrastructure"*** (VI20172020070, hereinafter referred to as the project CRUSOE) solved in ***the Security Research of the Czech Republic*** program in 2017-2020 at Masaryk University.

The aim of the aforementioned project was to create tools that help specialists in the cyber security team to map and navigate the current cyber security situation in the computer network, quickly and well decide on the procedure for solving ongoing incidents and implement the proposed solution. The set of created tools reflects the thought concept of the so-called OODA cycle, which consists of four phases - Observe, Orient, Decide and Act. The OODA cycle was designed and described as a general procedure for information gathering and decision support.
The user of the OODA cycle iterates through individual phases, thanks to which he formalizes his thinking and is thus able to make decisions and act effectively and quickly without unnecessary mistakes. First, it is necessary to collect information about the environment (Observe phase), then to orient oneself in it (Orient phase), then to decide on a suitable next course of action (Decide phase) and finally to implement this procedure (Act phase). As part of the CRUSOE project, a tool was created for each phase of the OODA cycle, which allows users to implement the objectives of the given phase. ***"Web application for visualizing the security situation in a computer network"*** implements the Orient phase, i.e. the phase of orientation in the collected information and data.

The Orient software consists of a web application that conveniently presents and visualizes information collected by dedicated tools, such as the tools developed by the CRUSOE project for the Observe phase. Information about the cyber security situation in a computer network mainly consists of a list of devices and services in a given network, a list of found vulnerabilities and a history of security incidents, marking of critical infrastructure elements, a list of contact details, network topology, a list of users and their authorizations, links between individual entities and the like . The Orient software retrieves this data from the database and allows the user to browse and search the data. Different views of data reflect the cybersecurity teams' requirements and field experience. When dealing with cybersecurity incidents, you often need to know the context. That's why, for example, the Orient software includes a view of an entity in the network (device, service, user) and its immediate surroundings and connections, which allows you to quickly pick up the context. Other significant insights that the software allows include seeing the number of vulnerable devices for each vulnerability, which allows you to plan and prioritize remediation. The software also integrates visual tools and control panels for easier control and control of other tools created within the CRUSOE project.

This document consists of five parts. An introduction containing a summary of the technical and economic parameters of the result is followed by a description of the result explaining the principles on which

the software is built. Installation instructions, user documentation and programming documentation are presented in other sections.

Technical parameters of the result

The software in the form of a web application presents and visualizes data about the current cyber security situation in the computer network. The web application offers different views of the data collected by other tools. Significant data views include visualization of the context of a given network entity and a global overview of vulnerabilities and vulnerable devices in the network. The web application further implements controls and reports on the functioning of other software results of the same project. These elements and reports allow you to visually control and control data collection and further processing, thus creating an alternative to the default control of these tools.

The software is distributed as open-source under the MIT license, the owner of the result is Masaryk University, IÿO 00216224.

Contact person: RNDr. Martin Husák, Ph.D. Institute of
Computer Technology Masaryk
University Šumavská 416/15

Brno 602 00

e-mail: husakm@ics.muni.cz phone:
+420 549 49 4857

Economic parameters of the result

The market segment is represented by organizations that operate or build their CERT/CSIRT, Security Operation Centre, or provide cyber security as a service.
The result allows users and operators of networks and services to clearly visualize data about the computer network and the outputs of tools to support the activities of the cyber security team.
The software allows users to more quickly view the data about the protected network, orient themselves in the situation and decide on the next course of action in solving a security incident. The result thus brings a saving in the time of solving cyber security problems, which reduces the demands on human resources, or allows users to handle more tasks at the same time. The result is thus a shorter response time to a cyber security incident, thereby contributing to a general increase in the level of cyber security in the organization. Use of the result is licensed free of charge.

Description of the result

The web application for visualizing the security situation in the computer network (hereinafter referred to as the Orient software) is a web application that presents and visualizes the information collected by the tools intended for this purpose, for example, the tools created as part of the CRUSOE project for the Observe phase. It also includes common functionality for all visualizations, such as access control, panel overview, and the like. Information about the cyber security situation in a computer network mainly consists of a list of devices and services in a given network, a list of found vulnerabilities and a history of security incidents, marking of critical infrastructure elements, a list of contact details, network topology, a list of users and their authorizations, links between individual entities and the like . The Orient software retrieves this data from the database and allows the user to browse and search the data.
Different views of data reflect the cybersecurity teams' requirements and field experience. When dealing with cybersecurity incidents, you often need to know the context. That's why, for example, the Orient software includes a view of an entity in the network (device, service, user) and its immediate surroundings and connections, which allows you to quickly pick up the context. Other significant insights that the software allows include seeing the number of vulnerable devices for each vulnerability, which allows you to plan and prioritize remediation. The software also integrates visual tools and control panels for easier control and control of other tools created within the CRUSOE project.

Orient software covers two basic areas - data visualization and control panel. The area of data visualization aims at ways of presenting the outputs of the CRUSOE project to the user so that he can quickly obtain an overall overview of the state of the network and, at the same time, can easily obtain detailed information about individual parts of the network. Visualization components aggregate the outputs of other tools created within the project. Data visualization and presentation components are linked in a comprehensive web application to enable the user to get a global overview of the network status as well as a detailed view of specific machines. Control panel components then allow you to control data collection and processing tools and other tools, for example, decision support components and the application of reactive measures to mitigate cyber attacks. Both types of components, visualization and control elements, also need to be integrated into a unified system. For this purpose, a comprehensive control panel was created, which solves common functionality, for example, login and user management, and allows access to individual components using switchable panels.

Use cases

Orient software implements a number of different use cases, which are worth describing first. The use cases are based on the various situations and activities that members of cybersecurity teams encounter when dealing with incidents. Here are the software functionality requirements and a brief description of the use case for each of them. Not all use cases require a specialized component, as further shown in the requirements analysis and component design, which are the content of other parts of this document.

Network map

The first expected functionality is the visualization of the network graph with the possibility of interaction, which allows the user to orientate himself in the network and search for relevant entities in it. Instead of a topological map, implementations of which are commonly available, it should be more of a network map from a logical point of view, for example, taking into account subnets and other links between network devices than connecting them by a network line. Connection through the same administrator or user, similar device characteristics and software equipment, and the like is offered. The component will present the network topology in the form of a network graph with a focus on quickly obtaining an overview of the security situation with the possibility of interactive search. The display of each node of the network graph will include a complete menu of offered operations or information related to the given device with the possibility of searching. Individual menu items will be connected to other components of the visualization dashboard to facilitate obtaining detailed info

Detailed information about the machine

The goal of visualizing detailed information about the machine is to present information about a specific machine in the network to the user in a clear way. The component fulfills the use case of quickly finding information relevant to solving a security incident. Such information includes data about the machine's operating system, applications running on the machine, services provided by the machine and its possible involvement in a computing cluster.

Overview of current threats and vulnerabilities

Another goal of the tool is to provide a comprehensive overview of current threats and vulnerabilities in the network in one place, which also corresponds to the use case, in which the component provides the user with an overview of vulnerabilities detected by various systems and enables problems in the network to be sorted or otherwise structured according to severity. The basic division will be according to the severity of the vulnerability itself according to the CVSS metric, which will allow further refinement using the importance of the vulnerable machine and the requirements for this machine.

Overview of network attacks

The goal is also to display an up-to-date overview of attacks detected within the organization. This overview will fulfill the following use cases: 1. Visualization of attacks

on the organization's network

The component creates a Real-Time Threat Map capturing the current security situation of the organization from the point of view of the global Internet. This will enable the user to gain an overview of the organization's current security situation from a global perspective, the geographic origin of individual attackers and the relationships between individual attacks.

2. Visualization of attacks on individual devices in the organization's network

The component displays the current security situation of the organization from the point of view of individual end devices. This gives the user an overview of the types and number of attacks conducted on individual devices and thus allows him to detect a coordinated attack against a certain type of device or part of an organization. The map will also include information about the same types of attacks conducted on individual devices and contact information for the administrator of the given address range.

Mapping of critical dependencies of machines and services

The goal of the tool is to increase the user's awareness of the state of the network in terms of the importance of individual devices and their interdependencies. This leads to two use cases:

1. Visualization of dependencies in the network - The component will provide the user with an overview of all detected dependencies of a particular machine on other machines or services in the network.

2. Visualization of the organization's critical servers - The component will allow to highlight the machines in the network based on their calculated importance. This highlighting will be further adjustable using individual importance metrics such as the number of services provided, the number of users, the number of dependencies, and so on. Furthermore, the component will display the organization's requirements for the machine (availability, confidentiality, integrity).

Display the possible effects of a particular attack or threat

One of the goals of the tool is to provide an overview of current threats and attacks, including information on their possible impact on the organization's network. This fulfills the following use cases:

1. Evaluation of the immediate state and possible progress of the attack - the component displays the currently detected attacks and threats in the network to the user, which it maps to individual network elements. This status information will be further supplemented by the calculation of the possible further progress of the attacker or the further spread of the threat using the projection of attack graphs into the network topology.

2. Evaluation of attack risk - the component will make it possible to highlight threats whose relative impact significantly exceeds other current threats in the network. This will make it easier for users to make decisions when prioritizing solutions to network problems based on quantifying the damage a threat can cause.

Visualization of defensive action

The goal of the tool is to provide an overview of possible defensive actions and their impact on the organization's network. This solves the use case, which is the design of a defensive action and the determination of its suitability - the component displays a list of available defensive actions for each ongoing attack. In addition, for the displayed actions, details about their impact on both the ongoing attack and the state of the organization's network will be available, making it easier for the user to choose the most suitable one.

Visualization of active defense elements

The goal of the tool is to display the available active defense elements, their current configuration and the effects of activating their parts. This fulfills the use cases:

1. Visualization of settings of active defense elements - the component displays all active defense elements in the network to the user, including their location in the network or on specific machines.
   After clicking on a defense element, the user will see its current settings and configuration options.

2. Simulation of the effects of a defensive measure - the component displays the possible effects of the performed action in terms of the availability of individual services or network elements. This will provide an overview not only from the point of view of networking, but also of the dependencies between individual machines in the network and thus enable a comprehensive assessment of the effects of the action even before it is carried out.

The goal of the tool is to appropriately visualize the state of the measurement infrastructure, tools and processes that are used to collect data for further analysis. This fulfills the use case of checking the up-to-dateness and correctness of information. Before analyzing the data, it is often a good idea to check that the data is available, up-to-date and correct. The analysis cannot be started if the data is missing or incomplete, which may be caused by the failure of the measuring instruments. If the data is available but out of date, again due to the unavailability of measuring tools for example, a mistake can be made in the analysis of security incidents. Configuration changes or errors in the systems can also trigger reactions such as an increase in the volume of provided data. In such a case, the analyst may be overwhelmed with data and thus analysis becomes more difficult.

Requirements analysis

In this section, the web application requirements and their brief analysis are presented. Functional and non-functional requirements are listed first. Functional requirements express what the system should be able to do. Non-functional requirements describe limitations placed on the system, for example regarding performance, security, or technical parameters. The data model with which the web application works is also described.

Functional requirements

The basis for the creation of the web interface are the following functional requirements, which were identified during the analysis and design of the system and which are based on the use cases described above. These are: • management of tasks and workers, • obtaining an overview of the organization's missions, • applying configuration for individual missions, • searching for a network node by IP address, • verifying the occurrence of vulnerabilities according to the CVE identifier. Based on these requirements, four panels were designed, which are presented in more detail later in the text. Individual panels contain one or more components that both fulfill functional requirements and enable use cases to be implemented. One panel or component can implement multiple use cases.

Broken requirements

Non-functional requirements are those requirements that do not directly relate to what the system should be able to do, but are mainly requirements for performance, security, availability and various other technical parameters.

The resulting web interface should support authentication using OpenID Connect (OIDC), an extension of the OAuth 2 protocol with an authentication and user information retrieval API.[1]
Using OIDC allows you to connect a web application to the single sign-on system of the organization in which the application is deployed.

[1] https://openid.net/connect/

Since the web interface contains multiple graphic visualizations, it is important to also choose visualization libraries that will work smoothly and without unnecessarily long loading times.

Considering the frequency of visualization components and types of visualizations, it is also necessary to choose suitable visualization libraries that will work smoothly and without unnecessarily long loading times. It is also appropriate to take into account the availability of documentation and the vitality of the project, i.e. whether the given library is actively developed or maintained.

The web interface should be in the form of a SPA (Single Page Application). Unit tests and documentation should also be part of the resulting code if the technology used allows it. There should also be a simple option to run the web interface locally using Vagrant and Ansible technology [2], or starting the web interface in in the form of a Docker container[3].[4]

There are no specific requirements for the screen sizes on which the web interface should be available, so the application should be available for standard resolutions used on desktops or laptops. Support for mobile devices is not required, on the contrary, it is advisable to calculate with the possibility of running the web application on larger screens with a high resolution, for example in the case of deployment in security operation centers (SOC).

There are no additional requirements for the technologies and programming languages used, it is only recommended to stick to advanced languages and tools and, if possible, use similar technologies as other software tools created within the CRUSOE project.

Data model

The web interface mainly visualizes data that provides other tools that were created within the framework of the CRUSOE project. To facilitate collaboration between tools and unify the format for storing and manipulating data, a data model was proposed that describes the entities and relationships that occur in a given domain, i.e. network security with regard to building situational awareness. The data model presented in the professional article adheres to the graph representation of entities and relations, [5] where entities appear as nodes in the graph and relations as edges between them. Edges and nodes can then contain a number of other information in their parameters. For the purposes of working with this data, it is advisable to list the entities with which the web application works. The data model is extensible, so it is possible that in the future it will be extended with additional entities and sessions and it will be possible to implement additional visualizations. At the time Orient software development was completed, the following list of entities that could be visualized was known. The list is

[2] https://www.vagrantup.com/ https://

[3] www.ansible.com/ https://

[4] www.docker.com/ 5 KOMÁRKOVÁ,

Jana; HUSÁK, Martin; LAŠTOVICKA, Martin; TOVARŷÁK, Daniel. CRUSOE: A Data Model for Cyber Situational Awareness. In: Proceedings of the 13th International Conference on Availability, Reliability and Security. Association for Computing Machinery, 2018. ARES 2018. ISBN 9781450364485.

divided by entity categories as described in the relevant article and is also shown in its entirety in Figure 1.

