Software for applying reactive measures to the elements of active defense of a computer network

Stanislav Špaÿek, Milan Žiaran

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 ***"Software for the application of reactive measures on the elements of active defense of a computer network"*** (hereinafter Act), 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 CRUSOE project) solved in the Security Research program of the Czech Republic in the years 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. ***"Software for decision support in solving a security incident"*** implements the Decide phase, i.e. the decision support phase.

The Act phase software consists of active defense element (PAO) wrappers and a central act-overseer service. The element of active defense is a security tool that is able to stop an attacker by manipulating network traffic or otherwise protect connected devices and services, e.g. firewall, DNS firewall, or RTBH. Wrappers "wrap" the access interface of existing PAOs and externally unify the access functions and return types when calling any element. The act-overseer service then relies on a unified interface provided by wrappers and serves as a central point of contact between PAO and software from other phases of the project.

The act-overseer service has two primary functions – monitoring and executive. As part of the monitoring activity, act-overseer collects operational data from connected PAOs in real time. This data includes information on the current availability and capacity of the elements. Current data is stored in the project's central database for further processing, visualization, and presentation to system users. As part of the executive activity, the act-overseer processes the requests of the Decide phase to block or unblock entities in the network by changing the PAO configuration and, depending on the current state of the given PAO, executes these requests. The software of the Act phase therefore represents a means that converts the outputs of the software of the other phases into the form of specific measures, applies these measures to the active defense elements, and uses them to directly manipulate the traffic in the network.

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 implements a set of tools to support automated and assisted responses to security events using active defense elements in a protected internal network environment.
The software keeps track of available active defense elements and processes input commands from security operators or automated security systems. He checks, distributes and executes these orders on the relevant active defense elements with regard to their function, availability, and free capacity. The software displays the current state of the active defense elements and the results of the operations performed in a separate dashboard application.

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

Contact person: RNDr. Stanislav Špaÿek Institute of
Computer Technology Masaryk
University Šumavská 416/15

Brno 602 00

e-mail: spaceks@ics.muni.cz phone:
+420 549 49 6094

Economic parameters of the result

The market segment is mainly represented by organizations operating critical information infrastructure or other infrastructure with high requirements for confidentiality, availability and integrity.
The result enables users (operators of networks and services) to use central information management and a unified interface for controlling all common elements of active defense that are usually deployed in the network to ensure cyber security. This eliminates the problem of decentralization and inconsistency in access to these elements, which ensures easier control of the security system and faster response to a security incident. Use of the result is licensed free of charge.

Description of the result

Currently, responding to cyber security incidents can be done manually or automatically without the operator's intervention. Manual incident response relies on expert operators to resolve incidents in real time. However, cyber security incidents are still increasing, and the demands on the expertise and abilities of operators are growing accordingly.
Automating cybersecurity incident response is an ever-evolving and unsolved issue.

The CRUSOE project takes the Observe-Orient-Decide-Act (OODA) cycle and adapts it for use in a cybersecurity incident response environment. The project is divided into individual phases according to the steps of the OODA cycle, and each phase implements software providing support services with functions falling within their area. The goal of the software developed under the Act phase of the CRUSOE project is to partially automate the response to security incidents. An expert operator is still necessary, but the software simplifies manual operations, offers automated response processes, and provides feedback on actions taken. This should facilitate, clarify and speed up incident response processes.

When developing incident response automation software, it is necessary to take into account the environment into which the software will enter, i.e. the already existing tools with which it will necessarily have to cooperate. Security tools, or elements of active defense, are being developed to prevent or respond to a cyberattack. Necessarily, therefore, with the development of attacker processes and with the discovery of new possible attack vectors, new and existing PAOs are created. This method of gradual development can lead to different architectures, implementations, and thus also different ways of working with these tools. Thus, in a typical network that CRUSOE software is intended to protect, active defense elements are located in different physical or logical locations, their access interfaces support different sets of functions, and the return types of these functions are in different formats or with different syntax.

The heterogeneity and decentralization described above is undesirable from the point of view of the operator and from the point of view of a partially automated system. The operator is forced to know different control concepts and data interpretation for different PAOs. The system is also forced to adapt to work with a number of different environments. Therefore, it is necessary to solve the problems of centralization and unification of elements of active defense before implementing partial automation of response to an incident. The goal of the software phase of the Act is not to design or create new elements of active defense. On the contrary, the basic idea is to use the already existing security infrastructure and existing security tools as effectively as possible. The software therefore tries to ensure the centralization and unification of PAO so that it can also run in parallel with the existing security infrastructure.

The centralization of active defense elements is achieved by introducing a single central service (act-overseer), which serves as a contact point for all elements. If software from another phase of the project needs to manipulate some PAO, it is done through this central service.
The central service has two primary functions - (i) a monitoring function, i.e. to provide up-to-date

information on the status of all PAOs and (ii) an executive function, i.e. perform PAO configurations based on software requirements from other phases of the project.

The unification of active defense elements is achieved by introducing wrappers that wrap the existing access interface of the element. Each element thus has a clearly defined set of functions it supports and a strictly given format and syntax of inputs and outputs. The Act phase software defines five categories of common active defense elements. Elements have been divided into these categories based on the function and type of network entities they work with. The categories are as follows:

- **DNS firewall** is a security element that can block queries to specific domains. Typically, this is a DNS resolver equipped with a blacklist of known malicious domains. A query to a blacklisted domain is neither forwarded nor answered. An example of a DNS firewall is the DNS Response Policy Zones implemented in BIND 9 DNS server. [1]
- **A firewall** monitors network traffic, typically at the boundary of the internal network, and allows blocking access from the external network to devices on the internal network, based on the device's IP address. An example of a simple compatible firewall is, for example, the basic firewall in the Linux-UFW system. [2]
- **Mail Filter** allows you to block the forwarding of messages from specific source e-mail addresses in the internal network. The function of this PAO corresponds to a mail server with message filtering support (e.g. Sendmail).
- **Remote Triggered Black Hole (RTBH)** is an active defense feature that allows you to stop unwanted network traffic before it reaches the protected network. RTBH is defined in more detail in the CISCO document. [3]
- **User Blocker** is able to block access of specific users based on their identifier to resources located in the internal network. An example of user blocking is, for example, locking a user account in a Microsoft Active Directory environment.

The Act phase software was developed as part of the CRUSOE system, so it must be used in conjunction with software developed in other CRUSOE phases to fully utilize the features. However, the software can also work independently, while maintaining the minimum configuration containing the necessary dependencies, outside of the CRUSOE system. Only wrappers developed for active defense elements can be used. In such a case, the software will serve as an intermediate API layer between the active defense elements and another superior incident response system. It is also possible to use wrappers and the monitoring function of the central act overseer service without the executive function.
The software will then provide the current status of all connected PAOs.

The software designed and developed in the Act phase of the CRUSOE project thus serves primarily two purposes. On the one hand, it allows decisions made in and based on the Decide phase to be accepted

[1] https://www.dnsrpz.info/
[2] https://help.ubuntu.com/community/UFW
[3] https://www.cisco.com/c/dam/en/us/products/collateral/security/ios-network-foundation-protection-nfp/prod_white_paper0900aecd80313fac.pdf

manipulate network traffic, and on the one hand provides information about the consequences of these decisions back to the beginning of the OODA cycle to the Observe phase.

Methods and technologies The
following methods and technologies were used in the design and development of the software.

Wrappers

A wrapper function is defined as a function in a software library or computer program whose main purpose is to call another subroutine or system call with little or no additional computation. Wrapper functions are used to make computer programs easier to write by removing the details of the[4] underlying subroutine implementation.

In the CRUSOE project, the Act software phase plays the role of the program and the active defense element plays the role of the subprogram. The wrapper wraps the PAO implementation, which can be arbitrary, and introduces a uniform interface with a clearly defined set of functions and input/output format and syntax that the PAO must support. At the same time, it is not necessary to intervene in the existing interface of the element, and any existing and already deployed security tools remain functional.
Wrappers thus ensure the unification of PAO access interfaces, on which other security functions can be built.

The disadvantage of wrappers is the need to write the logic of the wrapper functions before deployment. To achieve the widest possible compatibility, wrappers are defined as a framework without any specific implementation of the defined functions. Their content depends closely on the specific existing PAO interface to which the respective wrapper is to be deployed. At best, the wrapper will use an existing interface function and just reformat its input or output. However, if the existing interface does not provide the function, it is necessary to implement it either in the interface itself or in the wrapper.

Despite the aforementioned necessity of implementing logic, wrappers provide the necessary variability that allows the Act phase software to be deployed in diverse environments. The Act phase software is thus not tied to specific elements of active defense or their various implementations.

Modularity

During the development of the software of the Act phase, the effort was to create a modular rather than a unified whole. The modular unit can be better adapted to the variability of the environment in which it is to be placed.
The software is divided into the act-overseer package, which includes the logical part, and the act-component package, which includes individual wrappers. The software as a whole is dependent on some elements developed within the project, but outside the Act phase (see chapter Installation, subsection Preparation). These are basic components of the system such as the Neo4j central database and the Celery organizational service. During the design, emphasis was placed on limiting these dependencies to the necessary minimum

[4] Siler, Brian. ***Special Edition Using Visual Basic*** 6. Que Corp., 1998.

The logic package, unlike the wrapper package, also forms an indivisible unit of the act-overseer service consisting of four interconnected components – monitoring, executive, configuration, and access interface. The relationships between these components are described in more detail in the Architecture subsection. The logic package must be installed in its entirety, but it is possible to use only part of the services after installation, depending on what other software from other phases of the Crusoe project have been deployed. PAO monitoring is a process operated exclusively within the Act software phase and the necessary dependencies. It is thus possible to use it in any case. In contrast, PAO executive management is dependent on the software outputs of the Decide phase, and if this software is not deployed, the service will not be available.

Wrappers are mutually independent and can therefore be used separately. If a PAO does not exist in the environment where the software is deployed, its wrapper does not need to be deployed. On the contrary, every PAO that is to be managed by the Act phase software must have its wrapper deployed. The procedure for possibly removing the wrapper from the software installation is described in the chapter Installation, subsection Preparation.

Possible modular variations of the software deployment are therefore the following:

- **Any combination of wrappers** – will only ensure the unification of the interface of active defense elements. Other services must be provided by other, e.g., existing tools.
- **Any combination of wrappers + act-overseer service** – will ensure the unification of the interface of active defense elements as well as the subsequent centralization of data collection and management of these elements. This option is recommended when deploying all software developed within the Crusoe project, and the documentation refers to this variant.

