Sharing and Automation for
Privacy Preserving Attack Neutralization

(H2020 833418)

D6.1 SAPPAN Dashboard, first version (M21)

Published by the SAPPAN Consortium

Dissemination Level: Public

H2020-SU-ICT-2018-2020 – Cybersecurity

Document control page

Document file: Deliverable 6.1
Document version: 1.0
Document owner: Robert Rapp (USTUTT)

Work package: WP6
Task: T6.1
Deliverable type: DEM
Delivery month: M21
Document status: ☒ approved by the document owner for internal review
☒ approved for submission to the EC

Document History:

Version	Author(s)	Date	Summary of changes made
0.1	Christoph Müller (USTUTT)	2021-01-25	Initial version of the document
0.2	Robert Rapp (USTUTT)	2021-01-27	Revised document in order to internal review
0.3	Robert Rapp (USTUTT)	2021-01-29	Improvements made on base of internal review
1	Robert Rapp (USTUTT)	2021-01-29	First version of the document

Internal review history:

Reviewed by	Date	Summary of comments
Martin Zadnik (CESNET)	2021-01-27	Suggestions for wording improvement.
Tomas Jirsik (MU)	2021-01-28	Grammar and Spelling check, non-technical review

Executive Summary

This first iteration of the description of the SAPPAN dashboard reports on the progress in context of T6.1 in developing a web-based dashboard application that serves as the integration point for all visualisation-related tasks throughout the SAPPAN project. After a short review of the task and its relation to other tasks, the deliverable illustrates the dashboard from an end-user perspective before explaining the architecture of the application and detailing on important implementation aspects. The latter include visualisation components that have been developed in context of T6.1 in order to test and demonstrate the functionality of the dashboard. The document concludes by reviewing the current state of the implementation and work that has still to be carried out.

Table of Contents

1 Introduction

This deliverable reports on progress of the SAPPAN Dashboard, which is being developed in context of task T6.1 “Dashboard for response and recovery awareness”. This report describes the first iteration of the task, which constitutes the base system and a few prototypical visualisations to demonstrate the concepts. It will be followed up by an updated, final version in M33.

Based on the findings of D2.3 “Visualisation requirements”, the dashboard is designed as a web-based application, which is used to integrate visualisation inputs developed within WP3, WP4 and WP5. Visualising this input, it creates new ways of interacting with the data. Interactive visualisation provides a clear and visually supported view of the data, what adds another layer of presentation to the analytical work and providing an alternative to view raw log files. It will be eventually integrated with T6.2 “SAPPAN demonstrator” as the source of data being visualised.

In the following, we continue by describing the role of the dashboard in the overall context of the project, illustrating its functionality from a user point of view and elaborating on important aspects of the system design and implementation.

2 SAPPAN Context

The SAPPAN concept distinguishes the local response and recovery level, which refers to the data collection, threat detection and handling within a single organisation, from the global one, which is added by means of the sharing system. While WP3 and WP4 mostly deal with the local level, WP5 add the global level and WP6 integrates all of the aforementioned result in a demonstrator for evaluation. The SAPPAN dashboard is the user interface of the project and therefore needs to address both levels. As it depends on the results and data from the other work packages and therefore requires some level of integration, we decided it to be part of the overall demonstration and evaluation activities in WP6. The SAPPAN dashboard itself is not a research result of the project by its own means, but it is the foundation and test bed for the visualisation-related research performed in other work packages.

From a system-architecture point of view, the dashboard needs to be able to access data from the local organisation and is therefore more privileged than the sharing system itself. It also needs to access the sharing platform in order to display events generated on the global level and to display requests and results of sharing information for model training.