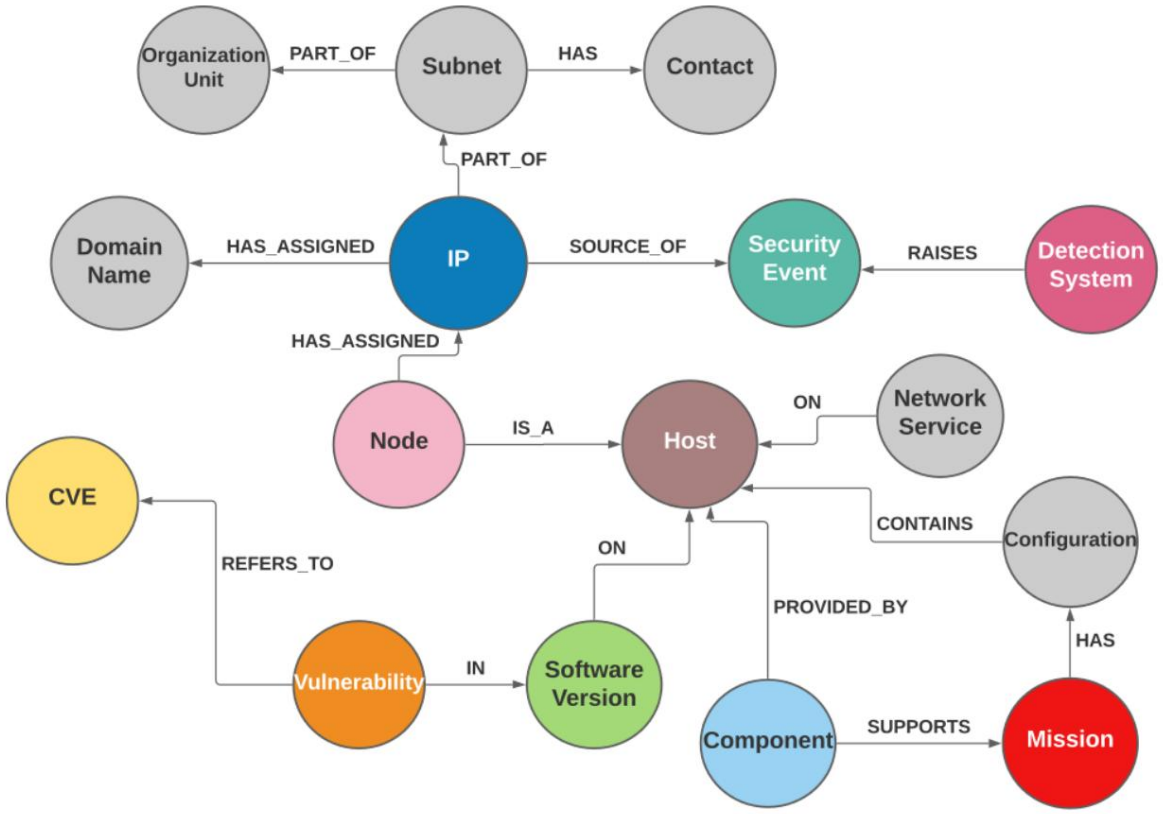


Figure 1: Schematic of the data model.

Component

The Component node represents a service that supports the operation of a certain mission.

Attributes:

- name - component name, e.g. Email blacklist.

The Configuration

node represents the configuration available for a given mission.

Attributes:

- config_id - configuration identifier, • availability
- measure of mission availability after configuration application, • confidentiality - mission confidentiality measure after configuration application, • integrity - mission integrity measure after configuration application, • time - configuration creation time in YYYY-MM- format DDHH:MM:SS.

Contact

The node represents contact information for a person or the team responsible for that subnet. This is usually an email address.

Attributes:

- name - Email address or other available contact.

CVE

A node represents the CVE code for a given vulnerability and contains the vulnerability attributes given by CVSSv2 or CVSSv3 metrics.

Attributes:

- CVE_id - CVE code of the given vulnerability, e.g. CVE-2014-0160, •
access_vector - indicates the way in which the vulnerability can be exploited, e.g.
network,
- access_complexity - indicates the difficulty of exploiting the vulnerability, e.g. LOW, •
attack_vector - attribute corresponds to the access_vector attribute in CVSSv2, •
attack_complexity - attribute corresponds to the access_complexity attribute in CVSSv2, •
authentication - indicates the number of authentications that must be performed during an
attack, • availability_impact_v2 - attribute CVSSv2 metrics indicate how service availability may be impaired during an attack, • availability_impact_v3 -
CVSSv3 metric attribute indicates how service availability may be impaired during an attack, • base_score_v2 - overall CVSSv2 metric vulnerability score, •
base_score_v3 - CVSSv3 metric overall vulnerability score, •
confidentiality_impact_v2 - attribute of the CVSSv2 metric indicates how the
confidentiality of the service can be violated during an attack, • confidentiality_impact_v3 - attribute of the CVSSv3 metric indicates how the
confidentiality of the service can be violated during an attack, • description - verbal description of the vulnerability, • impact - verbal description of
the impact of the vulnerability, • integrity_impact_v2
- the attribute of the CVSSv2 metric indicates how the
integrity of the service can be violated during an attack, • integrity_impact_v3 - the attribute of the CVSSv3 metric indicates how the integrity of the
service can be violated during an attack, • obtain_all_privilege - the attribute indicates whether the vulnerability can be improved

to system administrator rights,
- obtain_user_privilege - the attribute indicates whether the vulnerability can be worked on
to system user rights,
- privileges_required - the attribute indicates what rights are needed to carry out the attack, •
published_date - the date of publication of the vulnerability in the format YYYY-MM-DDTHH:MMZ.

DetectionSystem

The node represents the detection system that creates security events available under SecurityEvent nodes.

Attributes:

- name - name of the detection system.

DomainName

A node represents a domain name assigned to an IP address.

Attributes:

• domain_name - domain name, • tag - DNS
record type, eg A/AAAA.

Guest

A node represents a device on a network. A node has no attributes.

IP

A node represents an IP address assigned to a node on the network. Attributes:
• address - IP address of the device.

Mission

The node represents the organization's mission.
Attributes:
• name - name of the mission,
• description - textual description of the
mission, • structure - structure of mission dependencies on components and devices specified in
JSON format.

NetworkService

A node represents a network service running on a given device.
Attributes:
• port - the port on which the network service is operated, • protocol -
the protocol of the network service, typically TCP or UDP, • service - the
name of the network service.

Node

A node represents a node in a computer network.
Attributes:
• dependency_degree - degree of dependence on other network devices, •
topology_betweenness - degree indicating the position of the device within the network topology.

OrganizationUnit A
node represents the organizational unit under which certain subnets fall.
Attributes:
• name - name of the organizational unit.

SecurityEvent

The node represents a security event that was detected by the detection system.
Attributes:
• confirmed - the attribute indicates whether the security event was confirmed, • description -
textual description of the security event, • detection_time - security
event detection time, • type - type of security event.

The SoftwareVersion

node represents the software installed on the given device.

Attributes:

• tag - indicates the program with which the software was detected, e.g. nmap_client, • version - name and version of the detected software.

Subnet

A node represents a subnet.

Attributes:

• note - a note that specifies the subnet in more detail, • range - the range of the subnet specified in CIDR notation.

The Vulnerability

Node represents a vulnerability that is further specified by the CVE node.

Attributes:

• description - description of the vulnerability.

Overview of components

Based on the general functional requirements, we decided to divide the web interface into four panels, while a proposal was created for each panel, which was then consulted with the entire solution team. In the following subsections, we present the individual panels in more detail.

Individual panels will contain sub-visualizations as designed in the referenced technical reports, or in this technical report in Figure 1. Other panels will contain control elements for sub-components of the project's software outputs. The list of panels in progress is as follows:

• Control panel software for the Orient phase, • Panel for browsing and searching in the common database including detailed information about guests in the network,

• Visualization of the network topology including the occurrence of KII elements and their dependencies, • Visualization of the occurrence of vulnerabilities, • Visualization of the history of security incidents, • Control panel software for the Decide phase, including visualizations of the impact of an attack and support decision making

• Software control panel for the Act phase and components for controlling active defense elements.

The list is not exhaustive, during the finalization of software development and user testing, content may be moved between different panels to suit common software usage scenarios and facilitate the decision-making process about security threats and incidents from the operator's perspective.

Task Manager panel

In order to obtain the data required for increasing situational awareness in the network, it is necessary to run a relatively high number of different tasks. The management of these tasks is provided by Orchestration

service Result No. 1 Software for recording vulnerabilities in the computer network, specifically it is implemented by the Celery system[6].

The goal of the panel is to provide the user with an overview of these tasks and the ability to manage them. The user should be able to search for a task based on its parameters. The panel should also contain an overview of the status of Celery workers (mainly their workload and number of tasks) and the possibility of their configuration. It would also benefit from a pie chart that breaks down tasks based on how successful they are at completion, so that the user can see at a glance whether a large number of tasks are failing. The diagram should be continuously updated without the need to manually refresh the page in the browser.

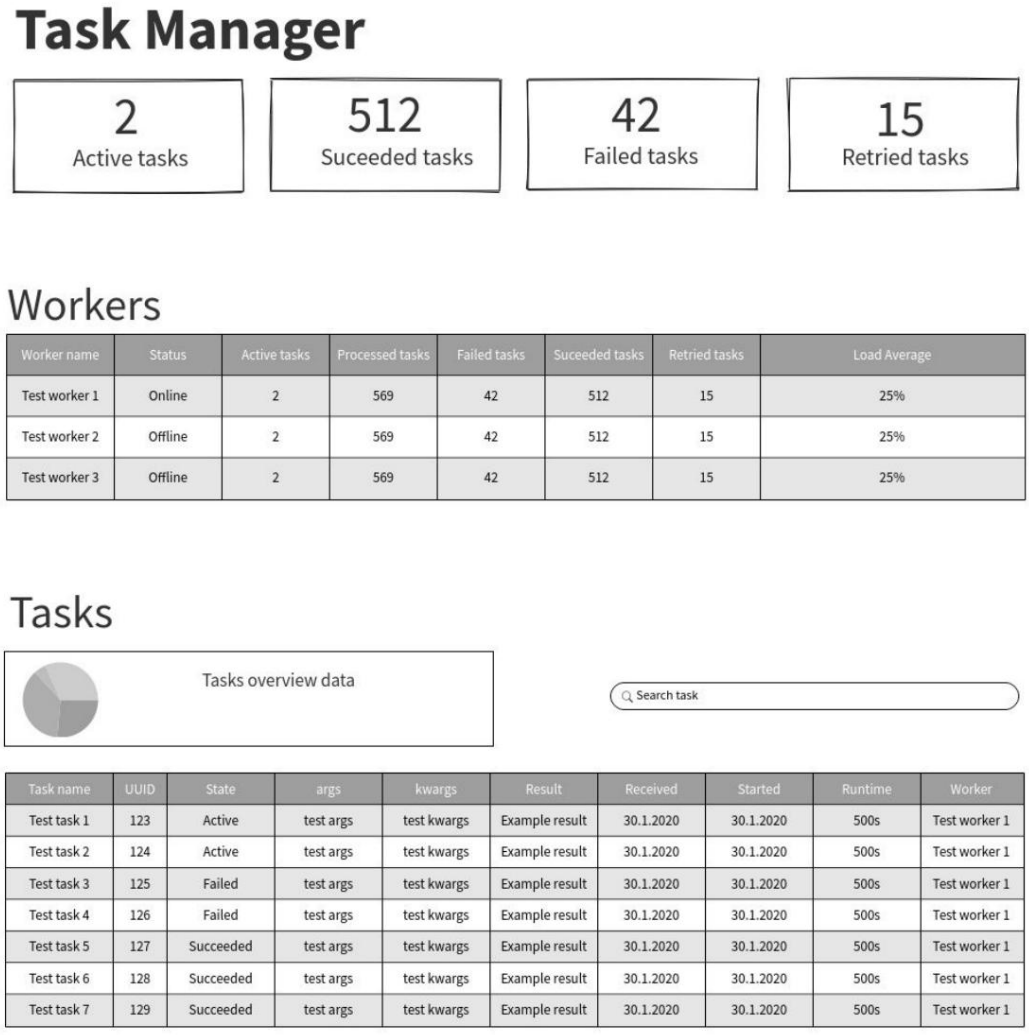


Figure 2: Task Manager panel wireframe.

[6] http://celeryproject.org/

Figure 2 shows a design of the Task Manager panel. The panel should consist of the following parts: • information about the
number of tasks based on their status (active, successfully completed,
unsuccessfully terminated and repeatedly started),
• an overview of Celery workers, their status, the number of assigned tasks and
workload, • a pie chart showing tasks according to their status,
• an overview of tasks sorted from the most recent with the possibility of searching.

Network Visualization panel

The ***Network Visualization*** panel should provide the ability to search for a node in a computer network based on its IP address. It should then display a graph for that node that will display the properties and entities associated with that node. The user will thus be able to quickly create an overview of the given device in the network, of the software installed on the device, of belonging to the subnet, or vulnerabilities associated with the installed software. Information is retrieved from ***the Neo4j*** database using CRUSOE Project Result #1 support tools.

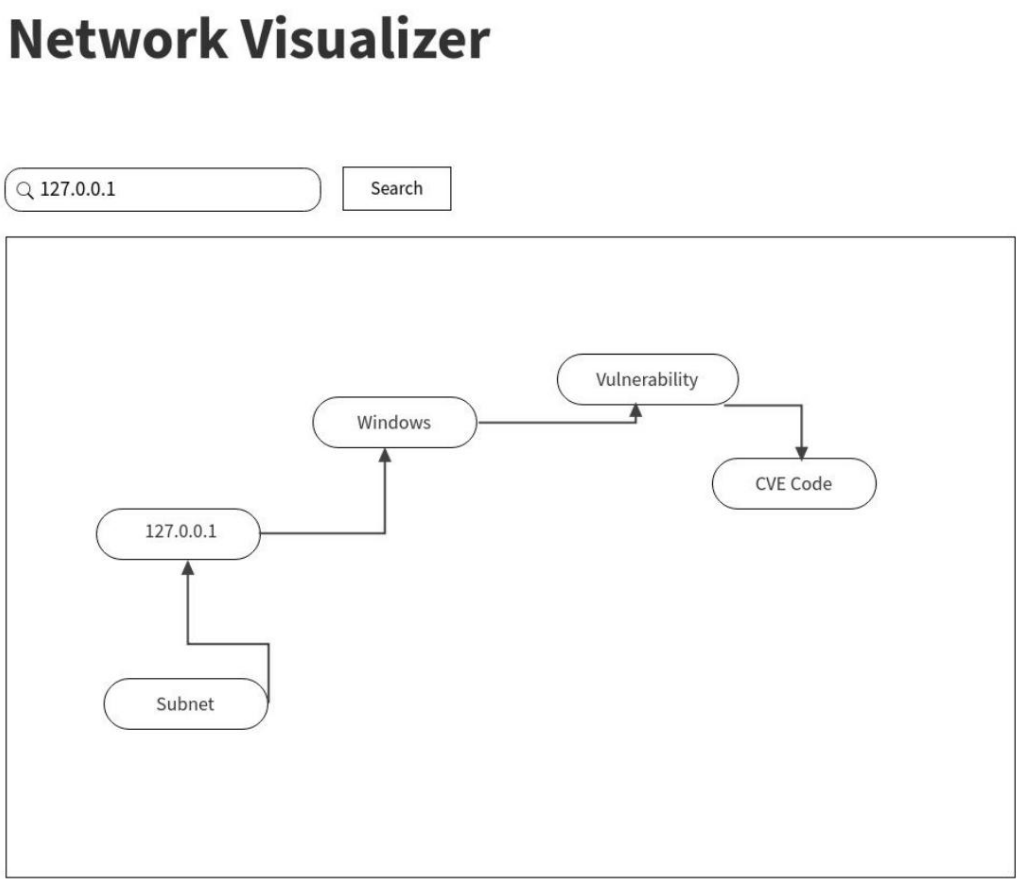


Figure 3: Network Visualizer panel wireframe.

