DTU Civil Engineering
Department of Applied Mathematics and Computer Science

Threat Identification & Response (TIR)

Methods for exhaustive identification of attack vectors represented graphically

Master Thesis

Methods for exhaustive identification of attack vectors represented graphically
Methods for exhaustive identification of attack vectors
represented graphically

Master Thesis
February, 2023

By
Jeff Gyldenbrand

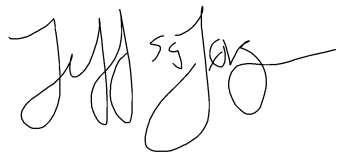
Cover photo: Vibeke Hempler, 2012
Published by: DTU, Department of Applied Mathematics and Computer Science, Richard Petersens Plads, Building 322, 2800 Kgs. Lyngby Denmark
www.compute.dtu.dk
ISSN: [0000-0000] (electronic version)
ISBN: [000-00-0000-000-0] (electronic version)

ISSN: [0000-0000] (printed version)
ISBN: [000-00-0000-000-0] (printed version)

Approval

This thesis has been prepared over five months at the department of Applied Mathematics and Computer Science, at the Technical University of Denmark, DTU, in partial fulfilment for the degree Master of Science in Engineering, MSc. Eng.

It is assumed that the reader has a basic knowledge in the areas of computer science and cyber-security.



Signature

12-02-2023

Date

Abstract

The goal of this thesis was to research the field of threat modeling within IT security and determine whether there exists a need for a new approach of formulating a system and if its possible to target people that are not specialist in the it-security field. The solution should be a graphical tool that allows for an easy and intuitive way of building a threat model that with a click of a button calculates all present threats of a system to the user.

This thesis uncovers the research areas such as threat modeling tools, open source vulnerability databases and graphical tools for developers by explaining the state of the art on each of these areas.

This thesis also provides the technical solution which is thoroughly explained. The initial steps for a complete solution is realized and we can conclude that there is a basis for further research and development.

Acknowledgements

Christian Nicolai Nielsen, MSc Civil Engineering, DTU
Creator of this thesis template.

Christian D. Jensen, Associate Professor, Department of Applied Mathematics and Computer Science.
Thanks for supervising this project, meetings and general guidance.

Altug Tosun, Ph.d.-student, Department of Applied Mathematics and Computer Science.
Thanks for co-supervising this project, meetings and general guidance.

Sofie Casparsen, Designer and cand.des, Designskolen Kolding
Thanks for participating in the survey and testing of this project.

Nicolaj Villadsen, BA-student, Economics, University of Copenhagen
Thanks for participating in the survey and testing of this project.

Lasse Villadsen, Financial Controller,
Thanks for participating in the survey and testing of this project.

Adam Villadsen, Theology student,
Thanks for participating in the survey and testing of this project.

Julian Olsen, Network Technician, Elproff.nu
Thanks for participating in the survey and testing of this project.

Glossary

0.0.1 Definitions, abbreviations, symbols

STRIDE	**S**poofing, **T**ampering, **R**epudiation,**I**nformation Disclosure, **D**enial of Service, **E**levation of Privilege
OWASP	Open Web Application Security Project
HTTPS	Hypertext Transfer Protocol Secure
TLS	Transport Layer Security
SSL	Secure Sockets Layer
ARPANET	Advanced Research Project Agency Network
CVE	Common Vulnerabilities and Exposures
CWE	Common Weakness Enumeration
CVSS	Common Vulnerability Scoring System
CISA	...
DHS	...
NVD	National Vulnerability Database
NIST	National Institute of Standards and Technology
MITRE	American not-for-profit organization
FIRST	Forum of Incident Response and Security Teams
DOM	Document Object Model
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
IoT	Internet of Things
SQL	Sequel query for databases
CPU	Central Processing Unit
PC	Personal Computer
CIA	Confidentiality, Integrity, Availability
AWT	Abstract Window Toolkit
JFC	Java Foundation Classes
JVM	Java Virtual Machine
WWW	World Wide Web
USB	Universal Serial Bus
HDMI	High Definition Multimedia Interface

Contents

1 Introduction

The rapid development of technology since the beginning of the Internet has meant that systems, despite better and stronger technology, have become far more complex. Systems must often be able to talk to users across technologies and often with several thousand simultaneous users. Other times, it can be completely ordinary everyday things that are now connected to the internet, for example a smart TV or fridge, known as the internet of things. Because all these things become more complex, it also means that the risk of errors and vulnerabilities increases, and where there are vulnerabilities there are always people with bad intentions. It is no secret that systems, individuals, private as well as companies, even governments are hacked or in the risk of it.