Django Class-Based Views

The Django framework was used to implement the Act software phase. Django is an open-source web application framework written in the Python programming language. Software access interface endpoints are implemented using Class-Based Views inheriting from the APIView class. A detailed description of the Class-Based Views technique can be found in the publicly available documentation.[5]

Ansible

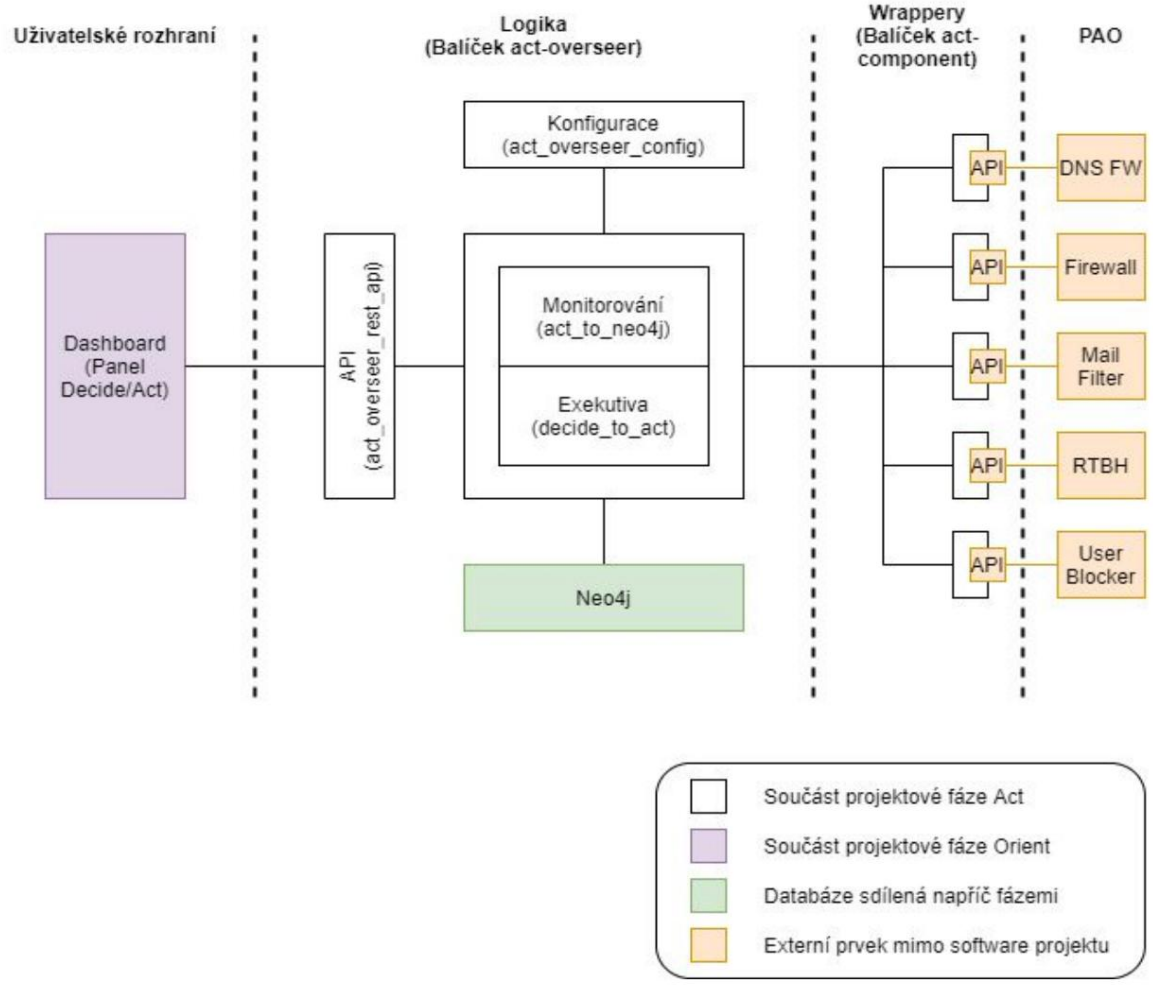
The Ansible tool was used to automate the deployment of the software developed within the Crusoe project. The Crusoe Ansible repository contains code that automatically installs the software developed in the project after initial preparation. Ansible's code and documentation are available as open-source. Software installation is also possible without using Ansible, but then it is necessary to perform all the steps described in Ansible roles manually.[6] This documentation recommends and further describes software deployment using Ansible.

[5] https://www.django-rest-framework.org/api-guide/views/
[6] https://www.ansible.com/

Architecture

The chosen software architecture is based on a modular approach and is shown in Figure 1. At the top level, the Act phase software consists of two packages - the act-overseer service and the act-component wrapper package. The act-component package also consists only of individual wrappers for different PAOs. In contrast, the act-overseer service can be divided into smaller components. The act-overseer package includes the implementation of the basic functions of the service, monitoring and executive, a file with the current configuration of the service, and an access interface.

To illustrate the integration of the software among other software developed in the project, elements falling outside the Act phase are also included in the figure. The affiliation of these elements to different phases and outside the project is described in the legend. The first connection is the access interface of the act-overseer service. The access interface of the service can be controlled from the Decide/Act page in the project dashboard. This connection ensures the transfer of information from previous phases of the OODA cycle to the Act phase. The second communication channel is connected directly to the logic of the service. The monitoring and executive functions have direct access to the central Neo4j project database. By updating the data in the database, feedback is ensured on the operations performed with PAO and thus the reversal of the OODA cycle. Wrappers are connected to PAO access interfaces located outside the CRUSOE project. This connection needs to be implemented based on the environment in which the software is deployed.



Giant. 1: Schematic of the interconnection of software modules of the Act phase and connections between other software developed within the CRUSOE project. The name in parentheses indicates the name of the module as it is stored in the project repository.

Act-overseer package

The organization service is the basic logical component of the Act software phase. It combines modules providing monitoring and executive functions on the PAO as well as configuration data necessary for the running of these modules. This component is the only central element in the otherwise decentralized PAO architecture. It consists of the following components:

- Monitoring
- Executive
- Configuration file • Access interface

Monitoring

The monitoring function provides up-to-date data on the PAO status in the Neo4j database. When the Act phase software is installed, this function is triggered and immediately updates the data in the database. The next start of monitoring is carried out using the Celery project organization service and possibly as needed. Monitoring is triggered periodically at intervals adjustable in the Celery configuration. Monitoring can also be invoked by invoking an executive function and making changes to the PAO configuration.

Monitoring calls the wrapper function to find the maximum and current capacity, and the function to verify the liveness of the PAO. The obtained data is immediately written into the database, where the software of the other phases of the project is accessible. The data is not transformed or modified in any way before being written to the database. Both successful and unsuccessful wrapper function calls are logged by the monitoring function and stored in a defined location. The monitoring algorithm is described in more detail in the programming documentation.

Executive

The executive function, in cooperation with the Decide phase software, provides partial automation of the PAO configuration. The function is started by the operator manually by selecting the recommended configurations using the GUI in the dashboard and command to execute the settings. After the end of the auto-configuration, the function informs the operator about the setting progress and possible errors.

Features from the GUI dashboard receive a list of recommended configurations to be applied to the network. The details of the given configurations are found by the function directly from the Neo4j database. Then, using custom logic processes (described in detail in the user and programmer documentation), it transforms the configuration into unblock and block lists. It then applies these lists to the given PAO by calling the relevant wrapper functions. The results of blocking and unblocking operations are logged by the service. Important events and the overall result of the setup can be monitored in real-time on the GUI dashboard. The executive algorithm is described in more detail in the programming documentation.

Configuration file The

act-overseer configuration file is located in /act-overseer/data/act_overseer_config.
It contains 5 adjustable parameters that are necessary for the service to run. The file contains the following parameters (the values are for example):

```
{ "security_treshold": 50,
  "log_path": "/var/log/crusoe/",
  "user": "user",
  "password": "pass",
  "server_url": "http://172.18.1.10: 8088" }
```

The ***security_threshold*** parameter contains the current parameter value valid for the execution function of the act-overseer service. A detailed description of the use and meaning of this value can be found in the user and programmer documentation. The ***log_path*** parameter contains the path to the directory in which the act-overseer service should store files recording events when the functions run.
The user and ***password*** parameters contain login data to the project REST API combining access to the Neo4j database with other functions. The ***server_url*** parameter contains the url or ip address and port, including the protocol, where the project's REST API can be found.

Access interface

The access REST API provides functions to control the behavior of the act-overseer service. Allows you to read and change the value of the ***security_threshold parameter.*** Furthermore, it allows reinitialization of active defense elements in the database if an existing element is added, removed, or changed. The last key function of the interface is the launch of the executive function of the act-overseer service and the auto-configuration of the PAO according to the configurations recommended by the Decide phase software and selected by the operator. The behavior of the monitoring function is not controlled through this interface, but through the configuration of the Celery organization service. A detailed specification of the interface functions is given in Appendix B.

The act-component package

The act-component package contains wrappers developed for defined PAO categories.
Wrappers define the set of functions that the PAO must support, as well as the format and syntax of the inputs and outputs of those functions. Wrappers are divided into folders, each folder contains all necessary wrapper dependencies to be able to deploy wrappers separately on different devices.
Detailed wrapper specifications are part of this documentation and can be found in Appendix A.
This specification is also supplied in electronic form in this package in the /act-component/specification folder.

For the needs of software testing in the CRUSOE project, a special simulated active defense element was developed. Testing software in a real environment and on real PAOs could otherwise cause connection failures or unavailability of services. This simulated active defense element replaces the real firewall and is located in the /act-component/simulated-pao-firewall folder.

The element behaves outwardly in the same way as a real firewall would, but does not apply the implemented measures to the network. The element can be used to test all software developed within the CRUSOE project in both virtual and real environments. Its installation instead of a real firewall wrapper is described in the Installation chapter.

Installation instructions

The instructions for installing the Act phase software are based on the use of Ansible. The software includes everything needed for automatic deployment of both the Act phase software. Before starting the installation, it is necessary to change the contents of some Ansible configuration files depending on the environment where the software is deployed. These changes are described in the subsection Preparation.

Preparation

• Unpacking the software archive to the /crusoe/act/ location. • Filling

in the password to the act-overseer service interface in ansible/ansible/group_vars/all/vault.

File The vault file then needs to be encrypted with a new password and filled into the ansible/vault_pass file. • Editing the file /anisble/act.yml

• The included configuration serves as an

example covering all available

components.

• It is necessary to fill in the IP addresses and ports of the machines on which the software components will be installed

deployed.

• It is recommended to deploy the act-overseer service on a server with other software from other

phases, the access interface will run on the specified port. • Wrappers ÿ

Warning: the

wrapper does not contain logic, it must be added before deployment - it differs depending

on the environment where the wrapper is deployed. ÿ It is possible to

deploy them to any server on the network, but the server running the PAO to which the

wrapper applies is recommended. ÿ If any of the supported PAOs

do not exist in the network, they should be deleted from act.yml. ÿ A simulated PAO firewall

can be used in a test

deployment.

For a wrapper with ***the wrapper: firewall*** parameter, ***the dst_wrapper*** parameter

must then be changed from ***the firewall-wrapper*** value to

simulated-pao-firewall.

ÿ For other PAOs and act supervisors, the specified parameters are mandatory. ÿ ip

… the ip address of the server where the wrapper is to be installed ÿ

portnumber … the port of the server where the wrapper is to run ÿ

server_name … the FQDN of the server where the wrapper is to be installed ÿ

pao_name …. PAO name {dnsfw, firewall, userBlock, MailFilter, rtbh} ÿ maxCapacity: "0" ÿ

usedCapacity: "0" ÿ

freeCapacity: "0" ... the

initial capacities do not need to be modified, after installation they are automatically

added by the act-overseer service

Installation

The Crusoe Ansible repository defines a virtual environment into which the software can be deployed. The installation can be started with **the vagrant up** command in the root directory of the repository. The installation will then take place automatically. When deploying to a non-prepared test environment, you need to run Ansible directly, with the playbook.yml parameter.