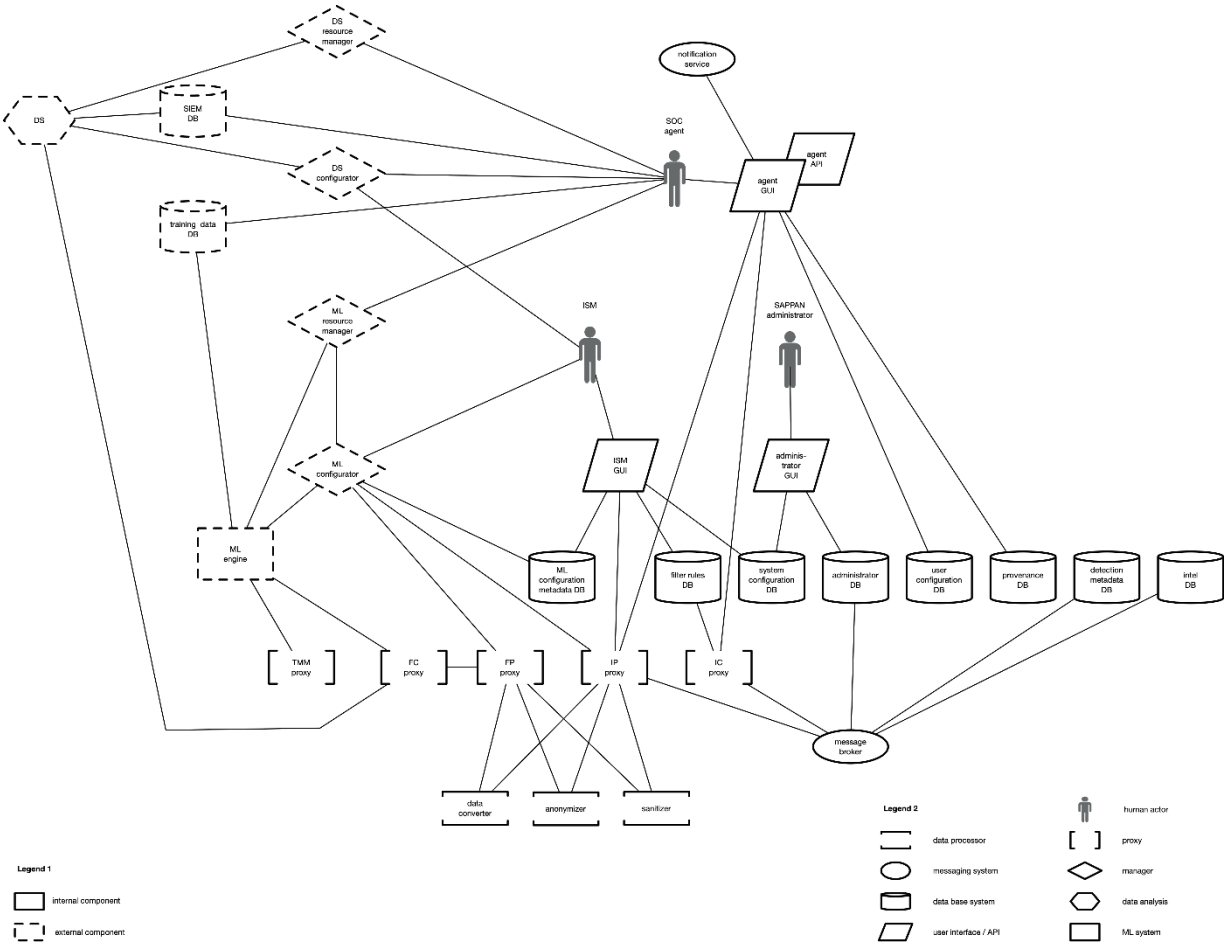

Figure 1: The SAPPAN system architecture

In the system architecture shown in Figure 1, we deliberately chose to have three concepts of a user interface, the Agent GUI and the ISM GUI and the Administrator GUI without defining whether all of these roles would be fulfilled by a single interface or not. As the project progressed, regard the reasons in D5.7 the MISP platform was chosen as the basis for the SAPPAN sharing system. MISP comes with its own user interface for administration of its instances, which fulfils the role of the ISM GUI, where an administrator of each organisation can define what is being shared, and the Administrator GUI for managing the sharing platform itself. The SAPPAN dashboard developed in T6.1 and described in this deliverable therefore represents only the Agent GUI, which is presented to the analysts in a security operations centre (SOC).

3 Related work

The term “dashboard” as a user interface is borrowed from cars where the indicators of the dashboard inform the driver continuously about the state of the car. For its use in the sense of an interactive tool for reporting in a management context [1], a widely used definition was proposed by Fry as “A dashboard is a visual display of the most important information needed to achieve one or more objectives; consolidated and arranged on a single screen so the information can be monitored at a glance.”, which he later clarified as dashboards being used to monitor whereas “faceted analytical displays” would “combine several charts [...] for the purpose of analysis” [2]. In the business management context, a dashboard typically displays multiple critical key performance indicator, if possible in real time, and is responsive with respect to automatically

displaying alerts when predefined thresholds are reached [1]. In that sense, the SAPPAN dashboard does meet the definition of a dashboard, because it was envisioned as a Visual Analytics [3] application that allows for interactive exploration of data and not a pure display of what is going on. However, as the concept of dashboards is used in different domains including software development [4], manufacturing [5], learning [6] and network security [7] and the technology to build them evolved, the term is also used for interfaces that enable interactive reasoning.

4 Description of the SAPPAN Dashboard

The SAPPAN dashboard is an application that is used for the visual first-approach to analyses in a SOC. The primary use case for the dashboard is therefore the analysis of incidents which have been detected by external detection code. The main design decisions for the dashboard are motivated by the findings described in D2.3 “Visualisation requirements”, which mainly resulted in (i) a web-based application as this is the primary type of tools used in all SOCs we investigated and (ii) a multiple-coordinated-views design that allows for complementing standard data visualisation techniques known to the analysts with more complex ones.

The main components of the dashboard are a navigation bar for accessing secondary features like the configuration information, a responsive content area for, which is the primary work area for the analysts, and a toolbar as shown in Figure 2.

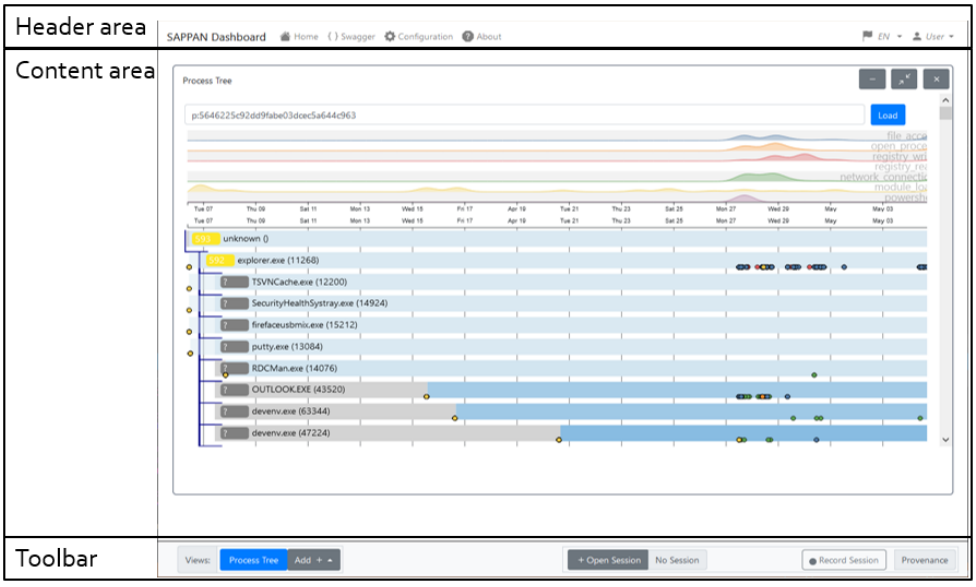

Figure 2: The SAPPAN dashboard

The content area allows for a flexible, user-defined arrangement of different visualisations in resizable and movable cards. Furthermore, cards can be closed or minimised, which gives the user all the functions needed to configure the dashboard content according to the needs of an ongoing analysis. In order to test and demonstrate this functionality, we developed two exemplary types of visualisation content for two distinct types of data, the host-based events collected by F-Secure’s RDR sensor and NetFlow data collected from switches. While the latter are displayed as standard charts, augmented by a decomposition of the time series, the former are used to extract a visual

representation of process trees that conveys not only the parent-child relationship between processes, but also the temporal course of the events. The two exemplary visualisations are described in more detail in Section 5.3.

The toolbar provides an overview of the currently open cards at the bottom left. Furthermore, new cards with new visualisations can be opened from there. Finally, the toolbar is the place where functionality related to tracking of analytical provenance is integrated, which are described in greater detail in D4.8. The user has the option to start to 'Record [a] session', which informs the system about the start of an analysis of a new incident. In a subsequent dialogue, users must enter a short description of this incident. Afterwards, all events recorded by the dashboard are associated with said incident. In order to make recorded sessions more useful, users can add comments to important steps that have been logged by the system.

Previously recorded sessions can be recalled via the button 'Open session' in the toolbar, which shows a list of the available sessions. If a session is selected, it shows then the state of the dashboard in the beginning of the analysis session. Subsequently, the buttons 'Next' and 'Undo' can be used to follow the recorded interactions step by step.

5 Prototype

5.1 Architecture

The architecture of the SAPPAN dashboard is based on core concepts outlined in the overall SAPPAN architecture (cf. Section 2). The dashboard is a web-based application (cf. Figure 3) to meet one of the core requirements identified in D2.3, which is meeting analysts where they currently are in terms of user interfaces, visualisation and interaction.