In this thesis, we will explain the social problem we experience in today's world when it comes to IT systems and vulnerabilities. This will be explained in section 2 as well as an explanation of the precise problem we are trying to solve. After this, we will present our research on the state of the art within vulnerability databases, graphical user interfaces and threat modeling frameworks in section 3. This will lead up to the idea behind this project in section 4, including what thoughts and choices, and opt-outs, have been made along the way , and how the various discoveries in Section 4 have had their influence. In this section, we also touch on various assumptions we have made in order to realize the project. Afterwards, in section 5, we will describe the technical solution, by first showing and explaining an overall architecture of the system, and then explaining the technical solution for the frontend and backend part respectively. Section 6 will be a section that shows proof of work in the form of users who have been allowed to test the solution and completed a survey where they have provided feedback. Section 7 will describe our thoughts on future work that can be done on the system. We will end with a conclusion in section 8. There is also a list of all references and an appendix.

2 The problem

2.0.1 The context of IT-Security

The digital era of the 90s was a time when IT security was not a concept known or recognized concept, particular among private individuals, but also to a large extend among companies. This is in great contrast to the enormous focus on IT security that we see today. Despite this, the threat of computer viruses, worms and hackers had already been present for several decades. In fact, the first computer worm, called *The Creeper*, was invented in 1971 by Bob Thomas[1]. This was a security-test designed to explore the possibility of a self-replicating program on the ARPANET[2], a program created with no malicious intent in mind.

However, this paved the way for the development of future worms and viruses by individuals or even government organizations with less noble intentions. Ray Tomlinson, a colleaque of Bob Thomsen, expanded on the Creeper and created *The Reaper*, which spread and tracked down installments of the Creeper worm and deleted them. In a way, The Reaper can be considered the worlds first antivirus program[3].

Before the internet became a mainstream phenomenon with the introduction of the world wide web in the early 90s, hackers primarily targeted military and corporate organizations. However, as the internet became more accessible and widespread, the scope of the potential targets expanded to include private individuals and small businesses too.

2.0.2 Current challenges in IT-security

The IT security industry has become a huge industry. This is a natural reaction to technology becoming larger, more complex and more widespread throughout the world. In line with this development, the need for specialists in cyber security, and IT security in general, has also become much greater. Protection against cyber attacks and IT systems, consultancy, threat assessment of system vulnerabilities are among the areas the specialist companies work with. Because of the crucial role IT security plays in modern businesses and even for private individuals, this means that significant resources are invested to secure systems and digital information.

The need for protecting, mitigating and knowledge in regards to protecting systems is thus not only a concern for businesses, but also private individuals. However for many private individuals, the cost of hiring an IT security export or purchasing comprehensive security solutions are simply not affordable. In our opinion, this creates a gap within the IT security sector where a great portion of the population are not able to properly secure their systems or devices. Furthermore, even for those that are able to pay for such services, the complexity of these services and solution might be too hard to comprehend for people without deep technical knowledge on the subject.

2.0.3 Addressing the challenges

The IT security sector is facing a significant challenge in addressing the gap between ordinary people and the complex technical knowledge required to protect systems from vulnerabilities and threats.

To address this issue, our project aims to create a plug-and-play friendly tool that enables users to easily describe their system and understand its potential vulnerabilities without needing to pay for potential expensive solutions from security experts and security companies. Users get the opportunity to use an open source and free tool that neither requires

a deep technical understanding for IT security, vulnerabilities and threats to a system or needs to have his finger on the pulse with the latest vulnerabilities, hacks, software patches and the like by graphically building systems and adding information to these that are available from public sources.

In Section 3 we will provide an overview and examine many of the frameworks that exist currently, also graphical tool and vulnerability databases. Here we will provide some background information and information on how to use each of the frameworks and explain their core features. Ultimately, we will reflect on the differences between these approaches and the goal of our project in section 3.4. We will do this to understand where we hope to make the first steps towards closing this gap and improve the overall security for the whole sector.

3 State of the Art

The cybersecurity field is constantly growing, new vulnerabilities emerges daily, threats to hardware systems grows, naturally, a demand for cybersecurity is also growing. *Research*, *vulnerability databases*, *threat modelling frameworks & tools* for mitigating vulnerabilities and exposures to systems thus need to evolve too. This section will take a closer look to the state of the art in each of these areas.

First, we will examine some of the latest research, techniques and technology that are being used to detect and mitigate threats and the challenges we face in cybersecurity today.