The installation was tested for Ansible version 2.9.10. If the software is deployed to a virtual environment defined in the repository, it also requires Vagrant 2.2.9 and VirtualBox 6.0.24 r139119 on the host device. Installation will likely be possible on newer versions of these tools as well.

After installation, check whether the url or IP:port REST API is filled in the server_url parameter in the configuration file of the act-overseer service (/usr/local/lib/python3.7/dist-packages/act_overseer/data/act_overseer_config). It is necessary to fill in this parameter including the http protocol (e.g. http://172.18.1.10:8088). But this address must be among the allowed addresses in Apache sites-enabled on the server with the Neo4j central database interface. If act-overseer is running on the same server as the access interface (recommended and default setting), ***localhost*** can be used instead of the address, which does not need to be explicitly enabled in Apache.

Checking the installation results

After installation, it is recommended to check the running of all deployed services. To check the operation, you can use the procedures described further in the User documentation - Troubleshooting section.

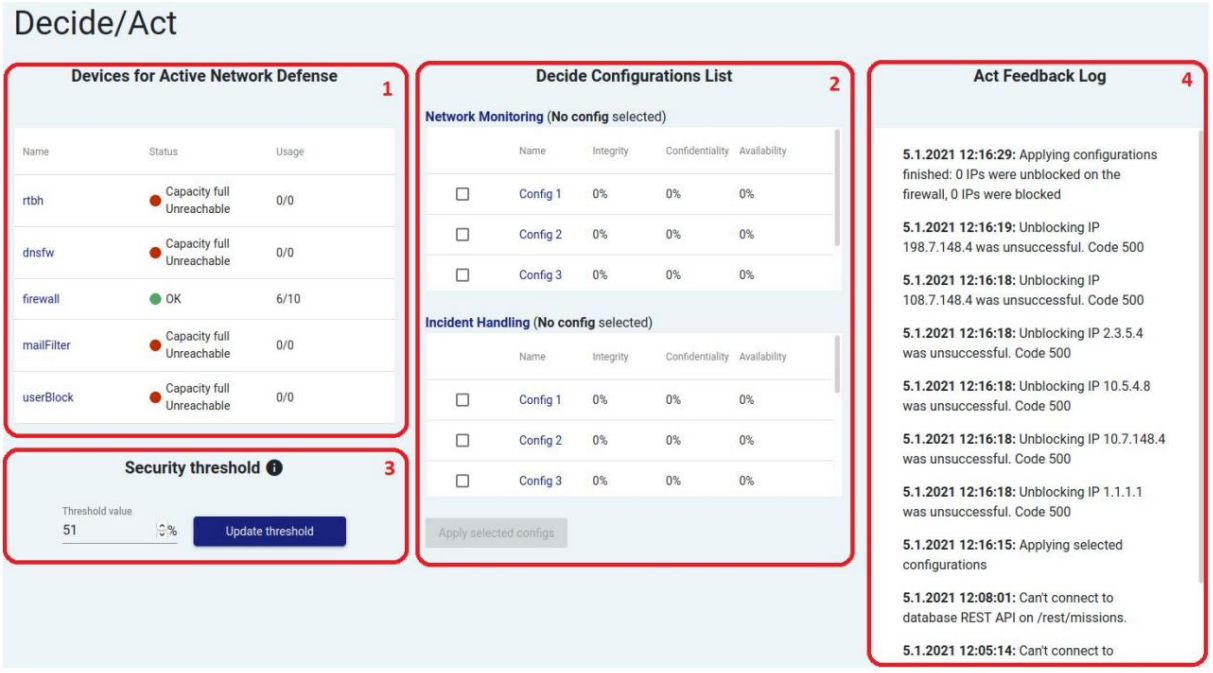
User documentation

The Act phase software is primarily intended for two use cases:

1. Centralization of information from available elements of active defense 2.
Implementation of automatic settings of elements of active defense according to the configurations recommended by the software of the Decide phase

This section describes all parts of the software that the operator will work with in normal use.

Decide/Act dashboard panel All informative 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 2.



Giant. 2: The Act phase software page in the dashboard.

The image 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 3. 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.

Giant. 3: List of active defense elements. A single element (firewall) is connected, the others have not been deployed.

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.

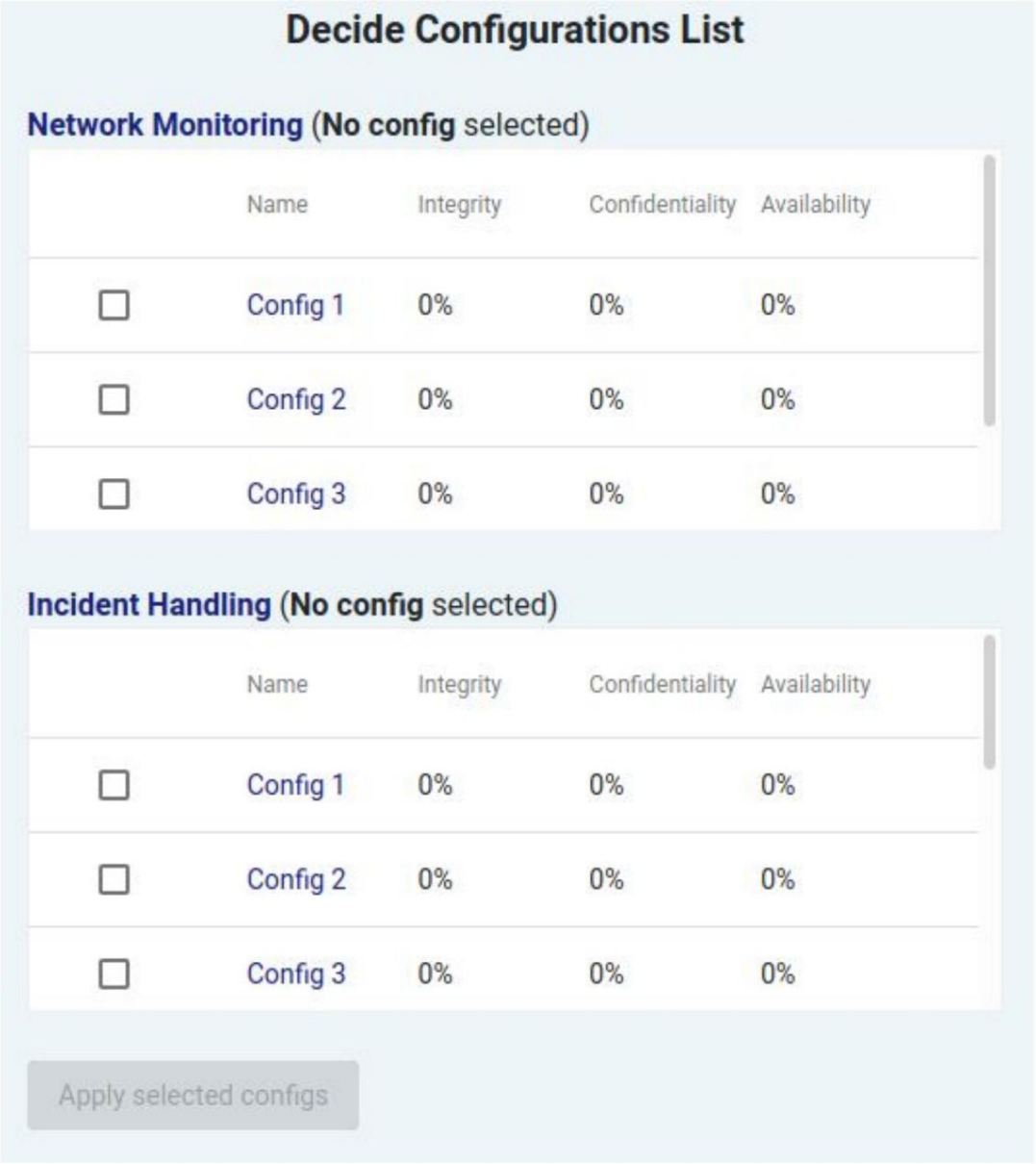
  - Capacity 90% full - 90% of maximum capacity detected. • Last liveness check failed -
    The last contact was successful more than 10 years ago
    minutes.
- Red - the element has a problem that must be solved, otherwise it is unable to continue operating.
  - Capacity full - The used capacity of the element is equal to the maximum capacity.

- Unreachable - Last contact was successful more than 30 minutes ago.

Note: Liveness check intervals can be set in the Celery central organization service.

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. From the point of view of the Act phase, each configuration requires blocking or, conversely, unblocking specific machines in the network in order to fulfill the mission. The mission and configuration list panel is shown in Figure 4.



Giant. 4: List of missions and configurations. 2 missions are defined - 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. Click on the name of the mission

able to view a map of the facilities that this mission requires. 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 checkbox for each configuration allows you to select this configuration and use the Apply selected configs button to block and unblock active defense elements.

Note: Before applying configurations, you must select **exactly one** configuration for **each** mission .

The current value of the Security threshold

Each machine located in the network protected by the Crusoe system receives an evaluation in three monitored parameters - the degree of threat to integrity, confidentiality, and availability. More about these parameters can be found in the software documentation of the Decide phase. The Act phase software works with the aggregate machine threat level, which is calculated as the arithmetic average of the threat level of integrity, confidentiality, and availability of the given machine. So it reaches values from the interval <0, 100>.

Giant. 5: Security threshold settings panel.

The meaning of setting the Security threshold parameter is as follows. Applying a configuration without a Security threshold setting would block all devices that are not mission-critical to which the configuration applies. Devices would therefore be blocked regardless of their level of threat. Such a procedure is unnecessarily restrictive, as in a standard environment it is not necessary to block devices with a relatively low level of threat.