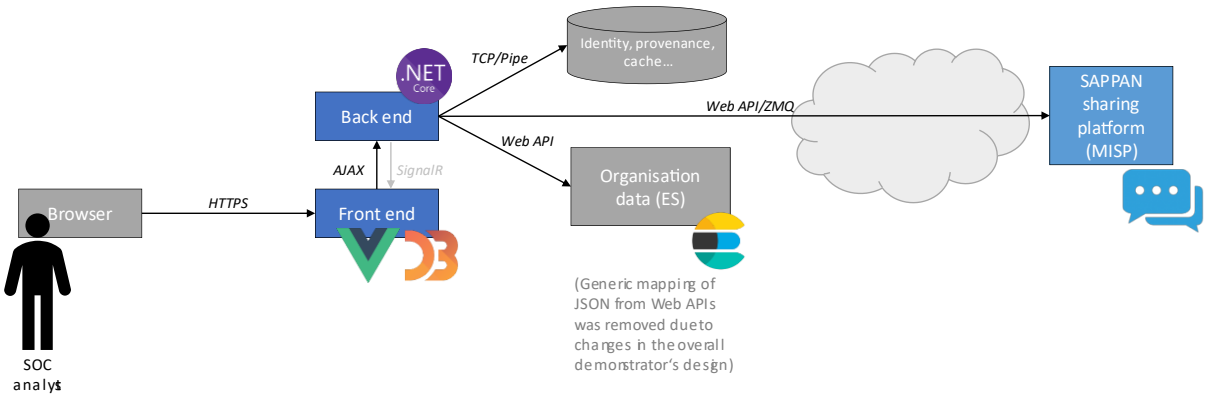

Figure 3: Architecture of the SAPPAN dashboard

The architecture depicted in Figure 3 follows modern design approach: it consists of an HTML- and JavaScript-based front-end application accessed via a web browser and a back end for performing costly computation and accessing sensitive data.

The front end uses Vue.js[1] as application framework and the ubiquitous Data-driven Documents (D3.js[2]) for building individual visualisations. Communication with the back end is implemented using a Web API accessed via AJAX calls. The whole functionality

[1] The Progressive JavaScript Framework: https://vuejs.org/ *last checked on 29.01.2021*
[2] Data-driven Documents Visualisation Framework: https://d3js.org/ *last checked on 29.01.2021*

of the back end is exposed via this Web API. The fact that the front-end uses the same API ensures by-design that there can be no functions that only the UI can do, but not the API. This makes the back-end an interface to the data layer and provides the ability to use the shared information for other applications in SOCs – as envisioned for the Agent API in the SAPPAN architecture definition (cf. Figure 1).

The back end is written in ASP.NET Core (currently version 3, but this might be upgraded to version 5 for the final version). All of the Web APIs are mapped to individual controllers which are responsible for a specific area of tasks, for instance accessing data on the sharing platform or accessing data for a specific visualisation. The back end uses its own database, at the moment for two tasks: storing identity information and recording analytical provenance (T4.8). It also connects to the local organisation and its SIEM and other data via a Web API. For the purpose of demonstrating the SAPPAN solution, we decided to use Elastic Search (ES) as a stand-in for the local organisation for two reasons: first, it is a system that provides a Web API out of the box, and second, it is the "native" way of storing the host-based events we collect in the SAPPAN project. Finally, the back end uses the Web API of MISP to access the SAPPAN sharing platform. In the future, we might add ZeroMQ as well if quick updates on new entries in the platform are required in the dashboard.

5.2 Implementation details

5.2.1 Project organisation

We split the dashboard project into several sub-projects, which are more or less self-contained libraries, for two main reasons: first, having certain functionality in such libraries makes it easier to test it, and second, such libraries might be reused in future projects. In Figure 4 an overview is given which framework and librarys are used for the back-end and front-end functionalities. The latter specifically holds for two libraries we have already published on the VIS institute's public GitHub site, the first is our .NET Core port[3] of the Seasonal Decomposition of Time Series by Loess (STL) and the second is a native .NET Standard library[4] of the CryptoPAn algorithm [8], used for pseudonymisation of IP addresses. While experimenting with data for the dashboard we developed a support library for interpreting raw NetFlow and IPFIX streams, which we eventually plan to separate from the SAPPAN dashboard and publish for general reuse.

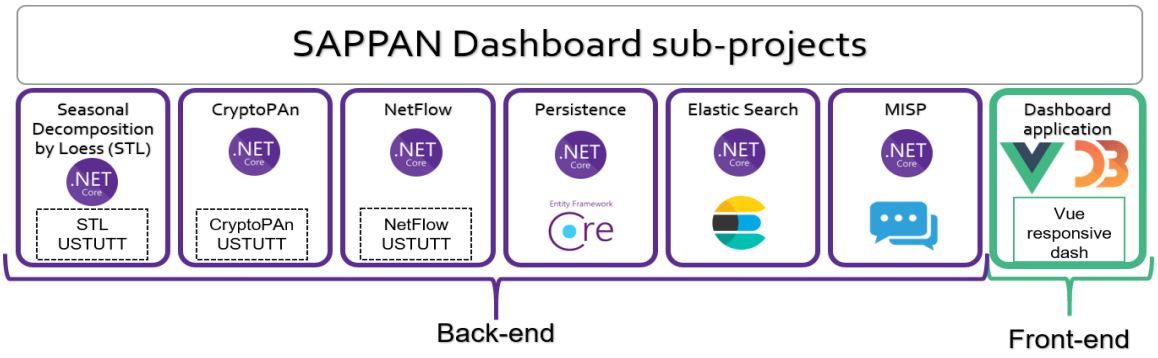

Figure 4: Sub-project organisation and used frameworks

[3] STL Alogrithm: 2020 *https://github.com/UniStuttgart-VISUS/Visus.Stl* [*checked on 29.01.2021]*
[4] CryptoPAn Alogrithm: 2020 *https://github.com/UniStuttgart-VISUS/Sappan.CryptoPAn* [*checked on 29.01.2021]*

Besides the aforementioned libraries, the dashboard itself is based on a library handling the persistence/database layer, another handling the communication with MISP, i.e. the SAPPAN sharing platform, and a third handling communication with an Elastic Search cluster, i.e. the local organisation in the demonstrator scenario.

Finally, the dashboard itself depends on all of the aforementioned libraries and implements both, the controllers exposing the functionality of the back end via API controllers and the front-end application mostly written in TypeScript. Technically, these two could be separated, but are kept together in the current state as it eases testing and debugging of the solution.

5.2.2 CryptoPAn library

The collection of data for the demonstrator might require the anonymisation of IP addresses. The CryptoPAn algorithm, which provides a pseudonymisation that preserves the subnet structure of the data, is a widely used method for this purpose. Provided the same AES key is used, the algorithm yields a deterministic result, which enables correlating NetFlow data obtained from switches with other data sources, even after pseudonymisation. As the original concept of the SAPPAN dashboard envisioned the back end to directly connect to data sources within the local organisation, we provided the means to perform this pseudonymisation there. However, this functionality is not used in the current iteration anymore as the current concept of the demonstrator represents the local organisation as an instance of Elastic Search. Pseudonymisation happens before storing the data in ES. The library is therefore used during import at USTUTT, who collect testing data from the real-world institute infrastructure, which requires pseudonymisation.