The following information should be available for a given node in the
network: • subnet
membership, • assigned domain names,

- installed software, •
  vulnerabilities found in installed software.

Another benefit is the user's ability to click on a given node, and after clicking on it, its other relevant surroundings will be displayed. The design of the Network Visualization panel is shown in Figure 3.

Vulnerability panel

The ***Vulnerability*** panel should support the ability to search for a vulnerability based on its CVE (Common Vulnerabilities and Exposures) identifier. The data needed for this panel is available in the central Neo4j database, similar to the ***Network Visualization panel.***
The panel should display a description of the vulnerability, a pie chart showing its occurrence in subnets, and a table containing the computers on the network that have the software containing the vulnerability. Clicking on one of the computers in the table redirects the user to the ***Network Visualization panel,*** which displays the selected computer and its surroundings in the form of a graph.

The main goal of the panel is to enable the user to verify the presence of a given vulnerability in the managed network, and in the event of its occurrence, the user should be able to quickly form a picture of the extent of the spread of this vulnerability and its severity.

Figure 4 shows a draft of the ***Vulnerability panel.***

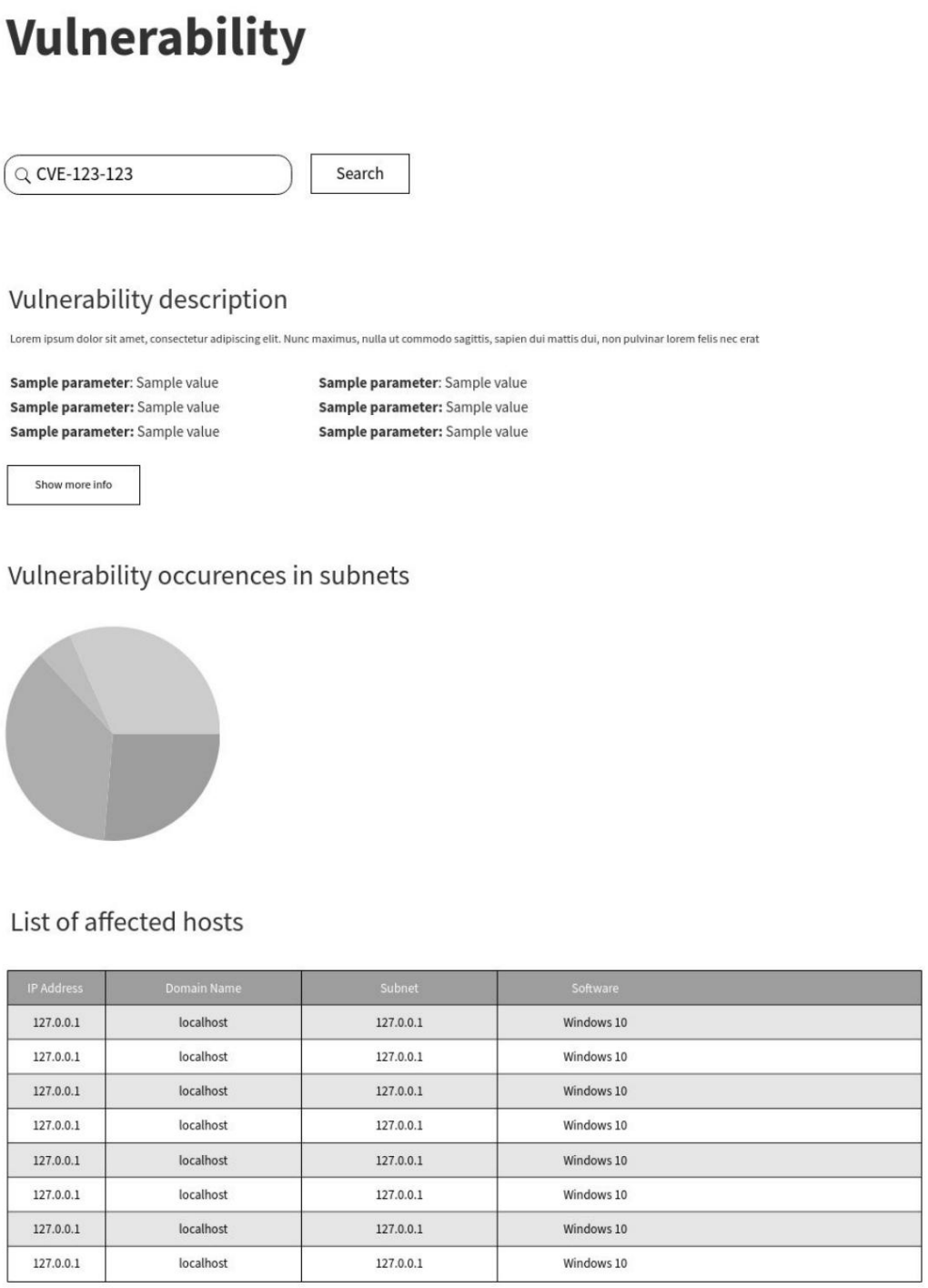


Figure 4: Wireframe of the Vulnerability panel.

Panel Decide/Act

The ***Decide/ Act*** panel should display information obtained from the web API tools of the Decide and Act phases, i.e. Outcomes No. 3 and 4 of the project. It is primarily about information about missions and elements of active defense (hereafter PAO). The user should be able to find out the status of PAOs, their utilization, the IP address and the port on which they are available.

For individual missions, their description and list of configurations should be available. There should also be an option for each mission to apply one of its configurations. Each configuration is defined by three attributes - confidentiality, integrity and availability. There should be a log on the panel, which will provide feedback to the user about whether the configuration was successfully applied, or inform him about possible failures. Applying configuration and getting feedback will be done using PAO web APIs.

The benefit for the panel would be to display a graph that would visualize the availability of individual missions. Individual missions are secured by information services, which can be further decomposed into supporting components. The connection of these nodes is expressed through ***AND/ OR*** nodes, which determine the conjunctive or disjunctive dependency on offspring.

The relationships between these entities and the availability of supporting components will be provided by the Web API of Outcome #3. Based on the availability of supporting components and the relationships of the ***Decide/ Act*** panel , it will generate a graph that will highlight in red all unavailable missions, information services and supporting components. The user will thus be able to quickly find out which missions are unavailable and what causes this unavailability.

Figure 5 shows a draft of ***the Decide/ Act panel.***

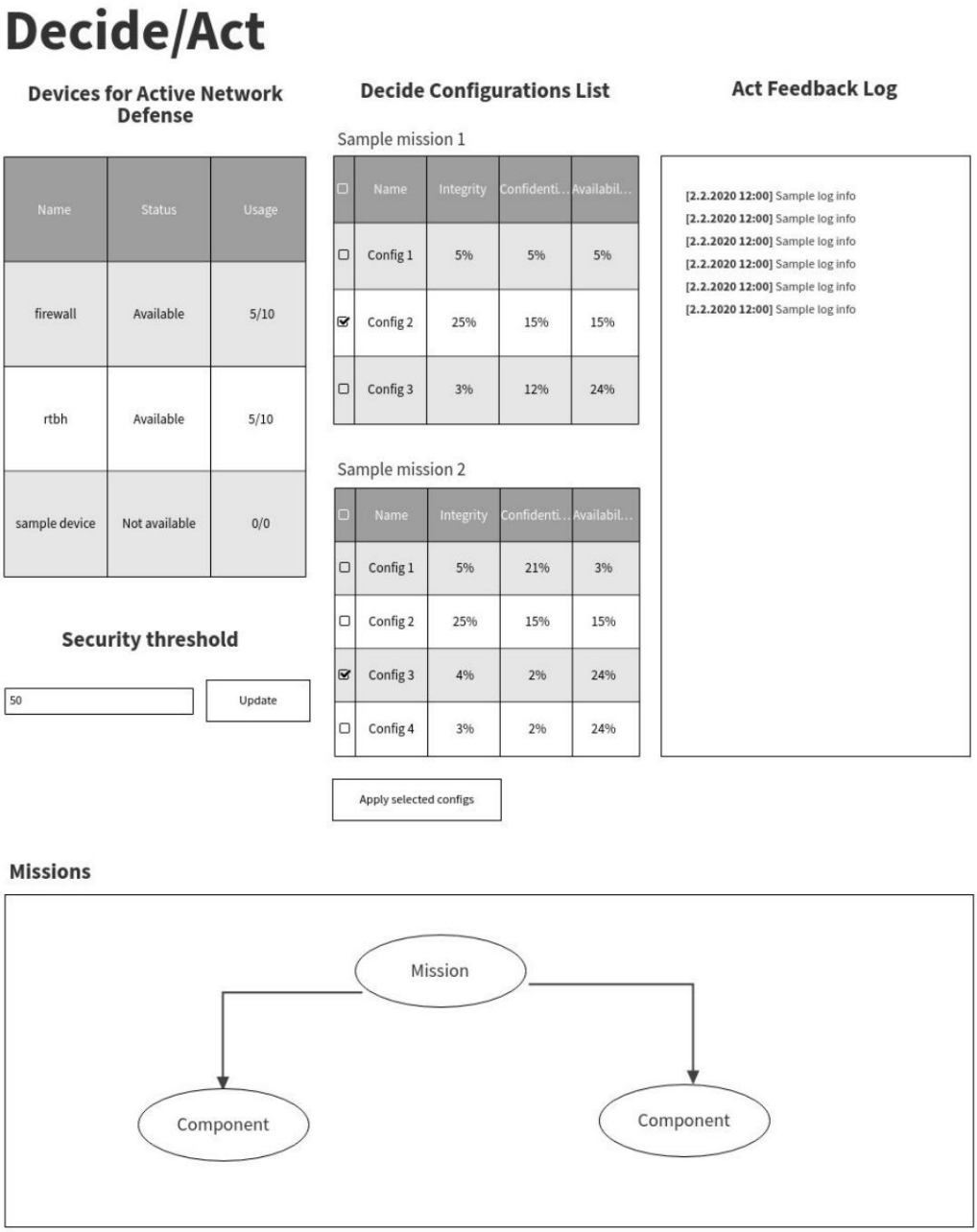


Figure 5: Wireframe of the Decide/Act panel.

Installation instructions

A web application for visualizing the security situation in a computer network can be launched in several different ways. For testing or development, we recommend using Docker or Vagrant. Startup procedures are covered in the first two sections of the installation guide. Installation instructions on a local system are described in the following section and are recommended for users who wish to install and run the application permanently, for example in a production environment.

Install and run with Docker

Installing and running a web application using Docker consists of creating a Docker image and running it in a container.

First you need to install Docker, we recommend following the instructions on the page:

https://docs.docker.com/get-docker/

Docker is available for all major operating systems.

Before building the Docker image, you need to make sure that the ***src/ environments/ environment.prod.ts*** file contains the correct API URLs.

The content of ***the src/ environments/ environment.prod.ts*** file looks like this: as

```
1. import *              packageData from '../../package.json';
2.
3. export const environment = { 4. applicationName:
'CRUSOE Dashboard', 5. production: true, 6. version: packageData.version, 7.


        /* Redirect API URL */
8. apiUrl: 'https://crusoe.muni.cz/redirect-api/redirect/', 9.
        /* Act API URL */
10. tmpActApi: 'https://crusoe.muni.cz/act/', 11. /* GraphQL API URL */ 12.
graphqlApi: 'https://crusoe.muni.cz/graphql-
api/graphql', 13. /* Firewall API URL */ 14. firewallApi: 'https://crusoe.muni.cz/firewall', 15.};
```

The example above assumes that the data is available through multiple different APIs on the ***crusoe.muni.cz machine.*** Individual APIs are set on lines 8, 10, 12 and 14.

A Docker image is created with the command:

```
$ docker build --no-cache -t crusoe_dashboard .
```

Next, the created Docker image needs to be run in the container with the command:

```
$ docker run --name crusoe_dashboard -d -p 4200:80 crusoe_dashboard
```

The web application can be accessed at:

http://localhost:4200/

Install and run with Vagrant

Installing and running a web application using Vagrant is all about creating virtual machine and installing the web application on the given virtual machine.

First, Vagrant must be downloaded and installed if it is not already on the machine installed. We recommend following the instructions on the project website. Installer files for all major operating systems are available at:

https://www.vagrantup.com/downloads

Next, you need to install Ansible if it is not already installed. Again we recommend follow the instructions on the developer's website available at:

https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html#installing-the-control-machine

Before running, you need to check that the API endpoints are set correctly. It is a need check the contents of the ***ansible/ playbook.yml*** file to see if it contains the correct URL. Content ***ansible/ playbook.yml*** file might look like this:

```
---
- hosts: all
    vars:
      # Dashboard folder
      dashboard_folder: /home/vagrant/dashboard

      # Redirect API URL
     redirectApi: 'https://crusoe.csirt.muni.cz/redirect-api/redirect/'
      # Act API URL
      actApi: 'https://crusoe.csirt.muni.cz/act/'
      # GraphQL API URL
      graphqlApi: 'https://crusoe.csirt.muni.cz/graphql-api/graphql'
      # Firewall API URL
      firewallApi: 'https://crusoe-worker.csirt.muni.cz/firewall'
# become: true
roles:
      - dashboard
```

Vagrant can then start and set up a virtual machine with a web application with the command:

```
$ vagrant up
```

After building and configuring the virtual machine, which can take several minutes, the web application is available at:

http://localhost:4200/

Local installation

A prerequisite is to install the Node.js package at least in version 10. We recommend getting Node.js from the developers' website and following the instructions at:

https://nodejs.org/en/download/

Verifying the installed version of Node.js can be done by entering the command:

```
$ node -v
```

The next step is to install the ***npm*** packages that contain the web application's dependencies. The ***npm*** package manager is usually installed together with ***Node.js.*** A list of web application dependencies is provided in the package.json file in the project root directory. To install them, just run the command in the root directory:

```
$ npm install
```

Similar to the other installation methods, it is necessary to verify the correctness of the endpoint settings for the API. The settings are located in the file ***src/ environments/ environment.prod.ts,*** or in the file ***src/ environments/ environment.ts*** if the project is managed by ***ng serve.***
However, typically the first option is valid, which is intended for production deployment, the second is relevant for development and test deployment. In this file, you need to set the URLs of the various APIs. A sample file is provided in the Docker Run tutorial, it is the same file.