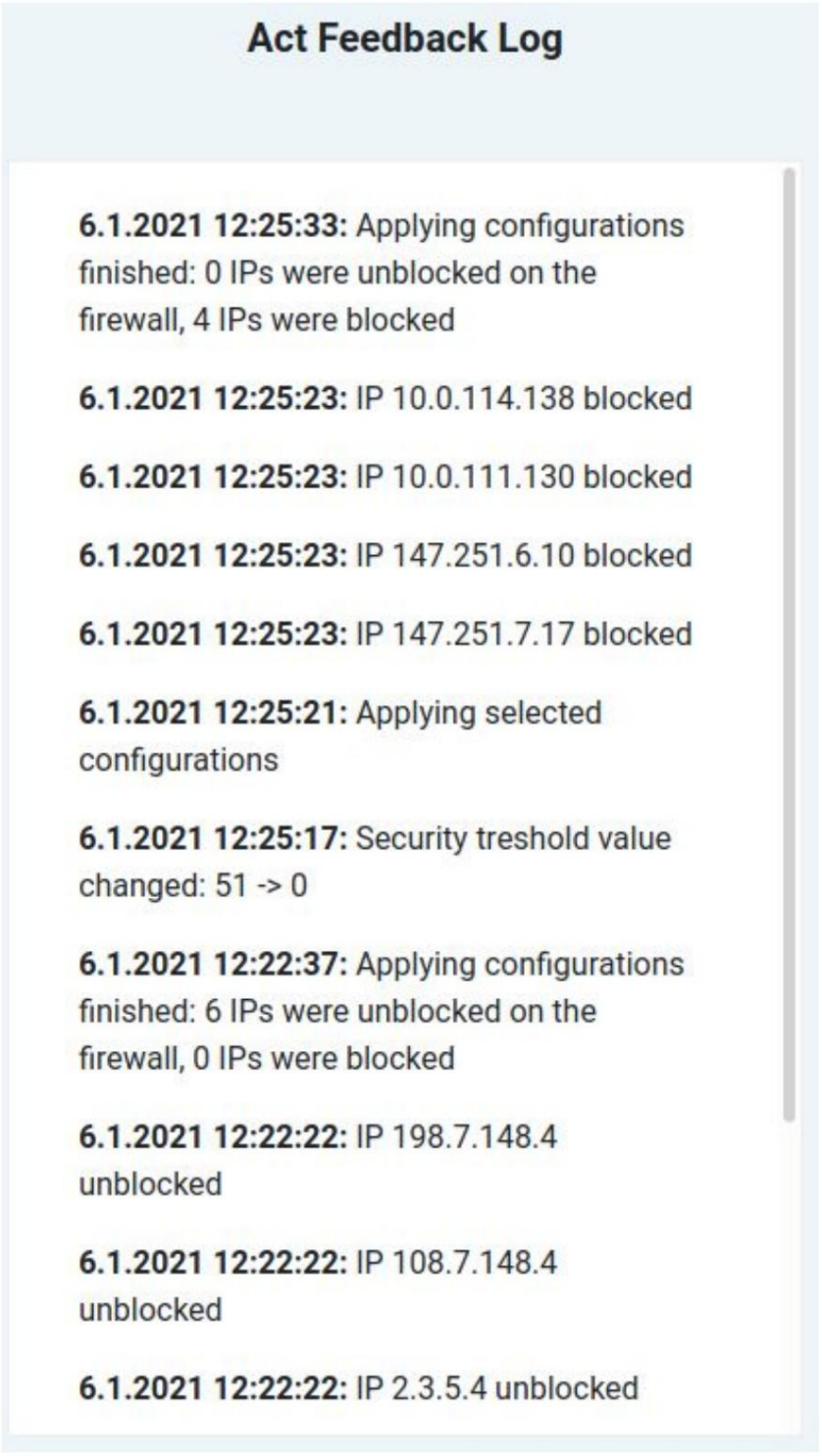
Setting the Security threshold value allows you to exclude devices with a level of threat **lower** than the set value from blocking. Figure 5 shows a Security threshold with a value of 51. If the configurations were applied to a network with this setting, all devices with a threat level higher than 51 would be blocked. Conversely, devices with a lower threat level would not be blocked, even though some configurations recommend it.

When the Security threshold value is set to 0, the system behaves as if this parameter was not implemented when applying configurations to the network, and everything according to the configurations is blocked. At the opposite extreme, Security threshold = 100, no devices 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,

and possibly errors or warnings that caused these actions. An example of the content of the panel when trying to apply configurations is shown in Figure 6. The messages displayed in the panel are filtered to maintain clarity and do not capture the complete form of the Act software phase logo. When troubleshooting problems with the software beyond normal use, it is recommended to check the software log files (see Troubleshooting subsection).



Giant. 6: List of current events in the Act stage. 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.

Monitoring of active defense elements

The actions described in this chapter describe the functions available for use case 1, i.e. centralization of information from available active defense elements. The current status of all elements of active defense can be monitored on the List of elements of active defense panel. The status is updated at regular intervals adjustable in the configuration file of the Celery central organization service.

The description of the data displayed in the panel can be found in the previous chapter.

Changes to active defense elements

If it is necessary to make changes to active defense elements after the Act phase software has been deployed, it is necessary to reinitialize the elements. Changes requiring reinitialization include:

- Adding or removing an active defense element.
- Moving an active defense element to another IP address and/or port.

Re-initialization of elements is carried out via the own function /act/initialize of the act-overseer service interface. This function needs to be sent (e.g. by the curl command) a JSON file in the format according to the example in the act-overseer repository.

The file path is /act-overseer/specification/ood-apis/pao-init.json. This file must be corrected to match the current location of active defense elements.

Method:

- If the element is not in the network, it is possible to delete it from the initialization file.
- For the remaining elements, modify the IP and port parameters so that they point to the wrapper of the relevant element.
- Leave the other parameters (capacity, contact) in the basic settings, they will be filled in automatically by the monitoring service after reinitialization.
- Use the curl command to pass the JSON file to the /act/initialize endpoint of the act-overseer interface

Applying configurations to the network

The actions described in this chapter describe the functions available for use case 2, i.e. performing the automatic setting of active defense elements according to the configurations recommended by the software of the Decide phase.

The List of Missions and Configurations panel displays all defined missions and their available configurations. By clicking on the name of the mission, it is possible to view its details - verbal description and assessment of the importance of the mission. The list of missions is updated automatically if a new mission is added to the database.

For each mission, Decide provides possible configurations. Configuration can be simply thought of as a list of devices that must be available to accomplish a mission. From the point of view of the Act software phase, the configuration therefore determines which devices can be blocked,

so that the mission can be fulfilled and at the same time the devices in the network are not available, if it is not necessary for the fulfillment of some mission. By clicking on the name of the configuration, it is possible to view detailed information, in particular a list of available devices and a network map. The list of available devices explicitly names the devices that will ensure the mission runs in the given configuration. The network map shows all the devices that the configuration works with. Highlighted in red are those devices that the configuration recommends blocking. A detailed description of the issue of missions and configurations can be found in the software documentation of the Decide phase.

Choosing the most suitable configuration for a given mission depends on several criteria. The basic criteria are requirements for integrity, availability, and confidentiality. Each configuration is rated by the threat level of the given attribute expressed as a percentage. Thus, **a lower** value for a given attribute means **a better** result for a given configuration compared to others with a higher level of threat. Another criterion is the requirement for the operation of a specific device, which, however, is not critical for the performance of any mission, or can be replaced by another device for the performance of the mission. Checking the devices available for that configuration will show whether applying the configuration to the network will block such a device.

Once exactly one configuration is selected in the panel for each mission, it is possible to apply it to the active defense elements by pressing the Apply selected configs button. This will start the act-overseer service process. The progress of this process can be followed in the List of current events panel in the Act phase. In this panel, you will first see a message about the start, then about the success or failure of each blocking and unblocking, and finally a final message about the end of the process. An example is shown in Figure 5.

Note: When new configurations are applied, previous configurations from active defense elements are deleted to prevent incremental configuration overlap.

Procedure:

- For each mission, it is necessary to select exactly one configuration, e.g. based on the lowest degree of threat to integrity, confidentiality, or availability.
- The configuration application process is started by pressing the Apply selected configs button. • The current events list panel shows the progress of the process.

Troubleshooting This chapter contains procedures for checking the operation of individual components of the Act software phase. Also documented are some problems that could occur during the operation of the software and their possible solutions.

- Are all installed wrappers available on the specified ports?
  - Each wrapper can be contacted by querying the REST interface in the format <ip>:<port>/<pao_name>/capacity. If the wrapper returns an answer to this query, it can be considered functional. You can call other endpoints of the wrapper in a similar way and monitor the responses. • The wrapper service may have crashed and needs to be restarted. • Is a central act-overseer service available?

• The central act-overseer service can be contacted by querying REST interface in the format localhost:<port>/act/threshold. If the service returns an answer to this query, it can be considered functional. • The act-overseer service may have stopped and needs to be restarted. • Is automatic monitoring of active defense elements running? • Automatic PAO monitoring at set intervals (5 minutes in the initial setting) detects the current liveness and capacity of all connected elements. If the service is running, the PAO capacities in the database and therefore on the Act page in the dashboard should change to current immediately after the installation is complete. • If there is an error when writing to the database, these are errors captured in /var/log/crusoe/act_overseer.log on the server running act-overseer.

• Are there any other errors while the services are running? • All services log their run, errors and operations performed stored in special log files

• /var/log/crusoe/act_overseer.log ÿ PAO monitoring service running • /var/log/crusoe/act_overseer_rest_api.log ÿ commands entered through the act_overseer service interface • /var/log/crusoe/act_decide_to_act.log ÿ PAO automatic configuration service running based on configurations recommended by the Decide phase software • Application of configurations the mission will fail with the error “Can't connect to database REST API on /rest/missions”.

• The act-overseer service failed to connect to the Neo4j database REST interface. First, check that the interface is running on the given IP:port or url.

• If the interface is running, check if it is in the configuration file act-overseer (/usr/local/lib/python3.7/dist-packages/act_overseer/data/ act_overseer_config) url or IP:port REST API filled in server_url parameter. It is necessary to fill in this parameter including the http protocol (e.g. http://10.0.0.1:8088). • The simulated firewall cannot block or unblock any IP addresses (error 500). • The firewall file does not have sufficient permissions set to execute changes.

• Set the file /var/www/simulated-pao-firewall/firewall_wrapper_project / simulated_pao_firewall access rights 666. The file is located on the server where the firewall wrapper was deployed.

Programming documentation

The Act phase software is primarily intended for two use cases:

1. Centralization of information from available elements of active defense 2.
Implementation of automatic settings of elements of active defense according to the configurations recommended by the software of the Decide phase

Two algorithms were designed to fulfill these use cases, which were subsequently implemented using the Python language and the Django framework. This section is devoted to the description of these algorithms as well as the functions by which the algorithms were implemented. Furthermore, all other implemented auxiliary elements are described, such as access interfaces and
wrappers.

Algorithm of information centralization from PAO Information
centralization takes place according to the following algorithm: 1. Act overseer
checks the liveliness of all PAOs periodically at a fixed interval t.
2. It updates the results immediately in Neo4J.
3. The Decide/Act panel, as part of the dashboard, retrieves data o from Neo4J at an interval ***t***
availability and capacity of PAO.