5.2.3 STL library

The STL algorithm [9] decomposes a time series like NetFlow data into several components representing different characteristics of the data. These components are the seasonal component, which e.g. captures the weekly pattern of traffic on workdays having a larger volume than on the weekend, the trend component capturing the long-term behaviour of the data like continuously increasing traffic and the residual or remainder, which describes the part of the data that cannot be expressed by the seasonal or trend part. This way, it highlights all irregularities in the data. Pursuing our objective of building visualisations in the dashboard based on the ones known to SOC staff, we decided to augment the commonly used time series displays of network traffic with an STL decomposition, which this implementation is used for.

5.2.4 NetFlow library

The NetFlow library was developed with the idea of connecting the dashboard directly to organisation-specific data sources in mind. It is currently not used any more as the overall architecture changed. Technically, the library provides means to read and write NetFlow and IPFIX streams in .NET Core. As such, it could be used to enable the dashboard to serve as the NetFlow sink directly.

5.2.5 Persistence library

The persistence library encapsulates the interface to a per-organisation relational database used to store provenance and user identity information. We opted for a code-first approach based on the Entity Framework Core (EF Core), which defines the data model first and generates the database schema from the code and some annotations in it. These annotations mostly cover the primary and foreign keys as this typically

cannot be derived automatically from the data model. Regards that the SQL database is used by analytical provenance in the dashboard a detailed description of the entity relationship model in Figure 5 can be read in D4.8.

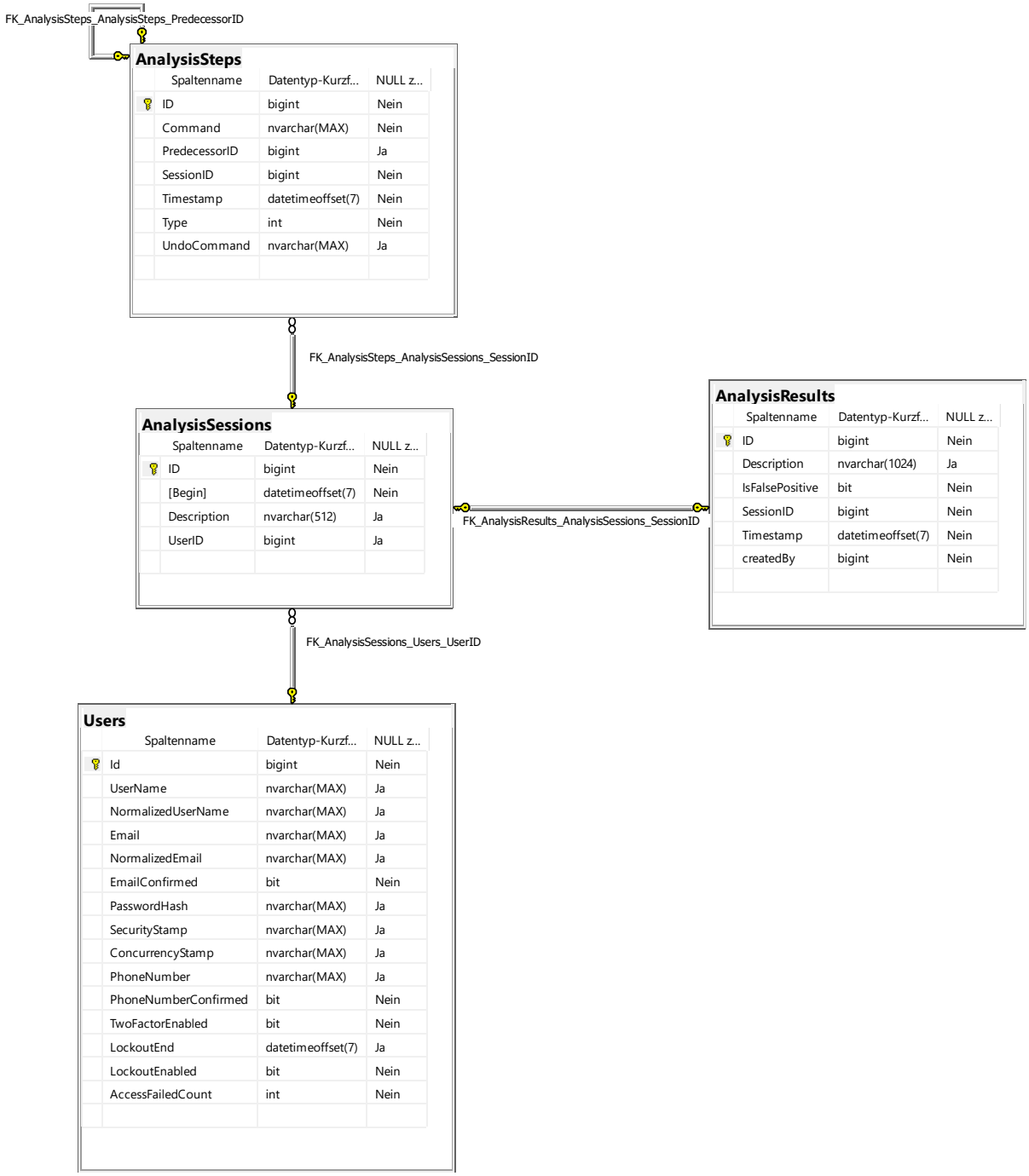

Figure 5: Entity relationship model of provenance database

5.2.6 Elastic Search library

As NEST (.NET client for Elastic Search)[5] already provides a strongly-typed interface for the Web API of ES, there is no need to implement similar functionality in SAPPAN. However, each organisation has structured their data in a different way, therefore we require a way to identify a minimum set of common properties in order to build visualisations from it. To this end, this library maps different JSON-base data models to the one used by the dashboard. This is achieved via JSON Path expressions that each installation of the dashboard must provide in order to find the minimal set of information like unique identifiers, timestamps, etc.

We currently provide two of these minimal sets, one for events generated by a host and one for NetFlows. However, the approach we use allows quickly defining new data sources in the dashboard and having them automatically interpreted. Listing 1 shows an exemplary configuration used for the two kinds of data sources at USTUTT. For each data source, multiple actual sources in form of ES indices can be specified. Furthermore, different subtypes of data can be specified by specifying certain fields that must be in a document in order to be imported to the dashboard. For instance, an event source in Listing 1 must have a field “event_type” which specifies the subtype of the event. As the filter is empty, any subtype is imported. In case of the flows, only documents having a field “@type” with the value “ipfix.entry” are processed by the dashboard.