The development server, which is used for development and testing, can be started with the command:

```
$ ng serve
```

Then just access the address:

http://localhost:4200/

A web application will be available at the address, which automatically reacts to changes in the source codes and is reloaded after each change.

To build the project you need to use the command:

```
$ ng build
```

Running the command will build the project and save it in the dist directory.

The production version of the web application can be built with the command:

```
$ ng build -- prod
```

Then move the assembled production version from the dist directory to the directory from which the application is to be loaded for the production environment, for example the web server directory.
The deployment of the production version can thus look like this:

```
$ rm -r /var/www/dashboard/*
$ cp -r ./dist/ /var/www/dashboard/ $ sudo /etc/init.d/
apache2 restart
```

First, the current contents of the web server directory are deleted (for example, if there is an earlier version of the web application), then the files are copied into the directory, and finally the web server (in this case, Apache 2) is restarted. The web application is then available at the address from the web server settings.

User documentation

The user documentation of the web application for visualizing the security situation in the computer network consists of five parts. First, access to the application and navigation in it are described, i.e. the process of logging in and managing users and navigating between panels in the application. In the next four parts, instructions for using individual panels with visualizations are described.
The first panel is the Task Manager used to manage data sources for visualizations. The second panel is the Network Visualizer, in which it is possible to browse the collected data, especially to find contextual information about entities in the network. The third is the Vulnerabilities panel, in which the occurrence of vulnerabilities in the network is visualized, including detailed information. The last panel is the Decide/Act panel, which contains visualizations and control elements of tools to support decision-making in the resolution of a security incident and the application of reactive measures on the elements of the active defense of a computer network.

Login and navigation

After opening the web application address, the splash screen captured in Figure 6 is displayed. The screen contains a Login button that redirects the user to the login screen. In the case of using OIDC authentication, the user is redirected to the login screen of the relevant identity management. After logging in, the user is redirected back to the web application, where they are presented with an initial screen with a list of panels in the menu on the left side of the panel and an area with tiles referring to individual panels in the rest of the screen. Clicking on a panel link in the menu or on a tile will display that panel instead of the tiles.
The menu on the left is preserved for navigation to other panels. In the upper right corner is the name of the currently logged in user. After clicking on the name, information about the user will be displayed. In the upper left corner is a button with a logo that directs the user to the splash screen.

Figure 6: Screenshot of the login screen.

Task Manager panel

The Task manager panel is shown in Figure 7 and is used to manage data collection tasks. The panel is connected to component monitoring systems and network data collection tools. The panel consists of four parts placed one below the other. In the upper part, the number of active, successfully completed, unsuccessfully completed and repeated tasks is visualized in colored rectangles. Thanks to this, the user has a quick overview of the current system load and whether the systems are working correctly. Below is the Workers table with a list of workers, tools that perform these tasks. Below the table, in the Tasks section, there is a graph of the number of tasks (more detailed data visualization from the top of the screen). The rest of the panel is made up of a task list table where you can browse, filter and search through the task records. After clicking on a record, a page with detailed information will open.

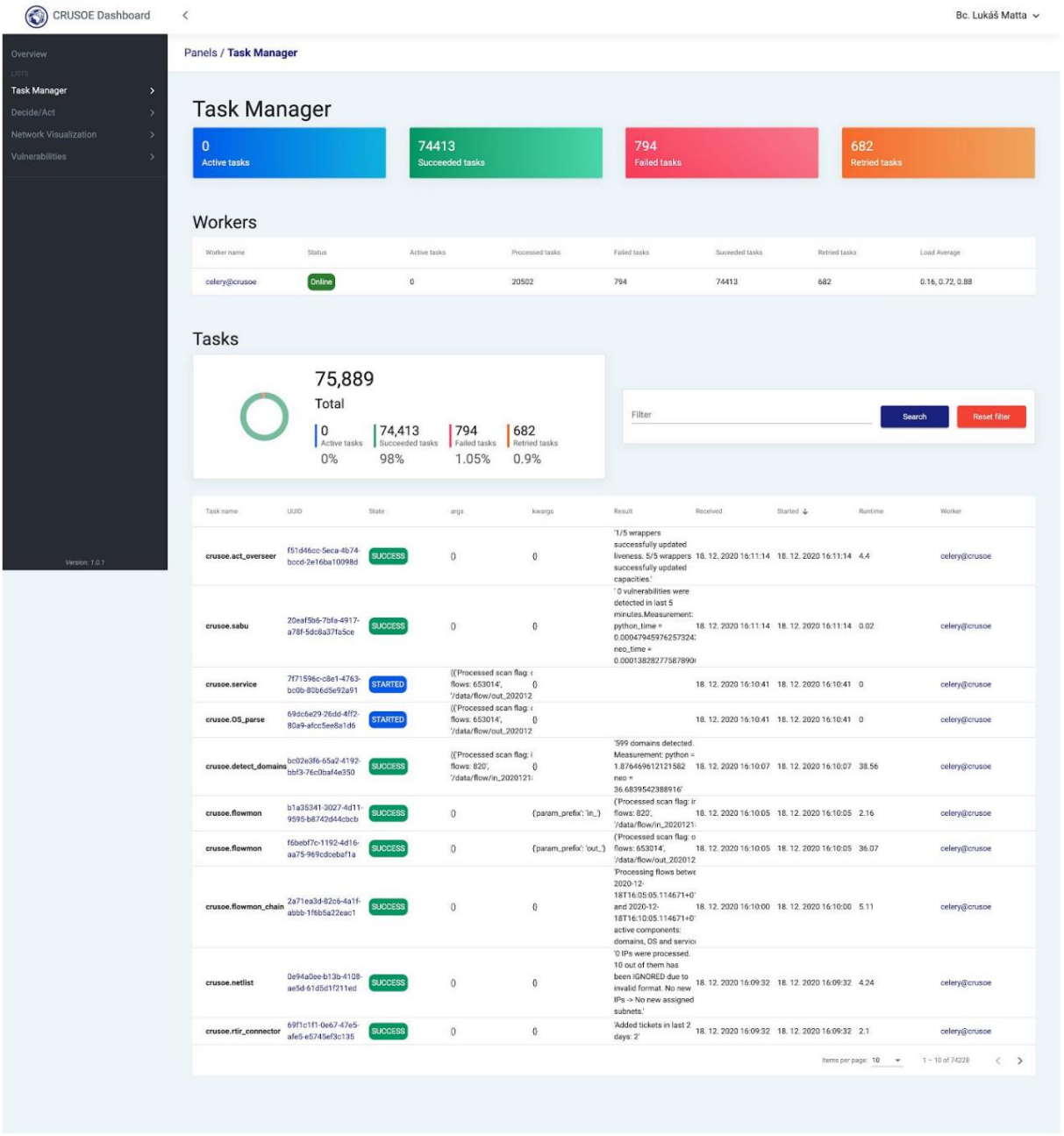


Figure 7: Screenshot of the Task Manager panel.

Network Visualization panel

The Network VIsualization panel, which can be seen in Figure 8, is used to visualize the content of the database and browse through the graph by which the data is modeled. Entities are modeled as graph vertices and relations as edges between them. Due to the large amount of data that is often located in the database, it is not advisable to display them all at once. The most common use case for this panel is to find contextual information about an entity in the network, most often an IP address. That is why there is a search box at the top of the panel for entering the IP address. After entering the IP address and pressing the Search button or confirming with the Enter key, the appropriate IP address is searched in the database and, if found, serves as the starting point for visualizations.

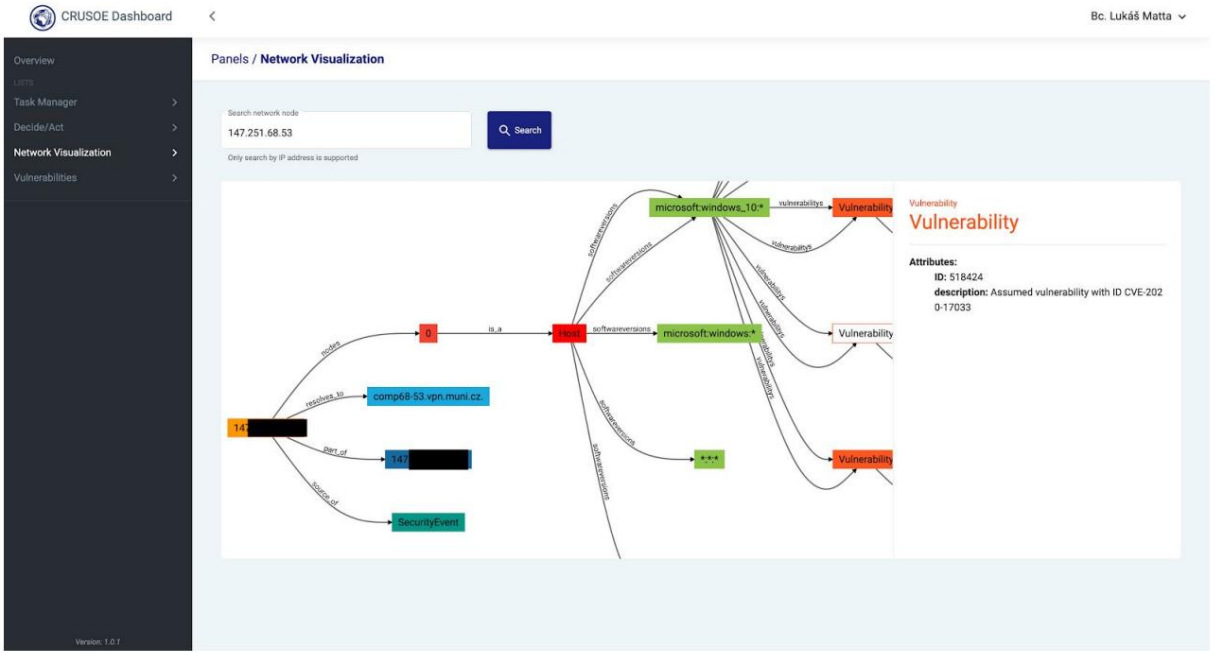


Figure 8: Screenshot of the Network Visualization panel.

The visualization occupies the rest of the panel. A graph is displayed in the white field, the central element of which is the searched IP address. Graph vertices are displayed as colored rectangles filled with a color corresponding to the data model layer in which the entity resides and the name of the entity. Edges between vertices are visualized by black arrows with a label corresponding to the name of the edge. The searched IP address is displayed on the left in the middle of the field for visualization. Furthermore, all entities that have an edge connecting them to a given IP address are selected from the database. These are drawn to the right of the IP address. After clicking on any vertex of the graph, a panel opens in the right part of the visualization area, on which all the information stored in the database as parameters of the given vertex is listed. It is possible to browse the graph further and access other adjacent objects. By double-clicking on a graph vertex, all vertices that have a common edge with the clicked vertex and the edges between them are added to the visualized graph. By repeating this action, it is possible to scroll through the graph and find more and more related information. A detail of such traversal of the graph is shown in Figure 9. Thanks to traversal and gradual loading, the size of displayed data is limited, focusing only on potentially relevant information. The user loads additional data as needed.

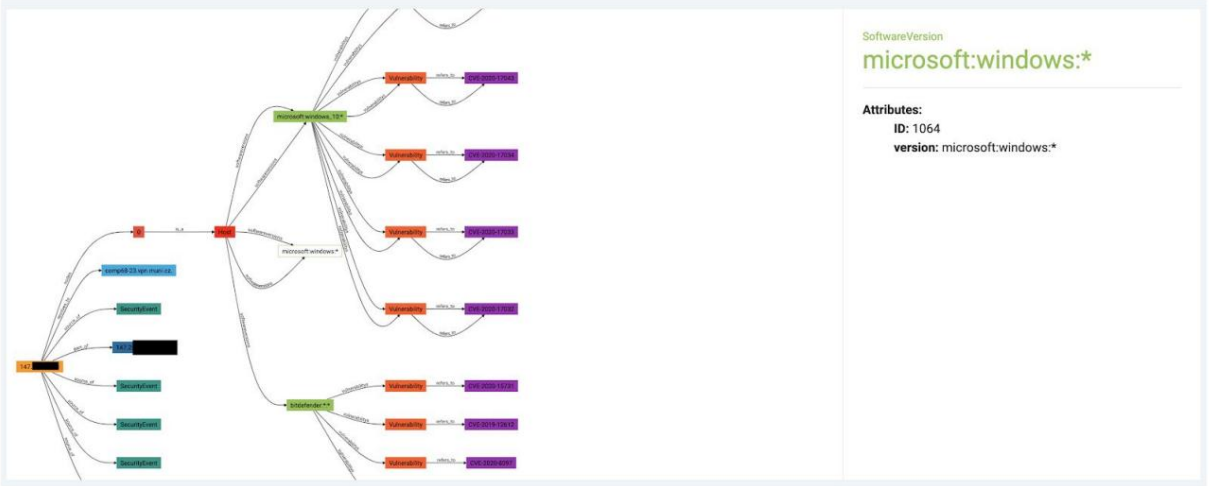


Figure 9: Detail of browsing the graph in the Network Visualization panel.

Typically, a user will search for an IP address that is currently compromised as part of an ongoing security incident. Thanks to the reporting of detection tools, for example, we know that the attacker is intensively scanning the given device. By looking up the IP address, the user finds out what kind of device it is, for example, that it is part of a critical infrastructure in one of the subnets. By browsing the graph, the user can access information about what software is running on the given device, for example the version of the operating system. By clicking on the version of the operating system, the user will then learn what the known vulnerabilities are for this version. Depending on the severity of the vulnerabilities and the importance of the device, the user can then decide on the appropriate action in response to the ongoing incident.

Vulnerabilities panel

The Vulnerabilities panel is used to search and visualize information about the occurrence of vulnerabilities in the protected network. The panel allows the user to search for a specific vulnerability by its CVE identifier, and if such a vulnerability is found, the information available in the database about that vulnerability and its occurrence in the network is displayed. The panel consists of several parts, which are described below. The entire panel is shown in Figure 10.