PAO autoconfiguration algorithm

1. The operator selects exactly one configuration for each mission in the dashboard and confirms the execution. The dashboard sends to endpoint /act/protect_missions_assets act_overseer_api a list of selected mission-configuration pairs.
2. Overseer gets the list of all missions from Neo4j (endpoint DECIDE/missions) and verifies that the json contains all missions, if not, throws the appropriate error and exits.
3. The Overseer will get a list of all machines and a list of critical pro machines from the database mission-configuration pairs.
    a. endpoint DECIDE/missions/hosts returns the IP of machines from **all** missions and for each the worst (highest) availability, confidentiality and integrity threat value that it has in any of the missions.

    b. endpoint DECIDE/mission/<name>/configuration/<config_id>/hosts returns IP
    mission-critical machines in a given configuration
    c. i.e. set difference of API function call results: {DECIDE/missions/hosts} - {DECIDE/mission/<first-mission>/configuration/<config-id1>/hosts} - {DECIDE/mission/<last-mission> / ... - configuration/<config_id2>/hosts} = list of POTENTIAL blocks

    d. {list of POTENTIAL blockages} - {machines with safety by rating < security
    threshold} = list of NEW blocks (SNB) i. endpoint DECIDE/
    missions/hosts returns a list of all hosts in missions
    + their worst parameters of confidentiality, availability and integrity (treat the case when the machine is not in the list - log in and do not block)

ii. security rating of the machine = arithmetic average of the confidentiality, availability and integrity parameters of the machine 4. Solving the problem when the original configuration overlaps with the new configuration
a. from the wrapper function, the overseer finds out the list of currently blocked IP addresses on the Firewall = blocking list (SB) b. IP is included in SNB && SB -> no action c. IP is in SNB && !SB -> block IP d. IP is in !SNB && SB -> unblock IP e. sorts the list of rules - unblock first (we save capacity) f. the created list of rules can already be applied to the PAO (firewall)
5. Overseer finds the current freeCapacity parameter and PAO availability using functions wrapper.
a. Checks free capacity on PAOs for which it has a list of actions (firewall). Free capacity is calculated as freeCapacity + the number of slots freed by unblocking

b. Returns an error on out-of-capacity operations. It will not perform these operations even partially (if there is no free capacity, the process will end with an error)
6. Sequentially performs actions from the rule lists on the respective PAOs.
7. Finds the current values of the PAO parameters using the wrapper. Updates PAO records in Neo4J (capacity, availability), wherever there has been a change (unavailability, capacity change).

8. Finally, the overseer displays a summary of the operations in the dashboard (number of success/ failures, blocking/unblocking...).

Logging

The software logs the following operations:

• success/failure for automatically triggered actions (aliveness check, configurations, ..) • success/ failure for manually triggered operations (coming from the dashboard, ..) • every operation on an active defense element • error states and communication failures

Events are stored in the following files: • /var/log/crusoe/ act_overseer.log • PAO monitoring service running • / var/log/crusoe/act_overseer_rest_api.log • commands entered through the act_overseer service interface • /var/log/ crusoe/act_decide_to_act.log • PAO automatic configuration service running based on configurations of the recommended software of the Decide phase

Log format:
Each message consists of four parameters separated by a space. event
• dates capture date event capture
• time time event severity level
• severity {info, warning, error}

- messages a message describing the event

Example event:

2020-12-08 13:10:43,845 INFO Security threshold value changed: 50 -> 60

The act-overseer package Act-overseer consists of 4 modules - act_overseer_config, act_overseer_rest_api, act_to_neo4j, and decide_to_act. Act-overseer functions are therefore further divided according to the module to which they belong.

The act_overseer_config module

The configuration file act_overseer_config contains:
"security_threshold" - Security rating value that can be read and changed by queries to act_overseer_rest_api. **"log_path"** - Path to the directory where the component logs will be stored. **"user"** - Username to access the database REST API. **"password"** - Password to access the database REST API. **"server_url"** - URL of the database server.

The act_overseer_rest_api module

The act_overseer_rest_api access interface allows you to control and configure the functionality of the act-overseer package. The REST API is built using the Django framework. It has three endpoints:

/protect_missions_assets - This function is used to initiate PAO configuration based on the configurations recommended by the Decide phase software. It expects POST data in the following form: [

```
  {
    "name": "name of mission
    #1", "config_id": 2
  },
  {
    "name":"name of mission
    #2", "config_id": 1
  }
]
```

If the data is evaluated as valid, the main function of the decide_to_act module, described below, is called and the PAO configuration takes place.

/threshold - Accepts two requests, GET and PUT. The current value of the security rating can be read using GET, and written using PUT. The data format for PUT is as follows: {

```
  "security_threshold": 50
```

}

/log - Using the log endpoint, it is possible to get the last 100 messages in the act-overseer log displayed on the dashboard.

The act_to_neo4j module

The act_to_neo4j module ensures communication between the act-overseer package and the Neo4j database. It maintains the current data of active defense elements in the database - IP address, port, capacity, and liveness. It is started periodically, or by a call from the decide_to_act module for the need to update new data to the database. The start-up period can be set in the configuration of the Celery organization service. The module has the following functions:

filename(name)

Returns the absolute path to the resource passed in the 'name' argument. The act-overseer software uses this function to locate the certificate file in the 'data' directory in the act-overseer package.

Parameters:

- **name:** Name of the resource.

get_ip_and_port(pao, wrappers)

Returns the IP and port of the given PAO wrapper.

Parameters:

- **pao:** Name of the active defense element.
- **wrappers:** List of all active defense element wrappers.

get_paos(user, passwd, server_url, logger)

Returns JSON with all contact details (IP, port) to available active defense elements. Returns None when the database REST API call fails.

Parameters:

- **user:** Username for accessing the database REST API. - **passwd:** Password to access the database REST API. - **server_url:** URL of the server on which act-overseer is running. - **logger:** File for logging.

update_last_contact(pao, user, passwd, server_url, logger)

The function updates the time of the last contact with the active defense element wrapper. Returns an object of type Response. Returns None when the database REST API call fails.

Parameters:

- **pao:** Name of active defense element. - **user:** Username to access the database REST API. - **passwd:** Password to access the database REST API. - **server_url:** URL of the server on which act-overseer is running. - **logger:** File for logging.

check_and_update(pao, user, passwd, server_url, logger)

The function detects the liveness of the active defense element and subsequently updates it in the database using the update_last_contact() function. Returns whether the liveness in the database was successfully updated.

Parameters: -

pao: Name of active defense element. - **user:** Username to access the database REST API. - **passwd:** Password to access the database REST API. - **server_url:** URL of the server on which act-overseer is running. - **logger:** File for logging.

update_capacity(pao, user, passwd, capacity_type, capacity_value, server_url, logger)

A function in the database updates the given type of active defense element capacity. Returns an object of type Response. Returns None when the database REST API call fails.

Parameters:

- **pao:** Name of active defense element. - **user:** Username to access the database REST API. - **passwd:** Password to access the database REST API. - **capacity_type:** Capacity type (one of 'maxCapacity', 'usedCapacity' and 'freeCapacity'). - **capacity_value:** The value of the given capacity. - **server_url:** URL of the server on which act-overseer is running. - **logger:** File for logging.

retrieve_and_update_capacity(pao, user, passwd, server_url, logger)

The function detects the capacities of the given active defense element and updates them in the database using the function update_capacity() Returns the number of successfully updated capacities.

Parameters:

- **pao:** Name of active defense element. - **user:** Username to access the database REST API. - **passwd:** Password to access the database REST API. - **server_url:** URL of the server on which act-overseer is running. - **logger:** File for logging.

update_db(user, passwd, server_url, logger=structlog.get_logger())

The main functions of the act_to_neo4j module. The function updates the liveness and capacities for all elements of the active defense using the functions described above. Returns a string with information about the number of successfully updated health and capacity of active defense elements.

Parameters: -

user: Username for accessing the database REST API. - **passwd:** Password to access the database REST API. - **server_url:** URL of the server on which act-overseer is running. - **logger:** File for logging. It is different when called periodically and when called from a module

decide_to_act.

The decide_to_act module

The decide_to_act module contains the logic responsible for setting the PAO according to the software configurations of the Decide phase. The module contains functions for blocking and unblocking IP addresses on the active defense element, as well as functions that decide which IP addresses need to be blocked and which, on the contrary, should be unblocked. These are described below:

get_missions(user, passw, logger, dashboard_log, server_url)

The function gets a list of all missions using the database REST API. He will also return it afterwards. The REST API throws an exception if the call fails.

Parameters:

- **user:** Username for accessing the database REST API. - **passw:** Password to access the database REST API. - **logger:** File for logging. - **dashboard_log:** Log displayed on the dashboard. - **server_url:** URL of the server on which act-overseer is running.

get_configurations(user, passw, mission, logger, dashboard_log, server_url)

The function retrieves a list of configurations for a given mission. It then returns this list. If the function fails to contact the database REST API, it throws an exception.

Parameters:

- **user:** Username for accessing the database REST API. - **passw:** Password to access the database REST API. - **mission:** The name of the mission to which the returned configuration belongs. - **logger:** File for logging. - **dashboard_log:** Log displayed on the dashboard. - **server_url:** URL of the server on which act-overseer is running.

get_fw_ip_and_port(user, passw, logger, server_url)

The function returns the IP and port of the firewall wrapper.

Parameters:

- **user:** Username for accessing the database REST API. - **passwd:** Password to access the database REST API. - **logger:** File for logging. - **server_url:** URL of the server on which act-overseer is running.

get_hosts(user, passw, logger, dashboard_log, server_url)

The function retrieves a list of all devices in the database (identified by IP address) using the database REST API. It then returns this list. The REST API throws an exception if the call fails.

Parameters:

- **user:** Username for accessing the database REST API. - **passw:** Password to access the database REST API. - **logger:** File for logging. - **dashboard_log:** Log displayed on the dashboard.

- **server_url:** URL of the server on which act-overseer is running.

get_important_mission_hosts(user, passw, mission, configuration, logger, dashboard_log, server_url)

The function returns a list of all devices that are critical to the running of the given mission in the given configuration.

Parameters:

- **user:** Username for accessing the database REST API. - **passw:** Password to access the database REST API. - **mission:** The name of the mission to which the returned configuration belongs. - **configuration:** configuration ID. - **logger:** File for logging. - **dashboard_log:** Log displayed on the dashboard. - **server_url:** URL of the server on which act-overseer is running.

potential_blocking(user, passw, missions_and_configurations, logger, dashboard_log, server_url)

The function compares the list of all devices with the lists of critical guest critical devices for the given mission-configuration pairs. The result is a list of potential blockages, ie guests that are not important for any of the missions in the given configuration.

Parameters: -

user: Username for accessing the database REST API. - **passw:** Password to access the database REST API. - **missions_and_configurations:** List of pairs (missions, configurations). - **logger:** File for logging. - **dashboard_log:** Log displayed on the dashboard. - **server_url:** URL of the server on which act-overseer is running.

get_firewall_health(logger, dashboard_log, firewall_ip_and_port)

The function from the firewall wrapper detects its liveness and returns True if the call was successful, False otherwise.

Parameters:

- **logger:** File for logging. - **dashboard_log:** Log displayed on the dashboard. - **firewall_ip_and_port:** IP address and port of the firewall wrapper.

get_firewall_capabilities(logger, dashboard_log, firewall_ip_and_port)

The function from the firewall wrapper detects its capacities and returns these capacities.

Parameters: -

logger: File for logging. - **dashboard_log:** Log displayed on the dashboard. - **firewall_ip_and_port:** IP address and port of the firewall wrapper.

get_threshold(logger, dashboard_log)

The function returns the security threshold value.

Parameters:

- **logger:** File for logging. -
dashboard_log: Log displayed on the dashboard.

remove_less_than_threshold(potential_blockings, logger, dashboard_log)

The function returns a new blocking list, obtained by removing guests from the list of potential blockings 'potential_blockings' that have a security rating less than the threshold security rating.