Likewise, the actual mapping of the minimal set of information is flexible in that it provides support for fallback properties. As shown in Listing 1, properties do not have a single JSON Path to look up for their information, but an array, which allows for specifying alternate paths if the first one did not exist. This way, we prepared the dashboard for flexibly handling unexpected data.

[5] Elasticsearch .NET client: *https://www.elastic.co/guide/en/elasticsearch/client/net-api/current/introduction.html [checked on 29.01.2021]*

```
"ElasticSearch": {
    "ScrollTime": "8s",
    "Uris": [ "https://xxx:9200" ],
    "EventSources": [
        {
            "Indices": [ "ustuttevents" ],
            "TypeFilters": {
                "event_type": [ "" ]
            },
            "EntityIdentityPath": [ "$.subject.host.id" ],
            "EntityNamePath": [ "$.subject.host.fqdn" ],
            "EntityTypePath": [ "$.subject.type" ],
            "EventIdentityPath": [ "$.event_id" ],
            "TimestampPath": [ "$.time.created" ]
        }
    ],
    "FlowSources": [
        {
            "Indices": [ "ustuttflows" ],
            "TypeFilters": {
                "@type": [ "ipfix.entry" ]
            },
            "BeginTimestampPath": [ "$.['ipfix.flowStartMilliseconds']" ],
            "BytesPath": [ "$.['ipfix.octetDeltaCount']" ],
            "EndTimestampPath": [ "$.['ipfix.flowEndMilliseconds']" ],
            "DestinationAddressPath": [ "$.['ipfix.destinationIPv4Address']" ],
            "DestinationPortPath": [ "$.['ipfix.destinationTransportPort']" ],
            "PacketsPath": [ "$.['ipfix.packetDeltaCount']" ],
            "ProtocolPath": [ "$.['ipfix.protocolIdentifier']" ],
            "SourceAddressPath": [ "$.['ipfix.sourceIPv4Address']" ],
            "SourcePortPath": [ "$.['ipfix.sourceTransportPort']" ],
            "TimestampConverter": "Sappan.Dashboard.ElasticSearch.Mapping.UnixTimestampConverter"
        }
    ]
}
```

Listing 1: Exemplary configuration of the Elastic Search data mapper

5.2.7 MISP library

The MISP library provides a strongly-typed .NET Core implementation of MISP's core schema. The types are fully generated from the JSON Schema of MISP 2.4 and annotated such that their JSON serialisation meets the needs of MISP's Web API.

5.2.8 Dashboard application

The SAPPAN dashboard application is developed for experimental information visualisation. To this end, its layout belongs to a special category of user-interface design. This assumes that the dashboard displays content clearly and relies on a simple design. By using the front-end framework vue.js we built a front-end layout which can be dynamically filled by content from the data sources of the dashboard as shown in the

architecture diagram in Figure 1. To display the content a responsive layout grid is chosen. The layout provides the frame for interactive visualisations in which different perspectives on the raw data sensors can be dynamically visualized. A user should be able to use all visualisations relevant for an analysis without being distracted from other elements during the analysis. For that reason, the visual components are displayed in cards, to define the area of a visualisation. As a result, all available visualisations will be placed in cards. With such a layout, new visualisation can be integrated in the scope of a card in the dashboard without changing the layout itself.

A dashboard layout with much clutter, noise and eye catchers is overwhelming the users. With the use of cards and a front-end CSS library called Bootstrap a uniform presentation of the content area is kept simple and a plain. The layout consists of three area as shown in Figure 6.

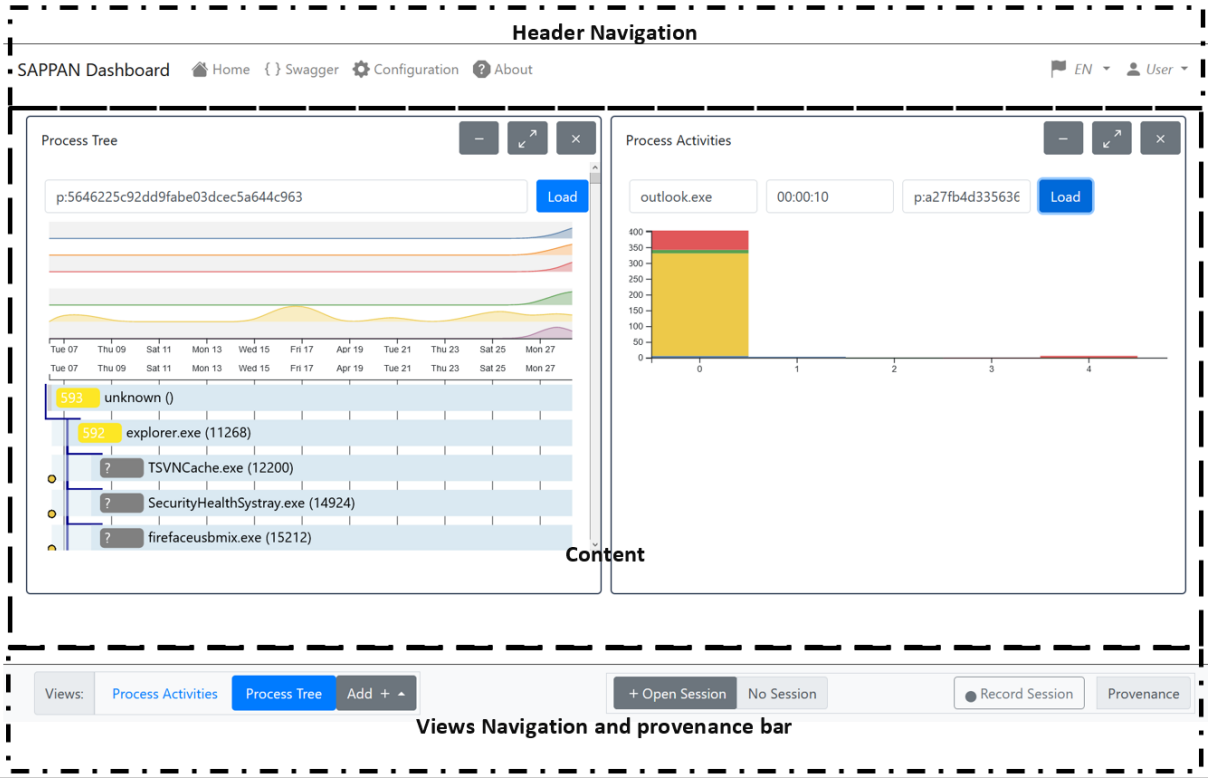

Figure 6: Parts of the SAPPAN Dashboard layout

Header: The left of the top area is for the logo and a navigation which is currently filled with development tools. The right part is for language settings and user options.