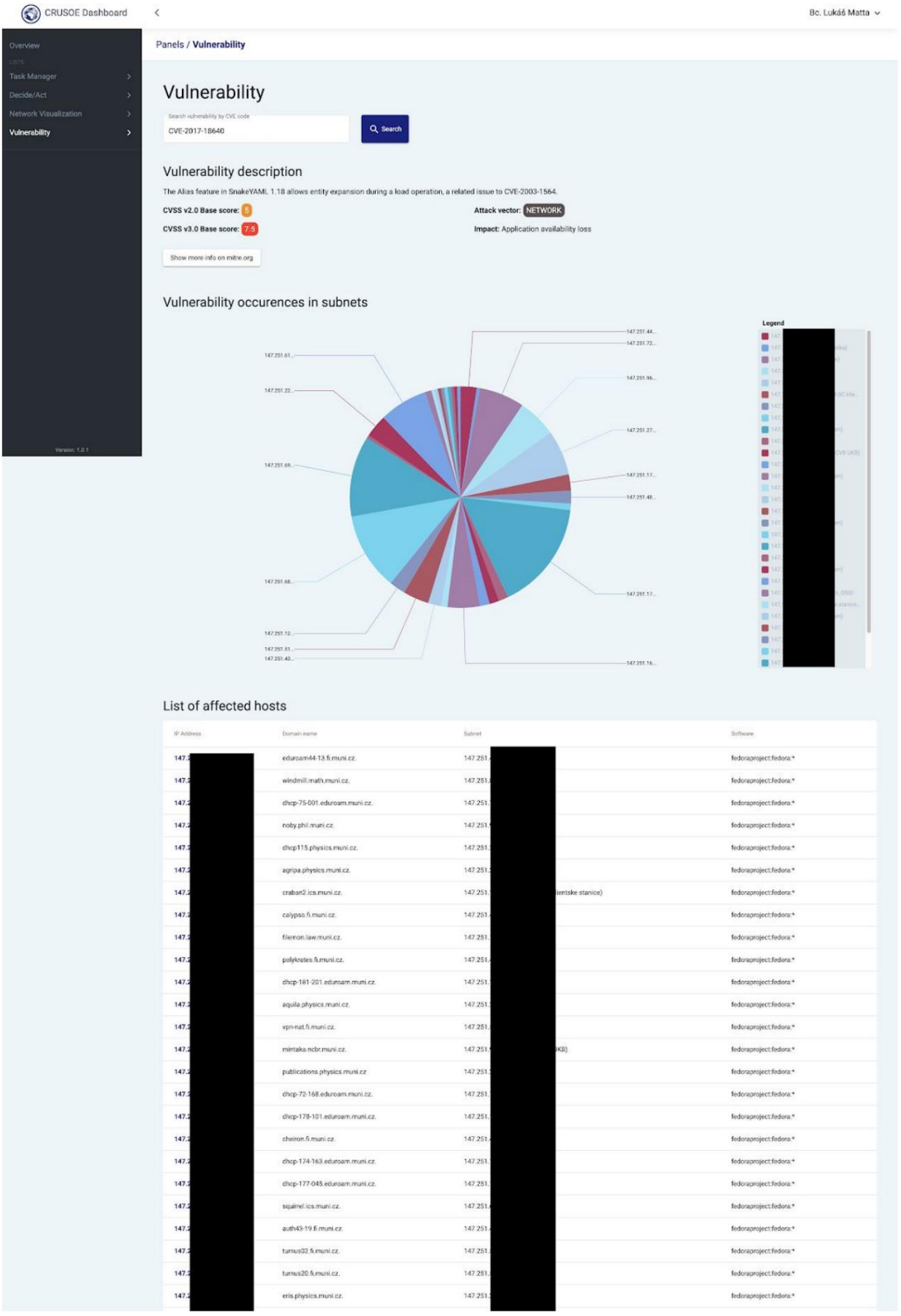


Figure 10: Screenshot of the Vulnerabilities panel.

The starting point for users is the vulnerability search box located at the top of the panel. The white field is provided with a hint with the expected input format, i.e. the CVE of the vulnerability identifier. The user enters the CVE vulnerability identifier and presses the button

Search lets the panel display the available information about a given vulnerability.

Just below the search field, selected information about the found vulnerability is displayed, especially its CVSS score and the attack vector that exploits the vulnerability. This information is obtained from the NVD database. Below the information is a button used to open the page dedicated to the given vulnerability on the website cve.mitre.org, where all detailed information about the given vulnerability is listed. If the user is interested in this information, clicking on the button will open the corresponding web page. However, the information used when working with the tools of the CRUSOE project is listed directly in the panel.

Below the vulnerability information is the first visualization, a pie chart of the occurrence of vulnerabilities in individual subnets of the protected network. The pie chart shows the number of devices on which the searched vulnerability is present or likely to be present. Each slice of the graph represents one subnet of the protected network. The pie chart is equipped with labels for easier orientation. A legend with full subnet names and the number of vulnerable machines listed is to the right of the pie chart.

The last element of the panel is the table in its lower part. The table contains a list of all vulnerable devices that are recorded in the database. In the table, for each record of a vulnerable device, the IP address and domain names of the given device, belonging to a subnet, and the identifier of the software that detected the vulnerability on the device are listed.

Panel Decide/Act

All informational and control elements of the Act software phase are collected on a single dashboard page. Because the Act phase software is closely linked with the Decide phase software, they also share this page. This section describes all the elements that can be used to control the Act phase software from the dashboard. The layout of the elements on the page is shown in Figure 11.

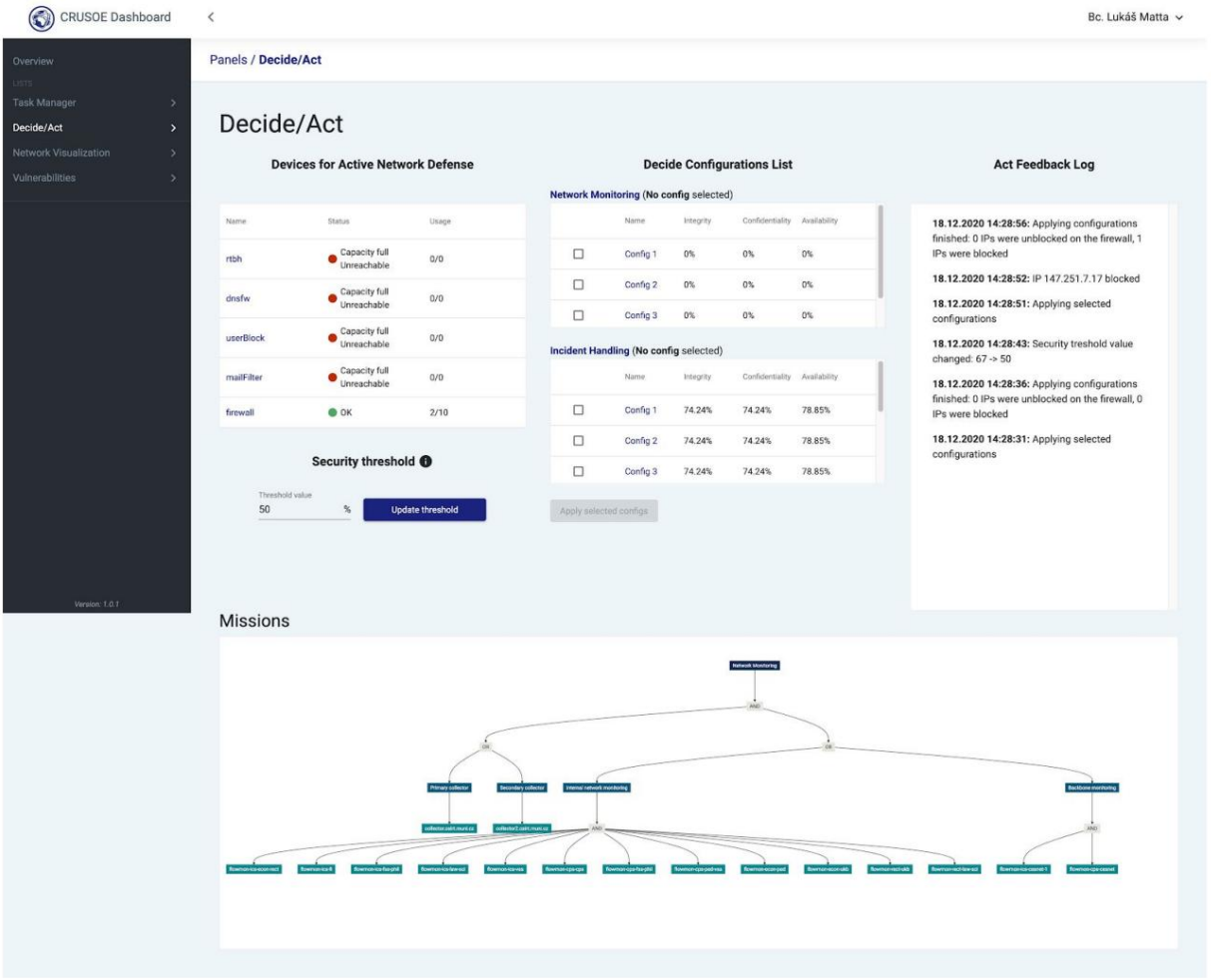


Figure 11: Screenshot of the Decide/Act panel.

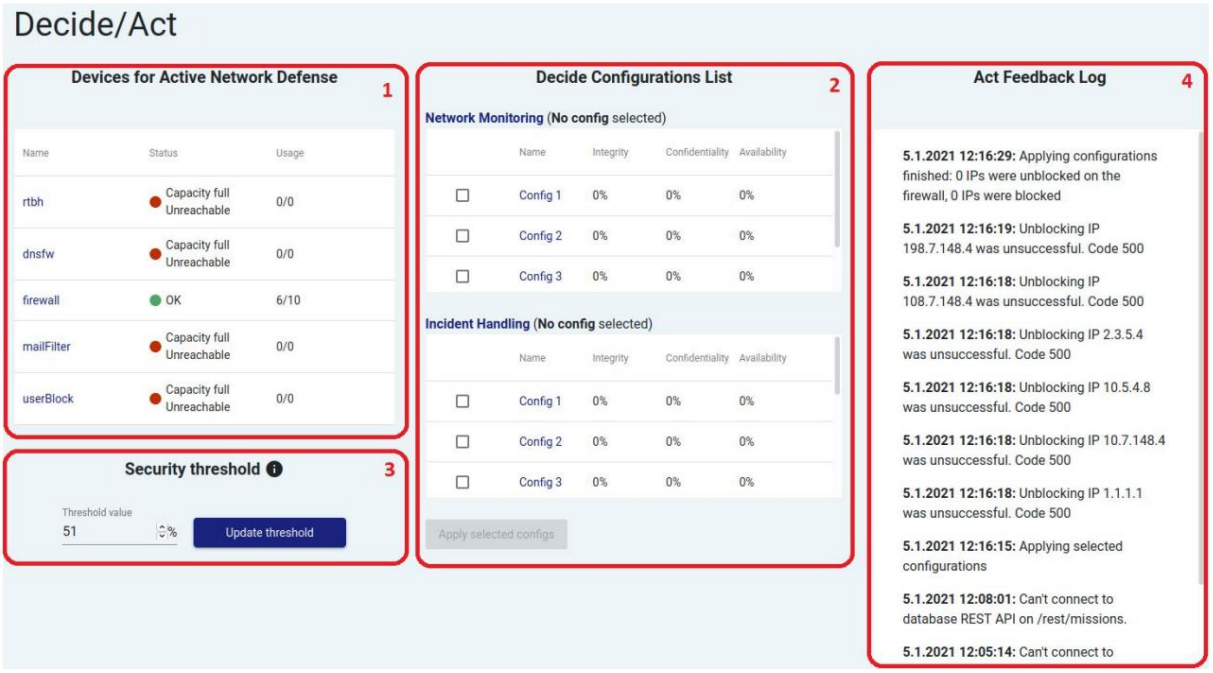


Figure 12: The Act phase software page in the dashboard.

Figure 12 highlights the division of panels according to their function.

1. List of active defense elements 2. List of missions and configurations defined in Decide 3. Current value of Security threshold 4. List of current events in the Act phase

List of active defense elements

The list contains a list of all active defense elements whose connection the software supports. It is used to get a quick overview of the status of these elements. The list panel is shown in Figure 13. It always contains the name of the element, summary information about its status, and capacity from the left. After clicking on the name of the element, its technical information is displayed - IP address and port on which the access interface of the element is running.

Figure 13: List of active defense elements with one element available.

For each supported element, the list shows a semaphore rating of its status, which depends on its current capacity and connectivity. The element can acquire three states (green, yellow, red) under the following conditions:

- Green - everything is normal.
  - OK - None of the conditions below are met.
- Yellow - the element works with reservations.
  - Capacity 90% full - 90% of maximum capacity detected.

- Last liveness check failed - The last contact was successful more than 10 years ago minutes.

- Red - the element has a problem that must be solved, otherwise it is unable to continue operating.
  - Capacity full - The used capacity of the element is equal to the maximum capacity. • Unreachable - Last contact was successful more than 30 minutes ago.

List of missions and configurations

The list of missions and configurations contains an inventory of all missions that the Decide phase software works with. A list of possible configurations is also given for each mission. A more detailed description of the missions and configurations is contained in the documentation of the Decide phase. The mission and configuration list panel is shown in Figure 14.