Parameters:

- **potential_blockings:** List of potential blockings. - **logger:**
File for logging. - **dashboard_log:**
Log displayed on the dashboard.

get_average_security_value(host, logger)

The function returns the average security value from three values in the guest: availability, confidentiality, integrity.

Parameters:

- **guest:** Dictionary representing the
guest. - **logger:** File for logging.

get_blocked_ips(logger, dashboard_log, firewall_ip_and_port)

The function gets the list of blocked IP addresses from the firewall wrapper. It then returns this list. In case of a failed call, the firewall throws an exception to the wrapper.

Parameters:

- **logger:** File for logging. -
dashboard_log: Log displayed on the dashboard. -
firewall_ip_and_port: IP address and port of the firewall wrapper.

is_already_blocked(ip, firewall_ip_and_port)

The function checks if the IP is already blocked on the firewall.

Parameters:

- **ip:** IP we want to find out if it is blocked on the firewall. -
firewall_ip_and_port: IP address and port of the firewall wrapper.

block_ip(ip, logger, dashboard_log, firewall_ip_and_port)

The function will block the IP address on the firewall.

Parameters:

- **ip:** IP we want to block.
- **logger:** File for logging. -
dashboard_log: Log displayed on the dashboard. -
firewall_ip_and_port: IP address and port of the firewall wrapper.

unblock_ip(ip, logger, dashboard_log, firewall_ip_and_port)

The function unblocks the IP address on the firewall.

Parameters:

- **ip:** IP we want to block.

- **logger:** File for logging. -
dashboard_log: Log displayed on the dashboard. -
firewall_ip_and_port: IP address and port of the firewall wrapper.

unblock_list(blocked_ips_list, to_block_list)

The function returns a list of IP addresses that will be unblocked on the firewall.

Parameters:

- **blocked_ips_list:** List of blocked addresses. -
to_block_list: List of new blocks.

block_list(to_block_list, blocked_ips_list)

The function returns a list of IP addresses that should be blocked on the firewall.

Parameters:

- **to_block_list:** List of new blocks. -
blocked_ips_list: List of blocked addresses.

can_unblocks_and_blocks_be_performed(blocked_ips, list_of_new_blockings, logger, dashboard_log, firewall_ip_and_port)

The function detects whether unblocking and blocking can be performed on the firewall, i.e. if sufficient capacity is available and the element is contactable. Returns True if yes, False otherwise.

Parameters:

- **blocked_ips:** List of blocked addresses. -
list_of_new_blockings: List of new blockings. - **logger:** File for logging. - **dashboard_log:** Log displayed on the dashboard. - **firewall_ip_and_port:** IP address and port of the firewall wrapper.

run_decide_to_act(user, passw, missions_and_configurations, logger, dashboard_log, server_url)

The main function of the decide_to_act module, which is invoked from the act-overseer REST API after applying the selected mission configurations in the dashboard. Returns information about the number of unblocked and blocked IP addresses on the firewall.

Parameters:

- **user:** Username for accessing the database REST API. - **passw:** Password to access the database REST API. -
missions_and_configurations: List of pairs (missions, configurations). -
logger: File for logging. -
dashboard_log: Log displayed on the dashboard. -
server_url: URL of the server on which act-overseer is running.

The act-component package The package contains a set of wrappers for 5 defined categories of active defense elements. Each wrapper defines a set of functions for managing an element. All defined functions are necessary to connect the software element of the Act phase. Wrappers do not contain a logical part, because

preservation of generality and the possibility of deployment in different environments with differently implemented elements of active defense. Therefore, it is necessary to supplement this logic and connect the wrapper to the existing access interface of the element before deployment. A detailed specification of the wrapper functions is given in Appendix A.

In addition to five wrappers for PAO, a simulated firewall was also developed for testing the project software without affecting the real network configuration. It is located in the /act-component/simulated-pao-firewall folder. The simulated firewall is represented by a data structure that contains two defined parameters – maximum capacity and blacklist.

The maximum capacity limits the maximum possible number of blacklist items. The blacklist contains a list of currently simulated blocked IP addresses, along with accompanying information defined in the wrapper. Furthermore, the element has implemented all functions defined by its wrapper. This element is a functional whole ready for deployment, combining the functions of the wrapper, the access interface, and the PAO firewall itself.

Appendix A

The appendix contains detailed specifications of wrapper functions, including the definition of data types and the format of input and output parameters. The specifications are described in YAML and are compatible with the OpenAPI standard in version 3.0.0.

```
DNS FW API openapi:
3.0.0 info:

  version: v1
  title: CRUSOE Act API Wrapper for DNS Firewall description:
  API wrapper for Active Network Defense devices of project CRUSOE
servers:
  - description: SwaggerHub API Auto Mocking url:
    'https:// virtserver.swaggerhub.com/ MadGeckoo/ act/ v1' paths: '/
dnsfw/
  health':
    get:
      description: Returns a health check for the DNS FW responses:
      '200':

          description: Successfully returned health check content:

            application/ json:
              schema:
                type: object
                properties:
                  serviceStatus:
                    type: string
        '400':
          description: Invalid request '503':

          description: Service unavailable '/
  dnsfw/ capacity':
    get:
      description: Returns DNS FW capacities
      responses:
        '200':
          description: Successfully returned current capacities content:

            application/ json:
              schema:
                type: object
                properties:
                  maxCapacity:
                    type: integer
                  usedCapacity:
                    type: integer
                  freeCapacity:
```

```
            type: integer
    '400':
      description: Invalid request '403':

      description: Function not supported '/
dnsfw/rules':
  get:
    description: Returns a list of all DNS FW rules
    responses:
      '200':
        description: Successfully returned list of DNS FW rules content:

          application/json:
            schema:
              type: array
              items:
                type: object
                properties:
                  ruleId:
                    type: integer
                  ruleZone:
                    type: string
                  ruleDomain:
                    type: string
                  ruleTarget:
                    type: string
                  ruleReason:
                    type: string
                  ruleNote:
                    type: string
      '400':
        description: Invalid request
  post:
    description: Add DNS FW rule to a DNS FW
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            required: -
              ruleZone
              - rulesDomain
              - ruleTarget
            properties:
              ruleZone:
                type: string
              ruleDomain:
                type: string
              ruleTarget:
                type: string
```

```
        ruleReason:
          type: string
        ruleNote:
          type: string
  responses:
    '200':
      description: Return ID of the added rule content:

      application/json:
        diagram:
          type: object
          properties:
            ruleId:
              type: integer
    '400':
      description: Invalid request '/
dnsfw/rules/{ruleId}':
  get:
    description: Get details of a rule with ruleID from a DNS FW parameters:
    - name: ruleId
      in: path required:
        true
        schema:

          type: integer
    responses:
      '200':
        description: Here are the details
        content:
          application/json:
            schema:
              type: object
              properties:
                ruleId:
                  type: integer
                ruleZone:
                  type: string
                ruleDomain:
                  type: string
                ruleTarget:
                  type: string
                ruleReason:
                  type: string
                ruleNote:
                  type: string
      '400':
        description: Invalid request put:

  description: Change a reason, or zone or target for a rule with ruleID from a DNS FW parameters: - name:
    ruleId
```

```
    in: path
    required: true
    schema:
      type: integer
  requestBody:
    required: true
    content:
      application/json:
        schema:
          type: object
          properties:
            ruleZone:
              type: string
            ruleTarget:
              type: string
            ruleReason:
              type: string
            ruleNote:
              type: string
  responses:
    '200':
      description: OK
      content:
        application/json:
          schema:
            type: object
            properties:
              ruleId:
                type: integer
              ruleZone:
                type: string
              ruleDomain:
                type: string
              ruleTarget:
                type: string
              ruleReason:
                type: string
              ruleNote:
                type: string
    '400':
      description: Invalid request
delete:
  description: Delete a rule with ruleId from a DNS FW parameters:
  - name: ruleId
    in: path
    required: true
    schema:
      type: integer
  responses:
    '200':
```

```
      description: Rule with the ruleId deleted content:

        application/json:
          diagram:
            type: object
            properties:
              ruleId:
                type: integer
              ruleZone:
                type: string
              ruleDomain:
                type: string
              ruleTarget:
                type: string
              ruleReason:
                type: string
              ruleNote:
                type: string
    '400':
      description: Invalid request '/
dnsfw/{ruleDomain}': get:

  description: Get all rules with the specified ruleDomain from a DNS FW parameters: -
    name:
      ruleDomain in: path
      required:
      true schema:

        type: string
    responses:
      '200':
        description: Successfully returned list of rules for the specified ruleDomain content:

          application/json:
            schema:
              type: array
              items:
                type: object
                properties:
                  ruleId:
                    type: integer
                  ruleZone:
                    type: string
                  ruleDomain:
                    type: string
                  ruleTarget:
                    type: string
                  ruleReason:
                    type: string
                  ruleNote:
                    type: string
```

'400':

description: Invalid request '404':

description: Domain not found put:

description: Change a zone or target or reason or note for all rules with the specified ruleDomain parameters: - name: ruleDomain

in: path
required: true
schema:
type: string
requestBody:
required: true
content:
application/ json:
schema:
type: object
properties:
ruleZone:
type: string
ruleTarget:
type: string
ruleReason:
type: string
ruleNote:
type: string
responses:
'200':
description: All rules for the ruleDomain were changed content:

application/ json:
schema:
type: array
items:
type: object
properties:
ruleId:
type: integer
ruleZone:
type: string
ruleDomain:
type: string
ruleTarget:
type: string
ruleReason:
type: string
ruleNote:
type: string
'400':
description: Invalid request

'404':

description: Domain not found

delete:

description: Delete all rules for the specified ruleDomain parameters:

- name:

ruleDomain

in: path

required: true

schema:

type: string

responses:

'200':

description: All rules for the ruleDomain deleted '400':

description: Invalid request '404':

description: Domain not found

Wrapper FW *openapi:*

3.0.0 info:

version: v1

title: CRUSOE Act API Wrapper for firewall

description: API wrapper for Active Network Defense devices of project CRUSOE. Firewall blcks access to the internal services from the outside network.

servers:

- description: SwaggerHub API Auto Mocking url:

'https:// virtserver.swaggerhub.com/ MadGeckoo/ act/ v1' paths: '/ firewall/

health':

get:

description: Returns a health check for firewall

responses:

'200':

description: Successfully returned health check content:

application/ json:

schema:

type: object

properties:

serviceStatus:

type: string

'400':

description: Invalid request '503':

description: Service unavailable '/ firewall/ capacity':

get:

description: Returns capacities for firewall

responses:

```
      '200':
        description: Successfully returned current capacities content:

          application/ json:
            schema:
              type: object
              properties:
                maxCapacity:
                  type: integer
                usedCapacity:
                  type: integer
                freeCapacity:
                  type: integer
      '400':
        description: Invalid request '403':

        description: Function not supported
  '/ firewall/ blocked':
    get:
      description: Returns a list of blocked IP addresses for firewall responses:
      '200':

        description: Successfully returned list of blocked IPs content:

          application/ json:
            schema:
              type: array
              items:
                type: object
                properties:
                  ruleId:
                    type: integer
                  ruleIp:
                    type: string
                  rulePort:
                    type: integer
                  ruleReason:
                    type: string
      '400':
        description: Invalid request
    post:
      description: Block an IP on firewall, port is optional
      requestBody:
        required: true
        content:
          application/ json:
            schema:
              type: object
              required: -
                ruleIp
              properties:
```

```
        ruleIp:
          type: string
        rulePort:
          type: integer
        ruleReason:
          type: string
  responses:
    '200':
      description: Return ID of a given IP address block rule content:

        application/json:
          schema:
            type: object
            properties:
              ruleId:
                type: integer
    '400':
      description: Invalid request
'/firewall/blocked/{blockedId}':
  get:
    description: Get details of a rule with ruleID from firewall parameters:
    - name:
      blockedId in: path
      required:
      true schema:

        type: integer
    responses:
      '200':
        description: Here are the details
        content:
          application/json:
            schema:
              type: object
              properties:
                ruleId:
                  type: integer
                ruleIp:
                  type: string
                rulePort:
                  type: integer
                ruleReason:
                  type: string
      '400':
        description: Invalid request
  delete:
    description: Delete a rule with blockedId from firewall
    parameters: -
      name: blockedId
      in: path
      required: true
```

```
      diagram:
        type: integer
    responses:
      '200':
        description: Rule for blockedId deleted content:

          application/json:
            diagram:
              type: object
              properties:
                ruleId:
                  type: integer
                ruleIp:
                  type: string
                rulePort:
                  type: integer
                ruleReason:
                  type: string
      '400':
        description: Invalid request '/
firewall/blocked/{blockedId}/port': put:

  description: Change port for a rule with ruleID on the firewall parameters:
    - name:
      blockedId in: path
        required:
        true schema:

          type: integer
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              rulePort:
                type: integer
    responses:
      '200':
        description: Return the new entry
        content:
          application/json:
            schema:
              type: object
              properties:
                ruleId:
                  type: integer
                ruleIp:
                  type: string
                rulePort:
```

```
                type: integer
              ruleReason:
                type: string
      '400':
        description: Invalid request '/
firewall/ blocked/ {blockedId}/ reason': put:

  description: Change reason for a rule with ruleID on the firewall parameters: -
    name: blockedId

      in: path
      required: true
      schema:
        type: integer
    requestBody:
      required: true
      content:
        application/ json:
          schema:
            type: object
            properties:
              ruleReason:
                type: string
    responses:
      '200':
        description: Return the new entry
        content:
          application/ json:
            schema:
              type: object
              properties:
                ruleId:
                  type: integer
                ruleIp:
                  type: string
                rulePort:
                  type: integer
                ruleReason:
                  type: string
      '400':
        description: Invalid request '/
firewall/ {blockedIp}': get:

  description: Get all rules with specified blockedIp on firewall parameters:
    - name:
      blockedIp in: path
        required:
        true schema:

          type: string
    responses:
```

```
'200':
  description: Successfully returned list of rules for given IP content:

    application/ json:
      schema:
        type: array
        items:
          type: object
          properties:
            ruleId:
              type: integer
            ruleIp:
              type: string
            rulePort:
              type: integer
            ruleReason:
              type: string
'400':
  description: Invalid request '404':

  description: IP not found put:

description: Change the reason for all rules with specified IP parameters: -
  name:
    blockedIp in: path
    required:
    true schema:

      type: string
  requestBody:
    required: true
    content:
      application/ json:
        schema:
          type: object
          required: -
            ruleReason
          properties:
            ruleReason:
              type: string
  responses:
    '200':
      description: Reason changed successfully content:

        application/ json:
          schema:
            type: array
            items:
              type: object
              properties:
                ruleId:
```

```
          type: integer
        ruleIp:
          type: string
        rulePort:
          type: integer
        ruleReason:
          type: string
    '400':
      description: Invalid request '404':

      description: IP not found
  delete:
    description: Delete all rules for IP
    parameters: -
      name: blockedIp in:
        path
        required: true
        schema:
          type: string
    responses:
      '200':
        description: All rules for the IP deleted '400':

      description: Invalid request '404':

      description: IP not found '/
firewall/{blockedIp}/{blockedPort}':
  delete:
    description: Delete a rule containing the IP and port
    parameters: -
      name: blockedIp in:
        path
        required: true
        schema:
          type: integer -
      name: blockedPort
        in: path
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: Rule deleted
      '400':
        description: Invalid request '404':

        description: IP or port not found
```

Wrapper Mail Filter openapi: 3.0.0

```
info:
  version: v1
  title: CRUSOE Act API Wrapper for Mail Filter
  description: API wrapper for Active Network Defense devices of project CRUSOE
servers:
  - description: SwaggerHub API Auto Mocking url:
    'https:// virtserver.swaggerhub.com/ MadGeckoo/ act/ v1' paths: '/

mailFilter/ health': get:

    description: Returns a health check for mail filter
      responses:
        '200':
          description: Successfully returned health check content:

            application/ json:
              schema:
                type: object
                properties:
                  serviceStatus:
                    type: string
        '400':
          description: Invalid request '503':

          description: Service unavailable '/
  mailFilter/ capacity': get:

    description: Returns mail filter capacities for mail filter responses:
      '200':

          description: Successfully returned current capacities content:

            application/ json:
              schema:
                type: object
                properties:
                  maxCapacity:
                    type: integer
                  usedCapacity:
                    type: integer
                  freeCapacity:
                    type: integer
        '400':
          description: Invalid request '403':

          description: Function not supported '/
  mailFilter/ blocked':
    get:
      description: Returns a list of e-mail addresses blocked by mailFilter
      responses:
        '200':
```

```
      description: Successfully returned list of blocked e-mails content:


        application/json:
          diagram:

            type: array
            items:

              type: object

              properties:
                ruleId:

                  type: integer
                ruleAddress:

                  type: string
                ruleFrom:

                  type: boolean
                ruleTo:

                  type: boolean
                ruleReason:

                  type: string
    '400':

      description: Invalid request
post:
  description: Block an e-mail address by mailFilter
  requestBody:
    required: true
    content:

      application/json:
        schema:

          type: object

          required: -
            ruleAddress

            - ruleFrom

            - ruleTo

          properties:
            ruleAddress:

              type: string
            ruleFrom:

              type: boolean
            ruleTo:

              type: boolean
            ruleReason:

              type: string
  responses:
    '200':

      description: Return ID of a given e-mail address block rule content:


        application/json:
          diagram:

            type: object

            properties:
              ruleId:

                type: integer
```

```
    '400':
      description: Invalid request
'/mailFilter/blocked/{ruleId}':
  get:
    description: Get details of a rule with the ruleID from mailFilter parameters:
    - name: ruleId
      in: path required:
        true
        schema:

          type: integer
    responses:
      '200':
        description: Here are the details
        content:
          application/json:
            schema:
              type: object
              properties:
                ruleId:
                  type: integer
                ruleAddress:
                  type: string
                ruleFrom:
                  type: boolean
                ruleTo:
                  type: boolean
                ruleReason:
                  type: string
      '400':
        description: Invalid request put:

  description: Change a reason, from and to for a rule with ruleID on the mailFilter parameters: -
    name: ruleId
      in: path required:
        true
        schema:

          type: integer
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            required: -
              ruleFrom
              - ruleTo
              - ruleReason
            properties:
              ruleFrom:
```

```
        type: boolean
      ruleTo:
        type: boolean
      ruleReason:
        type: string
  responses:
    '200':
      description: Changed, return the new entry content:

      application/ json:
        schema:
          type: object
          properties:
            ruleId:
              type: integer
            ruleAddress:
              type: string
            ruleFrom:
              type: boolean
            ruleTo:
              type: boolean
            ruleReason:
              type: string
    '400':
      description: Invalid request
delete:
  description: Delete a rule with ruleId from the mailFilter parameters:
  - name: ruleId
    in: path required:
      true
      schema:

        type: integer
  responses:
    '200':
      description: MailFilter rule with ruleId deleted content:

      application/ json:
        schema:
          type: object
          properties:
            ruleId:
              type: integer
            ruleAddress:
              type: string
            ruleFrom:
              type: boolean
            ruleTo:
              type: boolean
            ruleReason:
              type: string
```

'400':

description: Invalid request '/

mailFilter/{ruleAddress}': get:

description: Get all rules with specified ruleAddress parameters:

- name:

ruleAddress

in: path

required: true

schema:

type: string

responses:

'200':

description: Successfully returned list of rules for given e-mail address content:

application/json:

schema:

type: array

items:

type: object

properties:

ruleId:

type: integer

ruleAddress:

type: string

ruleFrom:

type: boolean

ruleTo:

type: boolean

ruleReason:

type: string

'400':

description: Invalid request '404':

description: E-mail address not found put:

description: Change a reason and direction for all rules with the specified e-mail address parameters: - name:

ruleAddress

in: path required: true

schema:

type: string

requestBody:

required: true

content:

application/json:

schema:

type: object

required: -

ruleFrom

```
            - ruleTo
            - ruleReason
          properties:
            ruleFrom:
              type: boolean
            ruleTo:
              type: boolean
            ruleReason:
              type: string
      responses:
        '200':
          description: OK
          content:
            application/json:
              schema:
                type: array
                items:
                  type: object
                  properties:
                    ruleId:
                      type: integer
                    ruleAddress:
                      type: string
                    ruleFrom:
                      type: boolean
                    ruleTo:
                      type: boolean
                    ruleReason:
                      type: string
        '400':
          description: Invalid request '404':

          description: e-mail address not found delete:

      description: Delete all rules for specified e-mail address and direction (from or to) parameters: -
      name:
        ruleAddress in: path
          required:
          true schema:

            type: string
      responses:
        '200':
          description: All rules for given e-mail address and direction deleted '400':

          description: Invalid request '404':

          description: No rules for the given address found
  /mailFilter/from:
    get:
      description: Get all rules with specified ruleAddress in 'from' direction
```

```
    responses:
      '200':
        description: Successfully returned list of rules for 'from' direction content:


          application/json:
            diagram:

              type: array
              items:

                type: object
                properties:
                  ruleId:

                    type: integer
                  ruleAddress:

                    type: string
                  ruleFrom:

                    type: boolean
                  ruleTo:

                    type: boolean
                  ruleReason:

                    type: string
      '400':
        description: Invalid request '404':


      description: No e-mails blocked in 'from' direction
  /mailFilter/to:
    get:
      description: Get all rules with specified ruleAddress 'to' direction
      responses:
        '200':
          description: Successfully returned list of rules for 'to' direction content:


            application/json:
              schema:

                type: array
                items:

                  type: object
                  properties:
                    ruleId:

                      type: integer
                    ruleAddress:

                      type: string
                    ruleFrom:

                      type: boolean
                    ruleTo:

                      type: boolean
                    ruleReason:

                      type: string
        '400':
          description: Invalid request '404':


          description: No e-mails blocked in 'to' direction
```