Content: The content a SOC agent works with, is all shown in the scrollable content area. This area is split in a grid with 12 even columns and 12 even rows. For example, in Figure 6 there are two cards with the size of 6 columns and 5 rows shown. The grid is managed in the class *DashboardContent* by including three components named *dashboard*, *dash-layout* and *dash-item* from *vue-responsive-dash*. The Dashboard component is responsible for deciding what responsive breakpoint is to be used. From this the layouts can then change to best suit the screen size[6]. The *dashboard* component is a flexible overall wrapper and has a unique id. It keeps the other elements

[6] Vue Responsive Dash Framework https://vue-responsive-dash.netlify.app/api/#dashboard [*checked on 29.01.2021]*

updated about the *currentBreakpoint* of the layout. The *dashboard-layout* component takes a computed layout object which can be manipulated by dragging, resize or hover over cards. The layout object is shown in Code example 1.

```
layout: {
                    breakpoint: "lg",
                    breakpointWidth: 1200,
                    numberOfCols: 12,
                    items: [{
                                        id: '0', x: 0, y: 0, width: 6, height: 5, minimized: false,
                                        cardContent: "ProcessTree", title: "Process Tree"
                                }, {
                                        id: '1', x: 6, y: 0, width: 6, height: 5,
                                        minimized: false, cardContent: "ProcessActivities",
                                        title: "Process Activities"
                                }],
                    activeItems: [{
                                        id: '0', x: 0, y: 0, width: 6, height: 5, minimized: false,
                                        cardContent: "ProcessTree", title: "Process Tree"
                                }, {
                                        id: '1', x: 6, y: 0, width: 6, height: 5,
                                        minimized: false, cardContent: "ProcessActivities",
                                        title: "Process Activities"
                                }],
                }
```

Code example 1: Layout object for Figure 6 view in JSON format

For each breakpoint a layout object can be created and managed by the number of pixels. In this example 'lg' indicates a large screen size. The number of columns is defined by 12 columns what makes it possible to show the 2 different visualisations in Figure 6 with a column size of 6. This number of columns can be reduced if less cards should be displayed. If the number of columns would be changed to 6, the second visualisation would rearrange then below, instead of next to the first visualisation. Each visualisation is linked in an item object from the component *dash-item*. The item attribute named *cardContent* is linked by name to a specific visualisation component and displays the content as an item via slot injection. If this card currently displayed on the dashboard, the object is present in the array *activeItems*. The differences between the attribute's *items* and *activeItems* indicate how much views are minimized. The attributes x, y, width and height are used to place the cards on the dashboard and rescale it.

For each visualisation a default width and height can be set in the class *Dashboard-Content* by change the attribute *prefComponentSize*, which will be load when a card is added to the dashboard. The attribute enables to customize the card size of a new opened visualisation to avoid that user has to change it first.

A *dash-item* has two slots which are dynamically filled if the user opens a card. The first slot is called card-title and is shown with the blue mark in Figure 7. In it the title of a visualisation like the *ProcessTree* is set. On the upper right buttons to interact with the card are placed. The blank area with the text fields and button 'Load' in Figure 7 is the second slot named *card-body*. It is filled with an actual visualisation component.

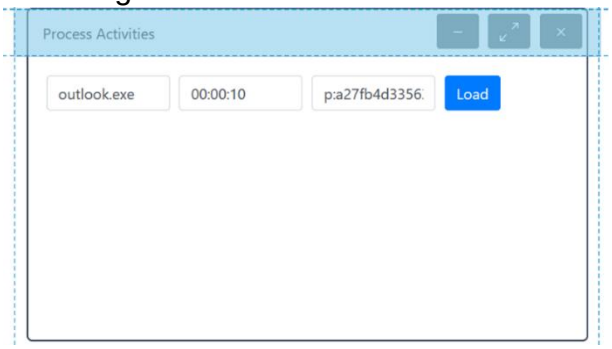

Figure 7: A dashboard card with header marked

The component *dash-item* emits different events by interacting with it. In Table 1 the events of the component are shown. These events can be used to trigger a specific action like it is used by the analytical provenance functions to save a layout change.

Name	Description
moveStart	Fires initially when an item is being moved (dragged) by human interaction
moving	Fires while an item is being moved (dragged)
moveEnd	Fires when the move is complete
resizeStart	Fires initially when an item size is changing (via human interaction)
resizing	Fires while the item is being resized
resizeEnd	Fires once resizing is complete
hoverStart	Fires when mouse begins to hover over DashItem
hoverEnd	Fires when mouse moves our of DashItem

Table 1: Events emitted by a dash-item component [10]

Views and provenance bar: At the bottom of the page is a fixed toolbar separated from the dynamic content area by a frame. On the left side, the 'views' area allows the user to open, close and minimise visualisations. By automatically updating the bar, it provides an anchor point on which cards are currently open and which are not. The 'views' area is visually separated from the 'provenance' area. On the right side is the provenance bar. With its use, SOC agents can record their activities through the use of analytical provenance and save them as an analysis session. The functions of the provenance bar will be implemented and developed within the project as part of Task 4.5.

5.3 Exemplary visualisation components

5.3.1 Process tree visualisation

The process tree in Figure 8 shows a filled line graph in the upper area of the visualisation that represents the frequency of the event types that occur during the process duration. If the user moves the mouse over the lines, the number of events around the mouse pointer is displayed.