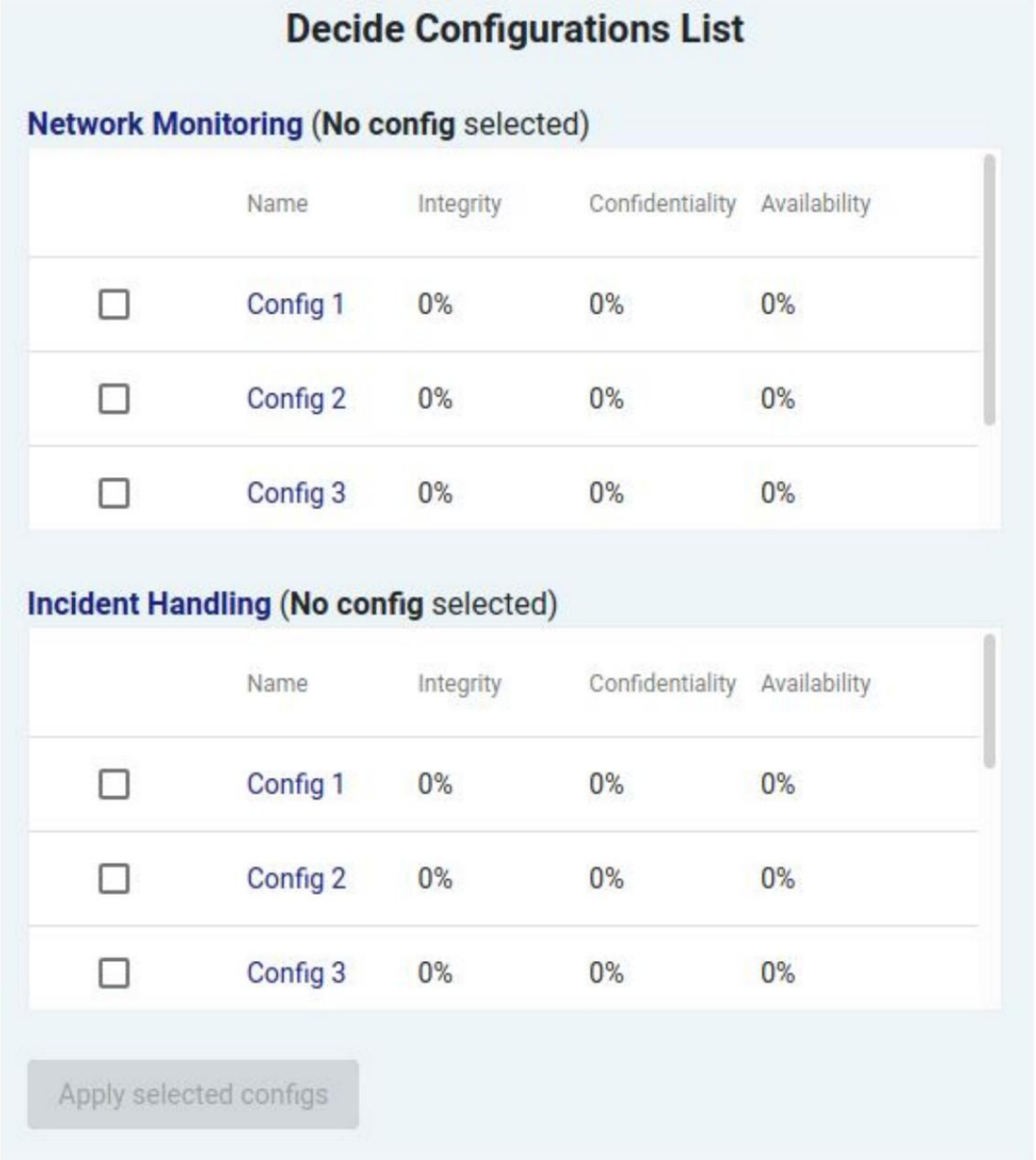


Figure 14: List of missions and configurations with two missions, Network Monitoring and Incident Handling.

Under the name of the mission, a list of possible configurations is always displayed along with the degree of threat for the three monitored parameters - integrity, confidentiality, and availability. By clicking on the name of the mission, a description of the given mission and its criticality value, i.e. the degree of importance for the organization, will be displayed. By clicking on the name of the configuration, a map is displayed on which the machines that the configuration requires to be blocked are marked in red.

The current value of the Security threshold

The security threshold indicates the threshold value for blocking the device. If this limit is exceeded for a device, the device is blocked, but only if it does not belong to the configuration chosen by the user. The Security threshold setting is shown in Figure 15.

Figure 15: Security threshold settings panel.

We can illustrate the situation with two examples. The mission has three configurations. The first two of them have zero values for all safety parameters and the third 75% for all safety parameters. The security threshold is set to 50. The user selects the first zero configuration. In this case, due to the zero rating of the second configuration, the devices belonging to the second configuration will not be blocked either, since its zero rating did not exceed the threshold. That's because they didn't unreasonably block devices on which we didn't detect any security flaws. Devices that exceed the threshold in the third configuration will be blocked.

The second illustrative case concerns a mission with three configurations. Let configuration 1 have zero rating for all security parameters, configuration 2 has 55% for all security parameters and configuration 3 has 68% for confidentiality, integrity and availability.
In this case, the administrator, based on his expert opinion, decides to choose configuration 2. For a threshold of 50%, none of the devices of configuration 2 will be blocked, even though they have exceeded the threshold. The configuration was chosen by the user. Devices with a zero rating (configuration 1) will also not be blocked. Other devices that have exceeded the threshold will be blocked.

List of current events in the Act stage

The current events list in the Act stage shows important events that are happening in real time. In particular, the progress and results of manually entered operations are displayed, as well as errors or warnings that caused these actions. An example of the panel contents when trying to apply configurations is shown in Figure 16.

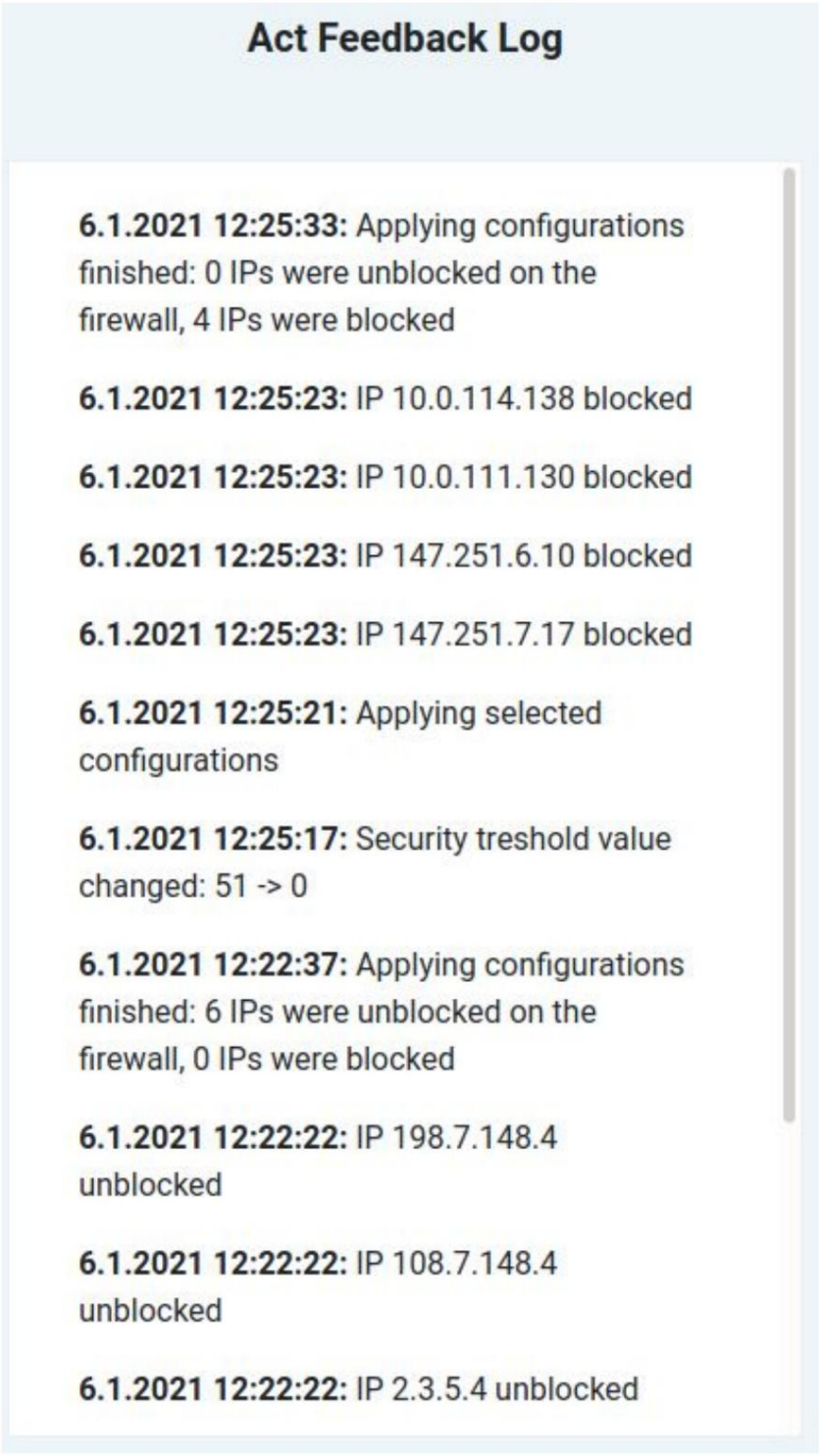


Figure 16: List of current events. The image shows (from bottom to top) the application of the configuration, the change of the Security threshold, and the re-application of the configuration with a different result.

Mission description visualization

The lower part of the panel is reserved for a clear visualization of the description of the missions that are recorded in the database. Missions are drawn in a hierarchical graph structure corresponding to the formal description of the mission model. In the relevant part of the panel, it is possible to zoom in and out of the mission visualization, which enables a clear view of the entire mission model and its individual details. The relevant part of the panel can be viewed in Figure 17.

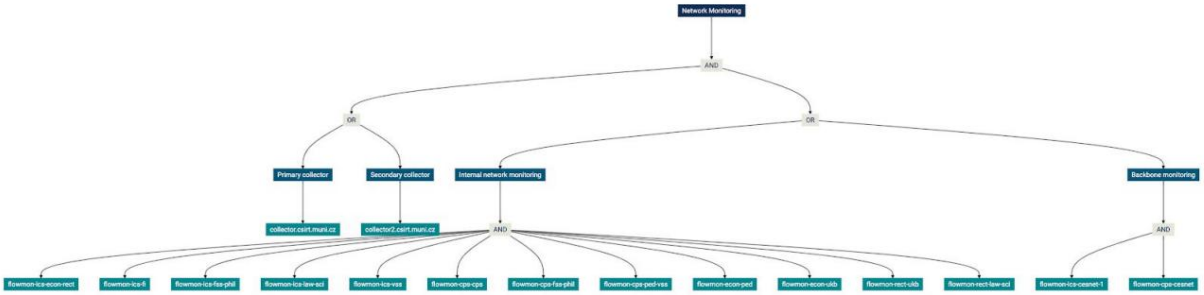


Figure 17: Visualization of the mission description.

Programming documentation

The programming documentation consists of two parts. First, the technologies used are described, and the principles of their use are described. The following is a section dedicated to the description of the system architecture. Subsequently, the implementation of the system is described, which includes a description of shared components and services and details on the implementation of individual panels. Notes on system testing and continuous integration are provided separately.

Technologies used

This section contains a brief description of the Angular framework along with other technologies related to Angular application development that were used to implement the visualization tool. The visual component style is based on the Bootstrap framework with the addition of Block-Element-Modifier (BEM) methodology to help achieve reusable CSS code.

Angular

Angular is a TypeScript-based JavaScript framework that provides tools for building highly interactive web and mobile web applications. It is an open-source software developed and maintained by Google. The visualization plugin described in this report is written in Angular version 7.[7]

Angular is a component-based framework that provides declarative templates offering a simple way to create highly flexible and reusable HTML + CSS components. In addition, it combines various programming concepts (eg dependency injection, two-way data binding, zero-step offline installation) that lead to cleaner code and rapid cross-platform development. Since both speed and performance are important for web application development, the framework also integrates several features (eg automatic code-splitting, lazy loading) that allow you to split the entire application package into several sections and load them only on demand.

Because Angular is a successful software tool, it is supported by a large community that regularly expands its features. The framework also provides extensive documentation, including tutorials with practical examples, which greatly simplify the transition to this technology.

TypeScript

Angular applications are written in TypeScript by [8] - statistically compiled language developed Microsoft with fully open source code. Typescript shares syntax and semantics with JavaScript and additionally adds several concepts from object-oriented languages (eg static typing, interfaces, classes, and modules).

[7] https://angular.io
[8] https://www.typescriptlang.org

TypeScript is primarily focused on solving problems related to the development of complex applications in JavaScript while maintaining all the advantages of the JavaScript environment. For example, introducing classes with public and private members helps maintain a clean structure and increases code reliability. Static typing and abstract interfaces address the lack of well-defined boundaries in plain JavaScript that often lead to too tight coupling. In addition, TypeScript also provides useful tools such as navigation, advanced auto-completion and refactoring.

Block-Element-Modifier (BEM)

To avoid the lack of front-end standards that often lead to unmanageable CSS classes, the implementation follows the BEM naming convention. This methodology helps reduce CSS [9] inheritance and style conflicts and increases code modularity and reusability. BEM stands for Block, Element and Modifier, which describes the three basic building blocks.

The key idea is to divide web content into logically and functionally independent components - blocks (eg menu, logo, search, login template) and represent each with a specific class attribute. Blocks serve as wrappers for elements - parts of blocks existing only in the context of their parent item (a menu item within a menu). Finally, modifiers act as flags that change the behavior or style of a particular element or block (an active menu item versus a passive one).

Bootstrap

The design of the visualizations is created using the Bootstrap framework[10]. A popular open-source front-end library used primarily for creating visually appealing websites and applications.
It was originally developed as an internal tool at Twitter and only later released for free use.
Bootstrap offers many easy-to-use features for rapid prototyping, including prebuilds, interactive components, layout tools, mixins, and jQuery plugins. It also supports all advanced browsers.

Visualization libraries

Angular Material UI is[11] an open-source component library that is developed by the community in collaboration with the Angular team. Individual components implement the specification defined by Google. Emphasis is placed[12] on their usability, while each component must also contain definitions that enable its use in special browsers designed for the visually impaired or blind. The components also maintain a uniform design and controls.
At the time of publishing Orient Software, the latest stable version of Angular Material UI was version 11.0.2, which contains 33 different components.

[9] http://getbem.com
[10] https://getbootstrap.com
[11] https://material.angular.io/
[12] https://material.io/components

ngx-charts is an open-source library providing several different types of charts. The main developer is the American company Swimlane. At the time of publishing Orient software, the latest stable version was version 16.0.0. The library is built on the well-known JavaScript visualization library D3.js and its goal is to facilitate the creation of graph visualizations in the Angular environment. In Orient, a pie chart comes out of this library.[13]

Ngx-graph is an open-source library built on top of the ngx-charts library and allowing the creation of graphs that contain vertices and edges. At the time of publishing Orient software, the latest stable version was version 7.1.2. In Orient, this library is used to visualize network nodes and elements connected to them.

REST

REST (REpresentational State Transfer) is an architectural style of web service interfaces introduced in 200 and widely used since then. It is important to note that this is not a clearly defined specification, but a style. An application programming interface (API) that follows the REST style is called a REST API.

A key element in the REST style is the so-called resource, which is worked with using a web service. It is possible to use various methods of the HTTP protocol (GET, POST, PUT, DELETE, etc.) to work with this resource. Each resource is defined by a URI (Uniform Resource Identifier). The REST approach is suitable in cases where resources can be easily defined.
Its use is preferred especially in connection with relational databases, but it can also be used elsewhere.