Wrapper RTBH *openapi:*
3.0.0 info:

version: v1
title: CRUSOE Act API Wrapper for RTBH
description: API wrapper for Active Network Defense devices of project CRUSOE
servers:
- description: SwaggerHub API Auto Mocking url:
'https:// virtserver.swaggerhub.com/ MadGeckoo/ act/ v1' paths: '/ rtbh/
health':

get:
description: Returns a health check for rtbh
responses:
'200':
description: Successfully returned health check content:

application/ json:
schema:
type: object
properties:
serviceStatus:
type: string
'400':
description: Invalid request '503':

description: Service unavailable
'/ rtbh/ capacity':
get:
description: Returns capacities for rtbh
responses:
'200':
description: Successfully returned current capacities content:

application/ json:
schema:
type: object
properties:
maxCapacity:
type: integer
usedCapacity:
type: integer
freeCapacity:
type: integer
'400':
description: Invalid request '403':

description: Function not supported '/ rtbh/
blocked':

```
get:
  description: Returns a list of blocked IP addresses for RTBH interface
  responses:
    '200':
      description: Successfully returned list of blocked IPs content:

        application/ json:
          diagram:
            type: array
            items:
              type: object
              properties:
                ruleId:
                  type: integer
                ruleIp:
                  type: string
                ruleReason:
                  type: string
    '400':
      description: Invalid request post:

  description: Block an IP on RTBH interface
  requestBody:
    required: true
    content:
      application/ json:
        schema:
          type: object
          required: -
            ruleIp
          properties:
            ruleIp:
              type: string
            ruleReason:
              type: string
  responses:
    '200':
      description: Return ID of a given IP address block rule content:

        application/ json:
          schema:
            type: object
            properties:
              ruleId:
                type: integer
    '400':
      description: Invalid request '/
rtbh/ blocked/ {blockedId}': get:

  description: Get details of a rule with ruleID from RTBH parameters:
```

```
    - name: blockedId
      in: path
      required: true
      schema:
        type: integer
  responses:
    '200':
      description: Here are the details
      content:
        application/json:
          schema:
            type: object
            properties:
              ruleId:
                type: integer
              ruleIp:
                type: string
              ruleReason:
                type: string
    '400':
      description: Invalid request put:

description: Change a reason for a rule with ruleID on the RTBH parameters: -
  name:
    blockedId in: path
      required:
      true schema:

        type: integer
  requestBody:
    required: true
    content:
      application/json:
        schema:
          type: object
          required: -
            ruleReason
          properties:
            ruleReason:
              type: string
  responses:
    '200':
      description: Reason changed, return the new entry content:

        application/json:
          schema:
            type: object
            properties:
              ruleId:
                type: integer
              ruleIp:
```

```
                type: string
              ruleReason:
                type: string
    '400':
      description: Invalid request
  delete:
    description: Delete a rule with blockedId from RTBH parameters:
    - name:
      blockedId in: path
      required:
      true schema:

        type: integer
    responses:
      '200':
        description: Rule for blockedId deleted content:

          application/ json:
            schema:
              type: object
              properties:
                ruleId:
                  type: integer
                ruleIp:
                  type: string
                ruleReason:
                  type: string
      '400':
        description: Invalid request '/
rtbh/ {blockedIp}':
  get:
    description: Get all rules with specified blockedIp on RTBH parameters:
    - name:
      blockedIp in: path
        required:
        true schema:
        type: string

    responses:
      '200':
        description: Successfully returned list of rules for given IP content:

          application/ json:
            schema:
              type: array
              items:
                type: object
                properties:
                  ruleId:
                    type: integer
                  ruleIp:
```

```
                type: string
              ruleReason:
                type: string
    '400':
      description: Invalid request '404':

      description: IP not found put:

description: Change the reason for all rules with specified IP parameters:
  - name:
    blockedIp in: path
      required:
      true schema:

        type: string
  requestBody:
    required: true
    content:
      application/json:
        schema:
          type: object
          required: -
            ruleReason
          properties:
            ruleReason:
              type: string
  responses:
    '200':
      description: OK
      content:
        application/json:
          schema:
            type: array
            items:
              type: object
              properties:
                ruleId:
                  type: integer
                ruleIp:
                  type: string
                ruleReason:
                  type: string
    '400':
      description: Invalid request '404':

      description: IP not found
delete:
  description: Delete all rules for specified IP
  parameters: -
    name: blockedIp in:
      path
```

```
        required: true
        schema:
          type: string
      responses:
        '200':
          description: All rules for the IP deleted '400':

          description: Invalid request '404':

        description: IP not found


Wrapper User Blocker openapi: 3.0.0
info:

  version: v1
  title: CRUSOE Act API Wrapper for User Blocker description:
  API wrapper for Active Network Defense devices of project CRUSOE
servers:
  - description: SwaggerHub API Auto Mocking url:
    'https:// virtserver.swaggerhub.com/ MadGeckoo/ act/ v1' paths: '/

userBlock/ health':
    get:
      description: Returns a health check for user block interface with given id responses:
      '200':

        description: Successfully returned health check content:

          application/ json:
            schema:
              type: object
              properties:
                serviceStatus:
                  type: string
        '400':
          description: Invalid request '503':

        description: Service unavailable '/
  userBlock/ capacity': get:

    description: Returns user block capacities for user block interface with given id
      responses:
        '200':
          description: Successfully returned current capacities content:

            application/ json:
              diagram:
                type: object
                properties:
                  maxCapacity:
```

```
        type: integer
      usedCapacity:
        type: integer
      freeCapacity:
        type: integer
  '400':
    description: Invalid request '403':

    description: Function not supported '/
userBlock/blocked':
  get:
    description: Returns a list of blocked users for userBlock
    responses:
      '200':
        description: Successfully returned list of blocked users content:

          application/json:
            schema:
              type: array
              items:
                type: object
                properties:
                  ruleId:
                    type: integer
                  ruleUser:
                    type: string
                  ruleBlockedFrom:
                    type: string
                  ruleBlockedTo:
                    type: string
                  ruleReason:
                    type: string
      '400':
        description: Invalid request
  post:
    description: Block a user by userBlock
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            required: -
              ruleUser
              - ruleBlockedFrom
              - ruleBlockedTo
              - ruleReason
            properties:
              ruleUser:
                type: string
              ruleBlockedFrom:
```

```
            type: string
          ruleBlockedTo:
            type: string
          ruleReason:
            type: string
    responses:
      '200':
        description: Return ID of a given user block rule content:

        application/json:
          schema:
            type: object
          properties:
            ruleId:
              type: integer
      '400':
        description: Invalid request
  '/userBlock/blocked/{ruleId}':
    get:
      description: Get details of a rule with ruleID from userBlock parameters:
      - name: ruleId

        in: path
        required: true
        schema:
          type: integer
      responses:
        '200':
          description: Here are the details
          content:
            application/json:
              schema:
                type: object
              properties:
                ruleId:
                  type: integer
                ruleUser:
                  type: string
                ruleBlockedFrom:
                  type: string
                ruleBlockedTo:
                  type: string
                ruleReason:
                  type: string
        '400':
          description: Invalid request put:

    description: Change a reason or date of the user block end for a rule with ruleID on the userBlock parameters: - name:
      ruleId

        in: path
```

```
      required: true
      schema:
        type: integer
  requestBody:
    required: true
    content:
      application/json:
        schema:
          type: object
          required:
            - ruleTo
            - ruleReason
          properties:
            ruleBlockedTo:
              type: string
            ruleReason:
              type: string
  responses:
    '200':
      description: OK
      content:
        application/json:
          schema:
            type: object
            properties:
              ruleId:
                type: integer
              ruleUser:
                type: string
              ruleBlockedFrom:
                type: string
              ruleBlockedTo:
                type: string
              ruleReason:
                type: string
    '400':
      description: Invalid request
delete:
  description: Delete a rule with ruleId from userBlock
  parameters:
    - name: ruleId
      in: path
      required: true
      schema:
        type: integer
  responses:
    '200':
      description: UserBlock rule with ruleId deleted
      content:
        application/json:
          schema:
            diagram:
```

```
                type: object
                properties:
                  ruleId:
                    type: integer
                  ruleUser:
                    type: string
                  ruleBlockedFrom:
                    type: string
                  ruleBlockedTo:
                    type: string
                  ruleReason:
                    type: string
      '400':
        description: Invalid request '/
userBlock/{user}':
  get:
    description: Get all rules for user
    parameters:
      - name: user
        in: path
        required: true
        schema:
          type: string
    responses:
      '200':
        description: Successfully returned list of rules for given user content:

          application/json:
            schema:
              type: array
              items:
                type: object
                properties:
                  ruleId:
                    type: integer
                  ruleUser:
                    type: string
                  ruleBlockedFrom:
                    type: string
                  ruleBlockedTo:
                    type: string
                  ruleReason:
                    type: string
      '400':
        description: Invalid request '404':

        description: User not found put:

  description: Change a reason and expiration date for all rules for specified user parameters:

      - name: user
```

```
    in: path
    required: true
    schema:
      type: string
  requestBody:
    required: true
    content:
      application/json:
        schema:
          type: object
          properties:
            ruleBlockedTo:
              type: string
            ruleReason:
              type: string
  responses:
    '200':
      description: OK
      content:
        application/json:
          schema:
            type: array
            items:
              type: object
              properties:
                ruleId:
                  type: integer
                ruleUser:
                  type: string
                ruleBlockedFrom:
                  type: string
                ruleBlockedTo:
                  type: string
                ruleReason:
                  type: string
    '400':
      description: Invalid request '404':

      description: User not found
delete:
  description: Delete all rules for specified user
  parameters:
    - name: user
      in: path
      required: true
      schema:
        type: string
  responses:
    '200':
      description: All rules for given user deleted '400':
```

description: Invalid request

'404':

description: No rules for given user found

Appendix B

The attachment contains a detailed specification of the act-overseer REST API access interface. The specification is described in YAML and is compatible with the OpenAPI standard in version 3.0.0.

```
openapi: 3.0.0
info:
  version: v1
  title: CRUSOE Act API for Act Overseer description:
  API specifying the access points to the ACT phase for other phases
servers:
  - description: SwaggerHub API Auto Mocking url: 'https://
    virtserver.swaggerhub.com/MadGeckoo/act/v1' paths: '/act/


protect_missions_assets':
    post:
      description: Act expects to receive a json with the list of the mission-configuration pairs.
All missions are required, with exactly one configuration chosen per mission.
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: array
              items:
                type: object
                required:
                  - on me
                  - config_id
                properties:
                  on me:
                    type: string
                  config_id:
                    type: string
      responses:
        '200':
          description: Applying configurations '403':

          description: Mission list does not match the list in neo4j, ie is not complete '404':

          description: Configuration ID does not exist '400':

          description: Invalid request
  '/act/threshold':
```

```
get:
  description: Returns the current security threshold, devices with a lower rating will be blocked

  responses:
    '200':
      description: Successfully returned threshold content:

        application/json:
          schema:
            type: object
            properties:
              threshold:
                type: integer
    '400':
      description: Invalid request put:

description: Changes current security threshold to a new one requestBody:
  required: true
    content:

      application/json:
        schema:
          type: object
          properties:
            threshold:
              type: integer
  responses:
    '200':
      description: New threshold set '400':

    description: Invalid request
```

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 authors of the software are Stanislav Špaÿek and Milan Žiaran.