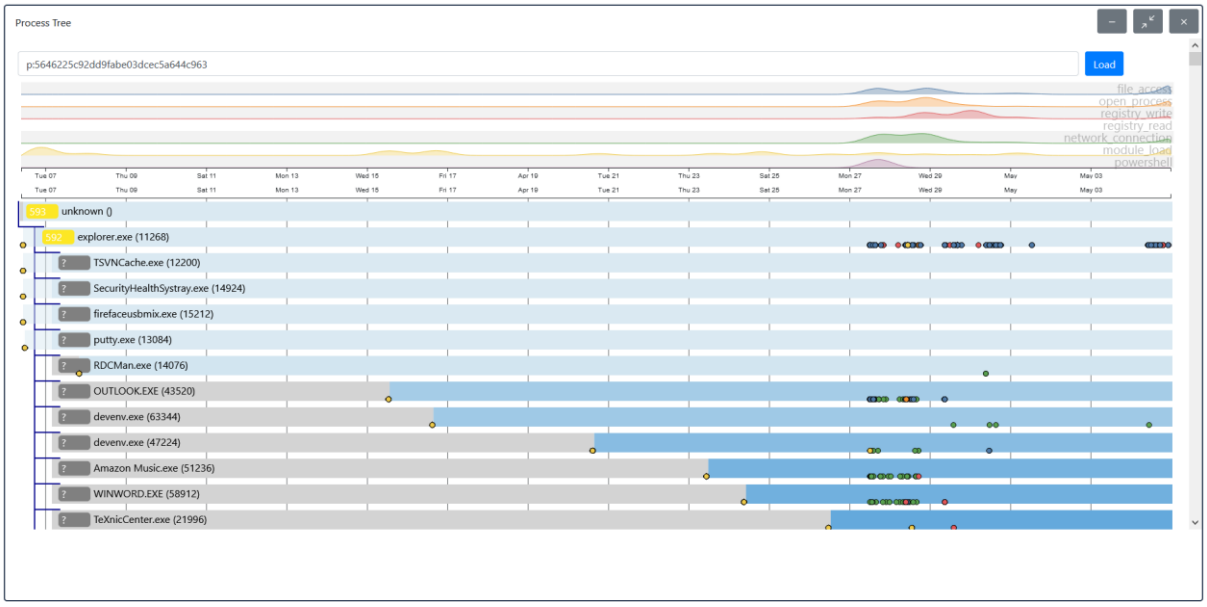

Figure 8: Process Tree Visualisation in the SAPPAN dashboard

The process to be visualised is then displayed with its hierarchy of sub-processes in an indented tree plot. The child processes are listed one below the other. When a child process started the saturation of the process changes. The part of the bar that is greyed out indicates that the process was not yet active at that time. The width of the bar indicates the lifetime of the child process. The user can interact with this representation by moving the mouse pointer over the events of a process. The number of children a node has is indicated by the rounded node at the start of its bar, which displays this number as text and also encodes this number via its colour. In case that the child processes are not loaded and their number therefore unknown, the node is coloured grey and contains a question mark. If the user wishes, she can call up the child processes for each process, which are reloaded when clicking on the node of the process. This sends the back-end controller a request, the children are added to the existing tree and the node is updated to show the correct number of children. By clicking on the process name, the SOC agent can view the process details in an overlay. For each process or sub-process, its associated events are drawn as circles on the bar, their position indicating the time they were recorded and the colour indicating the type of event. Hovering over any of these events prompts the display of a tooltip containing the raw data of the event such as event ID, event type and its timestamp and raises the associated SVG element to be drawn last, i.e. on top of all other event circles. When an event is op type 'open_process' and the target process is present in the tree, its node name is highlighted using bold red letters. In addition, the user may click on the event circle, which opens a dialog that shows the details of the target process and a load button where the user can load the process tree associated with the target process into the existing tree.

Given the data about process life times are provided by individual events of processes starting and stopping (and additional activities), significant computational work is required to reconstruct the hierarchy of processes starting other processes and the temporal order of these events. This work involves sifting through all relevant types of events (process starts and process stops) and correlating them by their globally unique ID that is assigned when the event is first recorded. As this would put unnecessary burden on the client and the network between front end and back end, the back end performs all of these tasks and returns a hierarchical representation in JSON format that can be directly fed into D3.js.

The reconstruction starts with the unique ID of the process of interest. For this process, the event representing its creation is obtained. This event already contains the so-called process chain, which is the list of its (recursive) parents, i.e. knowing the event of creation allows for retrieving the whole branch up to the root note (typically the system process). If requested by the front end, the siblings of each of the recursive parent processes are also included by searching for creation events with the same parent. Children of the initial process and of the siblings are only retrieved on-demand, i.e. as the user tries to open a subtree. From a back-end point of view, the process is exactly the same as described here as the newly retrieved branch is merged in the front end.

If requested, the back end retrieves all other events like registry and file system accesses for all of the nodes of the process tree. This operation is fairly straightforward as it only requires retrieving all documents related to the unique ID of each process. However, sequentially retrieving all events from ES is proved to be prohibitively slow, therefore this operation is performed in parallel batches using the async/await pattern, and the results of these queries are merged in memory.

Special handling is performed for open process events, i.e. events indicating that one process has opened another one via its process handle. For these events, the front end needs to know the target process that was opened in order to represent this operation visually. For all other events, we only retrieve their type, when it occurred and their unique ID to retrieve all the details of the event on user request.

5.3.2 NetFlow visualisation

In the frontend, the NetFlow visualisation in Figure 9 initially displays two filter options, which the user can use to decide in which time period and whether bytes or packets are to be selected as the data basis for the visualisation. If the filter entry is confirmed, a request is sent to the API controller of the backend. If the backend delivers the requested data, a visualisation with the d3.js library is generated.
As an example, the NetFlow visualisation shows in Figure 9 the bytes transferred within 9 days. The amounts of bytes are visualized in bars. The number of bars depends to the chose time period. Over the bar chart a Trend line, Season line and Remainder is drawn with the Seasonal Trend Decomposition by Loess. It isolates the long-term trend of the traffic volume, the weekly pattern and unusual behaviour.

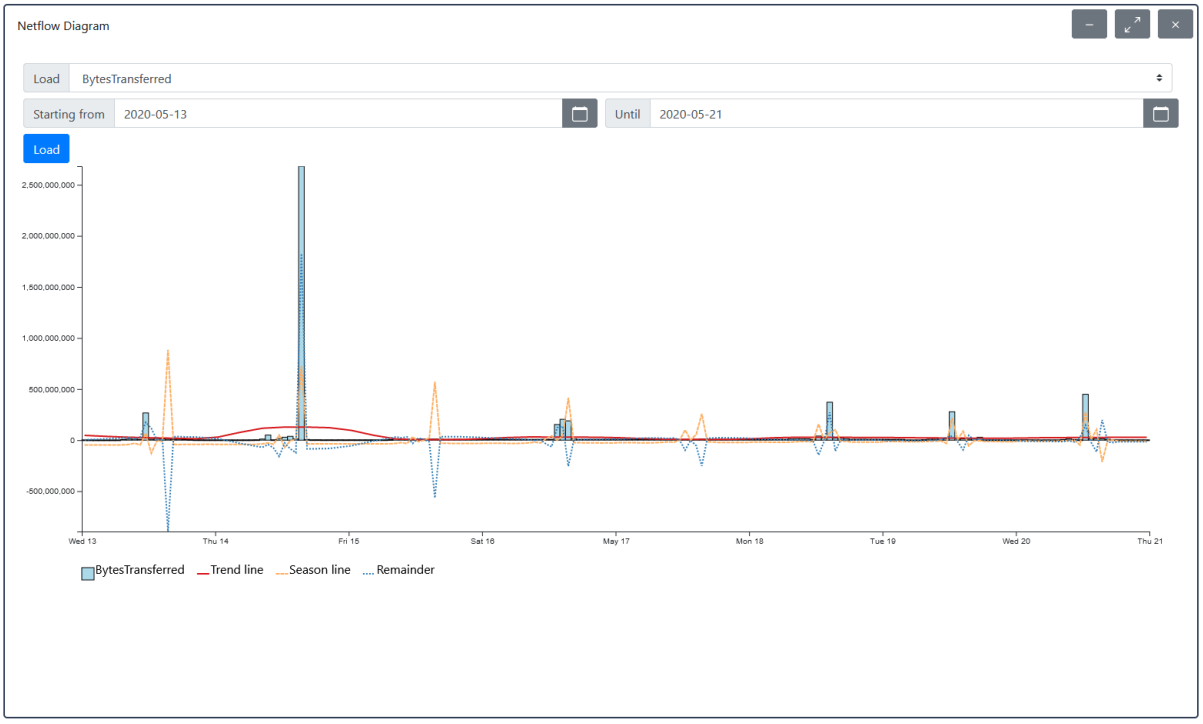

Figure 9: NetFlow Visualisation in the SAPPAN dashboard

On the back-end side, a dedicated API controller provides the necessary data for the time-series visualisation. As flows are stored as distinct documents in ElasticSearch, the back end has to perform a binning of the requested quantity (normally the number of bytes transferred) into a user-defined bucket size (one hour unless specified differently). In this step, the back end also adds empty buckets to remove the necessity to handle special cases in the front end.