OpenID Connect authentication

For authentication purposes, I use the OpenID Connect standard in Orient. OpenID Connect (OIDC) is an extension of the OAuth 2.0 authorization protocol with an authentication layer. OIDC supports several different authentication schemes, Orient uses the ***Implicit Flow scheme.***

The principle of operation of the ***Implicit Flow*** scheme can be described in three steps:

- In the first step, the user is redirected to the single sign-on page operated by the OIDC administrator in the organization. The operator is also referred to as ***an OIDC provider.*** On this page, the user fills in his login information.
- After filling in the login information and successfully logging in to the page you operate OIDC, the user is then redirected to the address of the web application, while a Bearer type access token is sent in the ***GET*** parameter .
- The web application processes this access token and stores it in the browser's local memory. The web interface then uses this token when authenticating against endpoints that support OIDC authentication.

Orient uses an open-source library supporting OIDC called angular-oauth2-oidc .[14]

[13] https://d3js.org/
[14] https://github.com/manfredsteyer/angular-oauth2-oidc

Implementation description

The implementation description consists of seven parts. First, the resulting architecture of the system is summarized, including its division into components. The following is a description of the implementation of shared components and services such as logging in with OIDC and navigation between panels. The use of the Angular framework, on which the entire application is built, is described separately. The remaining sections describe the four visualization panels.

System architecture

The system consists of several parts, the main part is the Angular application, the other parts are various application interfaces. The architecture consists of the following components:

- Angular application - Angular application, the main part of the system. •
- Redirect API - The API was created as an intermediate link to facilitate the communication of the Angular application with other tools implemented within the project. It is secured by OIDC authentication. Because some APIs are protected by ***Basic HTTP*** authentication , it was necessary to establish this API on which credentials can be securely stored. • neo4j-rest - the API was created to access data from the
- Neo4j database, it partially meets the architectural style of REST. • Act API - API of tools implementing the Act phase in the CRUSOE
- project, serves for
  applying configurations and obtaining information about active defense elements.
- Flower API - The API provides information about running jobs and running workers
  of the Celery system.
- Neo4j database - Neo4j graph database, Angular application gets data from it
  through various APIs.

The main part of the work is a SPA application created using the Angular framework in version 11.0.5, the last stable version of the Angular framework available at the time of the Orient software release. In addition to ***CRUSOE Orient,*** the tool's internal designation is ***CRUSOE Dashboard.*** The application code is composed of the following programming languages: TypeScript 57.9%, HTML 28.9%, SCSS 11.6%, JavaScript 0.9% and CSS 0.4%. The initial skeleton of the application was created using the Angular CLI tool.

An Angular application typically consists of various modules, components, and services. A simplified file structure of the application looks like this:

ÿÿÿ authentication ÿÿÿ
components ÿÿÿ
services ÿÿÿ
panels ÿÿÿ
decide-act ÿÿÿ my-
account ÿÿÿ network-
visualization ÿÿÿ panel-overview
ÿÿÿ task-manager

ÿÿÿ vulnerabilities

ÿÿÿ shared

ÿÿÿ components.
ÿÿÿ config.
ÿÿÿ interceptors.
ÿÿÿ models.
ÿÿÿ services.

The ***authentication*** directory contains authentication-related components (login screen, logout button, user profile) and a service that provides authentication using the angular-oauth2-oidc library.

In the ***panels*** directory , components and services are divided according to their affiliation to individual panels, which can be accessed via the navigation located in the left part of the screen.

The ***shared*** directory contains code that is used across the entire application, especially components such as the upper panel containing the name of the application and the name of the logged-in user, navigation between panels, and the like. The directory also contains configuration files, interceptor (Interceptor is a class that implements the HttpInterceptor interface and ensures insertion of authorization headers in HTTP requests), models and services.

Shared components and services

The following components and services are used across application components. It is about:

- The **Navbar** component is a blue bar located at the top of the screen. It contains the name of the application, the name of the currently logged in user, a logout button and a button to view the user's profile.
- The **Sidebar** component is located in the left part of the screen and contains a menu through which the user can navigate between individual panels.
- The **AuthService** provides methods to verify that a user is authenticated and retrieve user data through OIDC. The service uses the angular-oauth2-oidc library.
- The **ApiService** provides public methods for communicating with the API. Application obtains and sends data mainly using the methods of this service.
- The **ApiInterceptor** service implements the HttpInterceptor interface. This service enables the automatic insertion of an authorization token into the header of each sent request. Therefore, there is no need to insert the header separately each time the request is sent.

Using the Angular framework

Angular is a framework for creating SPA (Single Page Application) web applications.
A SPA is a type of web application that interacts with the user by dynamically loading and rewriting elements on the current web page. Thus, the entire page is not loaded as in standard web applications, where the entire code is loaded when the page is changed.
After the initial loading of the SPA, all the code necessary for the actual operation of the web application is available. When interacting with the user, the necessary data is obtained from various APIs and

their subsequent display is implemented using JavaScript. The advantage is a faster transition between subpages and getting closer to the behavior of a native application.

The development of the Angular framework is led by a team that is part of Google, but the public can also participate in the development, since it is an open-source framework. The source code is available from the public repository. The Framework uses the Typescript programming language from Microsoft. Source files written in Typescript are transpiled into JavaScript before being run in a web browser, whereby type checking takes place during this transpilation. The programmer thus gets the benefits of type checking for JavaScript as well.

The Angular framework, also referred to as Angular 2+ or Angular v2 and higher, is the successor of the AngularJS framework, also known as Angular 1. At the time of the release of the Orient tool, the current stable version of the framework was version 11.0.3.

The source code of a web application built using the Angular framework typically consists of three main components: modules, components, and services. Part of Angular is the dependency injection system, which facilitates the use of services.
The use of these components is now described in more detail.

Modules

Modules are the main building block of an Angular application, we also call them ***NgModules.***
The ***Ng*** prefix is an abbreviation for Angular, often used in conjunction with libraries for Angular. We create the module as a new class, which we mark with the ***@NgModule decorator.***
The convention is for a class to have the suffix Module in its name . We insert a metadata object with the following attributes into the ***@NgModule*** decorator:

- declarations - a sheet that defines the components, directives and available pipes in the module,
- imports - list of external modules that we import into this module, • providers - list of available services in the given module, used for the insertion system addiction,
- bootstrap - we use the attribute exclusively in the root module and define the main component in it, which is loaded into the ***index.html file during initialization.***

So a class definition with a decorator for an Angular application module might look like this:

```
1 @NgModule ({ 2 // the
module contains the ChartComponent component 3 declarations :
                                    [ ChartComponent ],
4 // the module imports the Angular module for working with the form 5 imports :
                                [ FormsModule ], 6 // the
module exports the ChartComponent component 7 exports : 8 // the module
provides the                    [ ChartComponent ],
DataService class 9 providers : [DataService],
```

[15] https://github.com/angular/angular

```
10 // empty because it is not a root module 11 bootstrap : [] 12 })

13 export class ExampleModule {}
```

We divide modules according to the way they are used into several categories: domain modules, routed modules, routing modules, service modules, widget modules and shared modules. Modules help organize and decompose code, and can also be used for easier sharing of functionality across different Angular applications.

Components

Every Angular application contains at least one root component, from which the component hierarchy is derived to the Document Object Model (DOM). Typically, an Angular application consists of several components that communicate with each other. Such decomposition allows us to reuse components in different places of the application or even move the component to another Angular application.

We will create the component as a new class, the convention is that this class should have the suffix ***Component in its name.*** Before defining this class, we add the ***@Component decorator.*** The ***@Component*** decorator requires as input a metadata object with the following attributes:

• selector - string with which it will be possible to add a component to the template, • templateUrl - path to ***the .html*** file that represents the component template, • providers - list of available services for the given component, used for the insertion system addiction.

In the template, it is possible to access the methods and attributes defined in the component class.
This way we separate the logic of the component from what the user sees (the template).

Angular creates, updates, and destroys components based on user activity. An example of a simple component definition looks like this:

```
1 @Component ({ 2 // the
component will be available as <app-people></app-people> 3 selector : "app-people", 4 // relative path to
the template

5 templateUrl : "people.component.html", 6 // the component provides
the PeopleService class 7 providers : [PeopleService], 8 }) 9 export class
PeopleComponent implements OnInit { 10 people:

Person[]; 11 selectedPerson: Person;


12
13 constructor(private peopleService: PeopleService) {                                                    }
14
15 // This method is called by Angular when the component is created 16 ngOnInit() {
```

```
            this.people = this.peopleService.getPeople();
17 }
18 selectedPerson(p: Person) { this.selectedPerson = p; } 19 }
```

Angular supports two-way data binding. In practice, this means that if we change a value in the component class, the change is reflected in the template and is automatically visible to the user as well. Conversely, if the user changes a value in the template, for example by typing text into a form field, the change to the value in the template is propagated to the component class. Figure 18 illustrates two-way data binding between the template and the component class, labeled as ***property binding*** in the direction to the template and ***event binding*** in the direction to the component class .

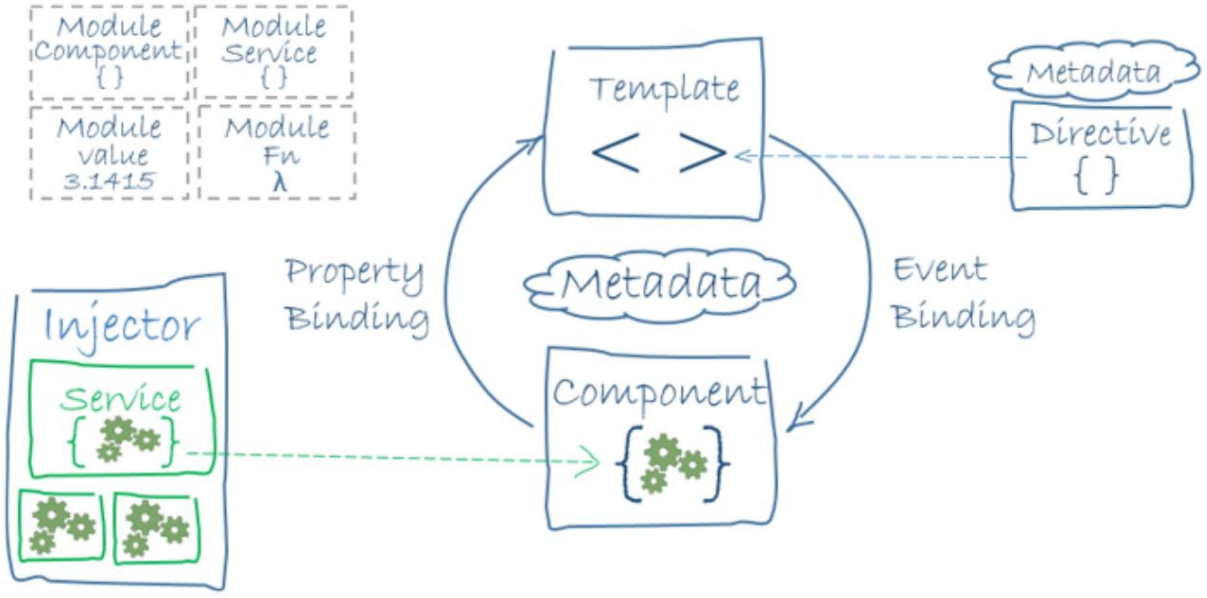


Figure 18: Diagram of connection and communication of individual parts of the Angular application.

Services and Dependency

Injection A service in an Angular application is a class that has a clearly defined purpose. It thus helps to simplify component classes, which should only contain logic directly connected to the given component. Services are often shared between different components, ensuring communication with external APIs, data modification and other tasks.

When communicating with external resources, we often use the ***HttpClient*** service provided by Angular itself. The service allows you to send HTTP requests with which we send or receive data. We then process the data using the RxJS library, which uses the ***Observer design pattern.*** [16],

The name of the service class should have the suffix ***Service,*** we add the @Injectable decorator before the class definition . An optional parameter of this decorator is a metadata object with an attribute

[16] https://rxjs-dev.firebaseapp.com/guide/overview

providedIn, which we specify to what extent the service will be accessible through the dependency injection system. The attribute can take on the following values: • root - the service is accessible
across the application as a singleton (a design pattern in which
a single instance of the class available in the entire program)
• platform - the service is accessible across multiple Angular applications defined in one web interface, • any - for each module loaded by
the ***lazy load*** system (a system in which modules are loaded only after the user visits the URI that requires them)
a unique instance, a ***singleton is shared for other modules.***

If we want a unique instance of the service class to be available for each component or module, we add the class to the ***providers list,*** which was mentioned when describing the ***@NgModule*** and ***@Component decorators.***

The class that will be available to the entire application as ***a singleton*** is defined as follows:

```
1 @Injectable({ 2 // The
instance of the trid will be available via the root dependency injector 3 providedIn: 'root' 4 }) 5 class TaxiService
{ 6 // Shortened notation of the

constructor in TypeScript, the so-called

    constructor shorthand
7       constructor(private service: TelephoneService) {}
8 }
```

Task Manager panel

The Task Manager panel is used to manage tasks and workers of the Celery system. It uses the API provided by Celery's web-based monitoring and management tool called Flower. A separate service class was created in the task-manager directory to communicate with the Flower API .
The Flower API itself, in its original version, did not allow simple task retrieval.
As this is an open-source project, the author was contacted and the necessary functionality was added directly to the Flower API. The change was approved by[17] the author of Flower API and added to the production version 0.9.7.
Communication with the Flower API takes place through the Redirect API, which ensures authenticated access.

The Task Manager panel contains two subpages: task detail and worker configuration detail.
These subpages are made up of separate components.

The requirements for the Task Manager panel also included the requirement that the user be informed about changes in the performance of tasks continuously without the need to refresh the page. This functionality was implemented by periodically sending HTTP requests to the Flower API. Thus, without the need to refresh the page, the panel header shown in Figure 19 and the task pie chart shown in Figure 20 are continuously updated. The pie chart is implemented using the ***ngx-charts library.***

---

17 Commit available at https://github.com/mher/flower/commit/11523ca0

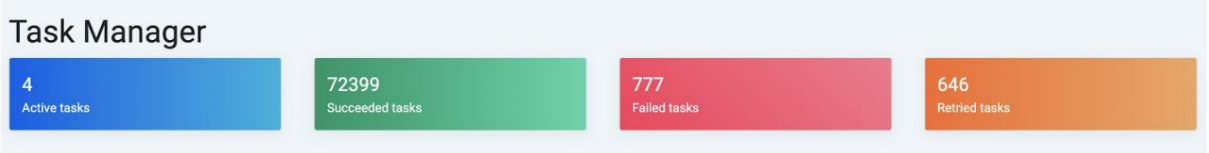


Figure 19: Subpage header in the Task Manager panel.