The STL decomposition is performed by the NetFlow API controller. Using the separate library for computing it (cf. Section 5.2.3), the implementation in the dashboard itself is trivial. As for the raw data, the front end can request different quantities to be smoothed as well as change the bucket size for computing the initial histogram and the STL parameters of period length and seasonal length.

6 Future work

T6.1 is an ongoing task as new elements from other work packages continuously need to be integrated. In addition to that, the notification service envisaged in the SAPPAN architecture will be added to the dashboard, which will enable push notifications to be sent to the users of the dashboard. We plan to use *SignalR* as basis for this feature. For the aforementioned additional visual components, a visual representation of playbooks and analysis sessions is planned. Tracking mechanisms need to be added to the visualisations to meet the requirements for recording an analysis session.

In order to evaluate the functionality of the dashboard and the individual visualisations, usability tests will be carried out.

With respect to the existing demonstrator visualisations, we are currently pursuing the idea of computing an average event profile for each process image – such as 'outlook.exe' – and show this profile in the swim lane of all processes using the same image. This way, users would be able to quickly see whether a specific process behaves similarly to the average instance of its image, with respect to its events. We have already integrated an algorithm to obtain these average event profiles. As a prototype, we added a stacked bar chart visualisation of the events per bin, with configurable bin width as show in Figure 10. A reference process is used to determine the maximum timespan to consider for the profile calculation.

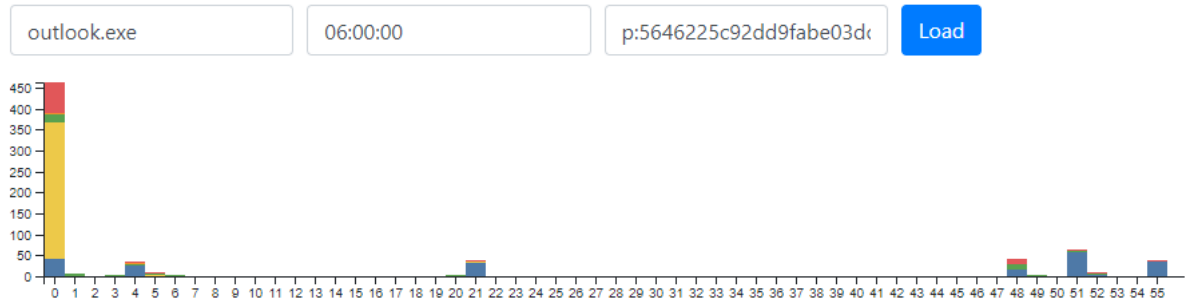

Figure 10: Process activities for the image 'outlook.exe' visualised as a stacked bar chart with a bin size of six hours

However, this solution has still a few drawbacks we want to address: first, determining the profile from the ElasticSearch database is computationally expensive, which makes it unfeasible to show that for all processes in tree. We therefore intend to compute it only on demand for processes the user interacts with. Second, the space in the swim lanes of the process tree is limited which warrants a different visual representation than a bar chart. We are still experimenting which representation would be suitable to convey all of the important information in the limited space available.

7 Summary

In the first version of the SAPPAN dashboard, we laid the foundations for integrating visualisation results from different work packages into a single demonstrator. The dashboard supports a flexible layout architecture that allows analysts to combine coordinated views of local data, and we demonstrate it on two examples, the NetFlow graph and the process tree in this iteration. The dashboard includes the functionality

to track analytical provenance as detailed in D4.8, which is used to track interaction with the dashboard at a high level with the integration of the per-visualisation-component level to follow. Foundational work to interact with the SAPPAN sharing platform based on MISP has also been completed. Integrating the data from the platform in the dashboard in a meaningful way is part of the remaining work in T6.1. The same holds for automating the deployment of the dashboard in the context of the demonstrator developed in T6.2.

References

[1] M. Shadan, Enterprise Dashboards: Design and Best Practices for IT, Hoboken, NJ: John Wiley & Sons, 2005.

[2] S. Few, „Dashboard confusion revisited," *Perceptual Edge,* 2007.

[3] J. Thomas und K. Cook, Illuminating the Path: The Research and Development Agenda for Visual Analytics, IEEE Computer Society, 2005.

[4] J. Biehl, M. Czerwinski, G. Smith und G. Robertson, „FASTDash: A Visual Dashboard for Fostering Awareness," *CHI 2007 Conference on Human Factors in Computing Systems,* p. 1313–1322, 2007.

[5] C. Gröger, M. Hillmann, F. Hahn, B. Mitschang und E. Westkämper, „The Operational Process Dashboard for Manufacturing," in *46th CIRP Conference on Manufacturing Systems*, 205–210, 2013.

[6] B. Schwendimann, M. J. Rodríguez-Triana, A. Vozniuk, L. Prieto, M. S. Boroujeni, A. Holzer, D. Gillet und P. Dillenbourg, „Perceiving Learning at a Glance: A Systematic Literature Review of Learning Dashboard Research," *IEEE Transactions on Learning Technologies,* Bd. 10, Nr. 1, p. 30–41, 2017.

[7] S. McKenna, D. Staheli, C. Fulcher und M. Meyer, „BubbleNet: A Cyber Security Dashboard for Visualizing Patterns," *Computer Graphics Forum,* Bd. 35, Nr. 3, p. 281–290, 2016.

[8] J. Xu, M. Amar und S. Moon, „On the Design and Performance of Prefix-Preserving IP Traffic Trace Anonymization," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet measurement*, 2001.

[9] R. Cleveland, W. Cleveland, J. McRae und I. Terpenning, „STL: A Seadonal-Trend Decomposition Procedure Based on Loess," *Journal of Official Statistics,* Bd. 6, Nr. 1, pp. 3-33, 1990.

[10] B. Sladden, „Vue responsive dash," 08 2020. [Online]. Available: https://vue-responsive-dash.netlify.app/api/#dashboard.