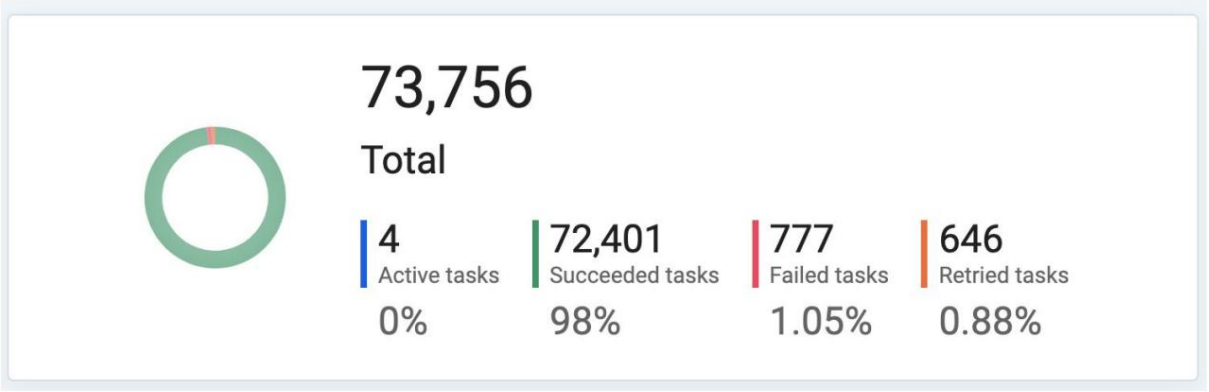
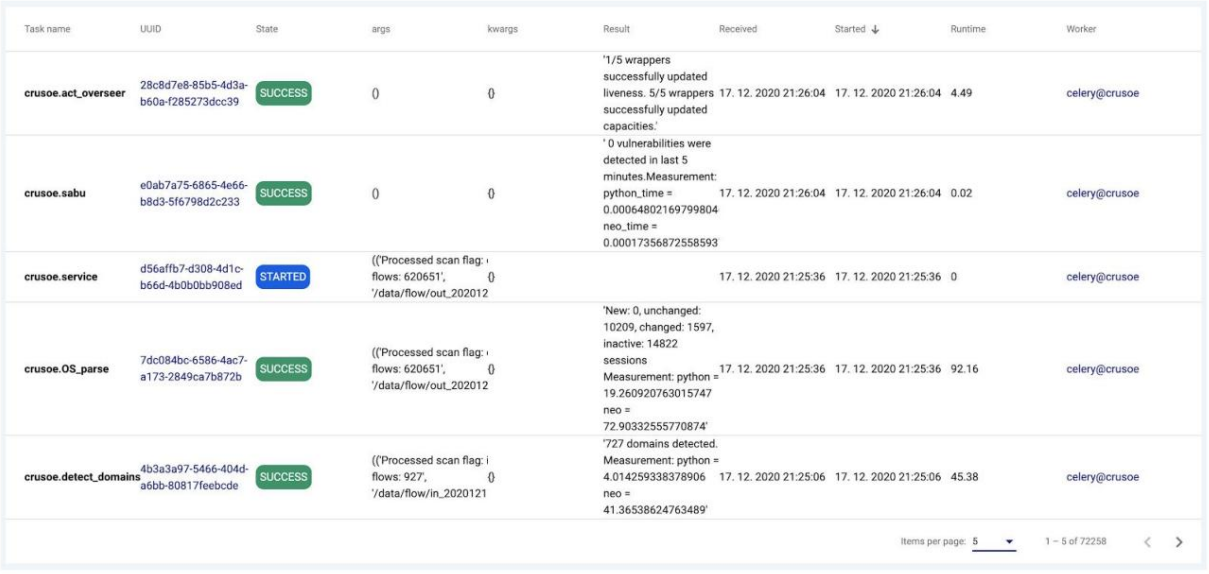


Figure 20: Pie chart of tasks in the Task Manager panel.

The task list table shown in Figure 21 allows the user to click through to a sub-page providing more detailed information about the given task. It was implemented using the ***mat-table*** component of the Angular Material library . Allows you to sort tasks according to the start time of their execution in ascending and descending order.



Task name	UUID	State	args	kwargs	Result	Received	Started	Runtime	Worker
crusoe.act_overseer	28c8d7e8-85b5-4d3a-b60a-f285273dcc39	SUCCESS	()	{}	'1/5 wrappers successfully updated liveness. 5/5 wrappers successfully updated capacities.'	17. 12. 2020 21:26:04	17. 12. 2020 21:26:04	4.49	celery@crusoe
crusoe.sabu	e0ab7a75-6865-4e66-b8d3-5f6798d2c233	SUCCESS	()	{}	' 0 vulnerabilities were detected in last 5 minutes.Measurement: python_time = 0.00064802169799804 neo_time = 0.00017356872558593	17. 12. 2020 21:26:04	17. 12. 2020 21:26:04	0.02	celery@crusoe
crusoe.service	d56affb7-d308-4d1c-b66d-4b0b0bb908ed	STARTED	(('Processed scan flag: flows: 620651', '/data/flow/out_202012	{}		17. 12. 2020 21:25:36	17. 12. 2020 21:25:36	0	celery@crusoe
crusoe.OS_parse	7dc084bc-6586-4ac7-a173-2849ca7b872b	SUCCESS	(('Processed scan flag: flows: 620651', '/data/flow/out_202012	{}	'New: 0, unchanged: 10209, changed: 1597, inactive: 14822 sessions Measurement: python = 19.260920763015747 neo = 72.90332555770874'	17. 12. 2020 21:25:36	17. 12. 2020 21:25:36	92.16	celery@crusoe
crusoe.detect_domains	4b3a3a97-5466-404d-a6bb-80817feebcde	SUCCESS	(('Processed scan flag: flows: 927', '/data/flow/in_2020121	{}	'727 domains detected. Measurement: python = 4.014259338378906 neo = 41.36538624763489'	17. 12. 2020 21:25:06	17. 12. 2020 21:25:06	45.38	celery@crusoe

Items per page: 5 1 – 5 of 72258

Figure 21: List of tasks in the Task Manager panel.

Network Visualization panel

The Network Visualization panel allows the user to search for devices on the network based on their IP address. This device is subsequently displayed as a node in the graph, whereby its neighboring nodes describe this device in more detail, or they add context to a given device. Neighboring nodes are: the subnet to which the device belongs, the assigned domain name, the security events associated with it

the device and the software that resides on the device and has been detected, for example by tools implementing the Observe phase. Nodes representing installed software include nodes describing vulnerabilities found on this software.

The graph itself is implemented using the ***ngx-graph*** library in version 8.0.0-rc.1. To obtain data, the panel uses the REST API, to which it sends a request in which the required data is explicitly listed, i.e. the nodes and their associated edges from the Neo4j graph database, in which the data is typically stored. The data is then processed and divided into two array type variables representing the edges and nodes of the graph.

Vulnerability panel

The Vulnerability panel allows the user to verify the occurrence of vulnerabilities in the managed network. On the panel there is a search form in which the user enters the CVE code of the vulnerability whose presence in the network he wants to verify. After pressing the ***Search*** button , two requests are sent to the REST API. The first verifies whether the given vulnerability is in the database. If a vulnerability is known in the system (information about it is stored in the database), details about the given vulnerability are sent from the database, which were downloaded to the database, for example, from the NVD database or another source of information about vulnerabilities. The second request is used to obtain a list of devices on which the given vulnerability was found.

Basic information about the vulnerability is displayed in a clear statement, which is shown in Figure 22. The statement also includes a button with a click through to the page with complete information about the given vulnerability listed on the website cve.mitre.org.

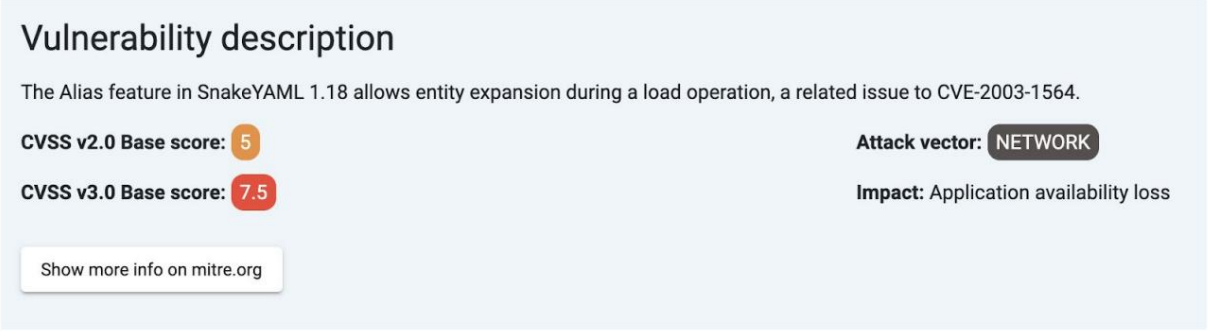


Figure 22: Vulnerability information listing.

A pie chart visualizing the number of vulnerability occurrences in the subnets of the monitored network allows the user to quickly get an overview of which subnet is most vulnerable to the vulnerability. This chart is shown in Figure 23 and implemented using the ***ngx-charts library.***

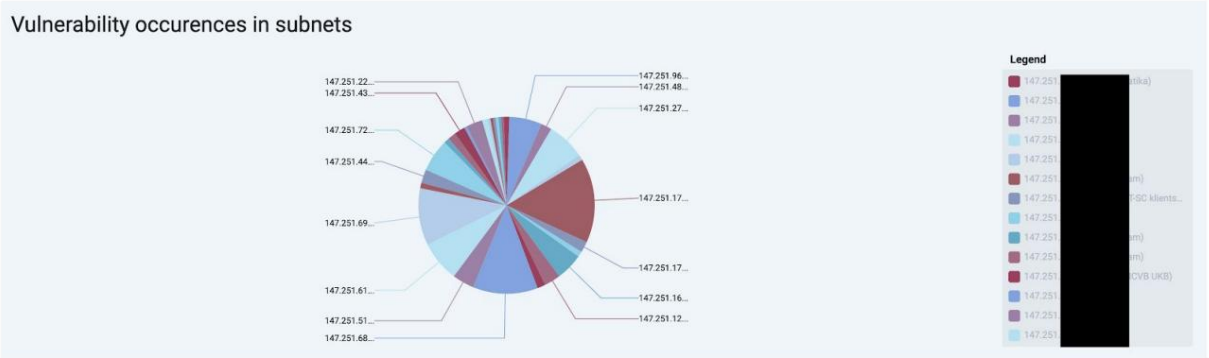


Figure 23: Visualization of vulnerability occurrence in subnets.

The user also has a table made up of individual devices on which the sought vulnerability occurs. The table consists of four columns: ***IP Address, Domain name, Subnet, Software. The IP Address*** column contains the IP address of the machine on which the given vulnerability was found, with the option of clicking through to the Network Visualization panel, on which, after clicking, you can see the given device with the relevant surroundings in the form of a graph. ***The Domain name*** column contains the domain name corresponding to the IP address of the device, and the ***Subnet*** column contains the address of the corresponding subnet in CIDR format. In the ***Software*** column , there is the name of the software that contains the searched vulnerability and was detected on the given device by one of the detection tools.

Panel Decide/Act

The Decide/Act panel allows you to get an overview of the organization's missions, the availability of individual missions, and also allows the user to apply mission configurations as recommended by decision support tools. The panel is decomposed into several different components. The main component is the ***DecideAct component.***

One of the requirements was to allow the user to gain an overview of the active defense elements. This information is displayed in the ***DecideAct component,*** which consists of a table from the ***Angular Material library.*** The table contains three columns: ***name, status*** and ***usage.*** The user has the option to click on the name of the element and detailed information about the relevant active defense element will be displayed. This detailed view is made up of a separate ***Pao component,*** which makes it possible to easily extend this component with additional information in the future.

Another requirement was to allow the user to set the ***security threshold value.*** This function is provided by a simple form with a button, after setting the value, the user receives feedback by displaying a small panel at the bottom of the screen. This panel is made up of the ***MatSnackbar*** component of the Angular Material library .

Another requirement was to allow the user to apply different mission configurations, with the user being able to find out more detailed information about those configurations. In the center of the screen there is a list of missions, while under each mission name there is a table with a list of configurations. The user has the option to mark one of the missions in the table and apply the missions by pressing the appropriate button. The button can only be pressed when a configuration is selected for each mission listed. The user also has the option to click on the name of the mission and

thus display a more detailed description of the mission, which is located in a separate ***Mission component.***
The table showing the configuration consists of four columns: ***Name, Integrity, Confidentiality, Availability.*** There is an option to click on a configuration name to view detailed information about that configuration along with a visualization of how applying that configuration will affect mission availability.

There is a component on the Decide/Act panel that visualizes the mission's dependence on information services and supporting IT components. The visualization is located in a separate component, the input of which is a dependency structure in JSON format.
Graph rendering is secured using the ***ngx-graph library.*** Since it is a standalone component, there is an option to port it to another Angular application as well.

Testing

Testing is handled in the form of unit tests, which are reusable for further development of the system. The application includes unit tests of services and components. Test files are named with a .spec extension, for example, for a file called api.service.ts, the unit tests are in a file called api.service.spec.ts in the same directory.

The tests use the Jasmine framework in version 3.4.0. The Karma library in version 5.0.0 is used for the test triggers themselves.

Continuous integration

Continuous integration (hereinafter referred to as CI) aims to automate the integration of source code changes into a central repository. For this purpose, the ***Gitlab CI/ CD*** tool was chosen for this project .

The ***Gitlab CI/ CD*** tool is configured using a YAML file named .gitlab-***ci.yml*** located in the root directory of the repository. This configuration file contains settings and instructions for the CI process itself. In the CI project, compliance with conventions is verified by running the ***ng lint*** command and unit test success is verified by running the ***ng test*** command in the CI environment every time a source code change is sent to the main branch of the repository.

Thanks

The work on the software was supported by the project ***Research of tools for assessing the cyber situation and supporting decision-making of CSIRT teams in the protection of critical infrastructure*** (VI20172020070) solved in ***the Security Research of the Czech Republic*** program in the years 2017-2020 at Masaryk University. The author of the software is Lukáš Matta, Martin Husák also participated in the design. The authors would like to thank Lukáš Sadlek and Stanislav Špaÿek, who participated in the design of some panels.