Next, we will explore two of the most popular vulnerability databases and examine how these databases are being used to identify cybersecurity vulnerabilities, and how they can be integrated into security frameworks and tools.

Then, we explore some graphical user interface libraries that can be used in software development of a graphical threat modeling tool.

Finally, we will take a look into some of the existing threat modelling frameworks and tools such as OWASP Threat Dragon, OWASP pytm, CORAS, Threagile, SeaSponge and Microsoft TM Tool.

3.1 Vulnerability Databases

Vulnerability databases are collections of information on known vulnerabilities and exposures in both software and hardware. These database are widely used by it-security organizations, researchers and individuals for various reasons such as identifying vulnerabilities, tracking and discussing cyber security vulnerabilities. A few examples of their usage:

- Software development developers can use the API to get information on the newest known vulnerabilities in libraries and frameworks.
- Incident response: incident response teams can use the API to get information about known vulnerabilities that may be related to something they are investigating.
- Academia: researchers can use the API to get information on known vulnerabilities to identify new vulnerabilities in software they are analyzing, or understanding the impact of a vulnerability of even developing new security tools.

Some of the popular vulnerability databases are:

- Common Vulnerability and Exposures (CVE)[4]
- National Vulnerability Database (NVD)[5]
- Coordination Center Vulnerability Notes Database (CERT/CC)[6]

3.1.1 Common Vulnerabilities and Exposures (CVE)

The Common Vulnerabilities and Exposures database is a large database used for identifying vulnerabilities in software and hardware. A vulnerability is an issue or weakness that can result in a successful attack. Identifying such vulnerabilities is therefor important to mitigate possible exploits and malicious actors that takes advantages of vulnerabilities.

The CVE database was established on 1999 by Mitre[4]. The goal of this database is to provide a standardized and publicly available way of identifying cybersecurity vulnerabilities and exposures. Mitre is responsible for maintaining the CVE list and also assigns new CVE identifiers to vulnerabilities. The CVE database can be accessed through Mitre's website[1], a command-line tool or their API.

The architecture of a CVE: CVE has records, CVE-Record, which describes the data about a vulnerability associated with an *CVE-ID*. There exists an *CVE-Record* for each vulnerability in the CVE-database. The CVE-ID is the number portion of a CVE-Record, a unique identifer of the vulnerability, where *YYYY* represents the year the vulnerability was assigned the CVE-ID and *NNNN* represents the unique number within that year:

$$(CVE - YYYY - NNNN)$$

The lifecycle of a CVE Record(3.1) begins with the discovery of a new vulnerability which then is reported the a CVE Program participant (A partner of MITRE CVE Program). The participant requests a CVE identifier, which then is reserved until the CVE Program participant has enough information and details to submit it. These details includes: affected product(s), versions, vulnerability type, root cause or impact and a public reference. Once these details are provided it will be published.

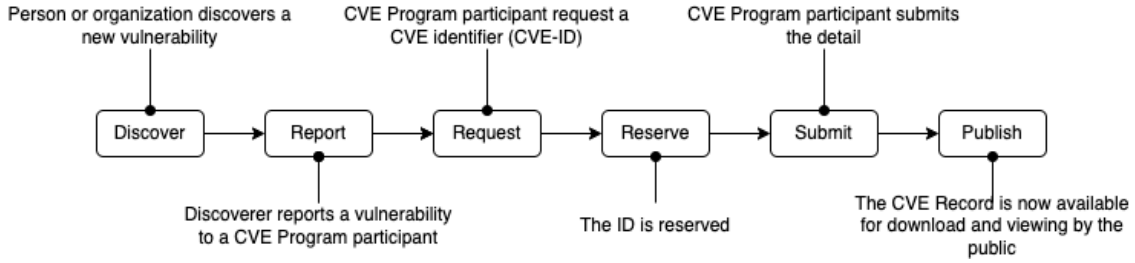


Figure 3.1: Lifecycle of a CVE-Record

3.1.2 National Vulnerability Database

The National Institute of Standards and Technology, NIST[7] is a agency of the U.S Department of Commerce[8] and responsible for developing the standards and guidelines in technology, including cybersecurity. NIST also has an API that allows users to access data from the NVD database on vulnerabilities and exposures. The API from NVD and MITRE are two separate API's using different datasets. The date derived from MITRE is primarily focused on providing a standardized and publicly available way of identifying and describing vulnerabilities where the data from NVD is derived from a variety of sources, including CVE data. In other words, NVD provides a more comprehensive data set and additional information. Information added by NVD includes: identifier, severity score(CVSS), weakness identifier(CWE), affected products, further information such as links to other ressources. The NVD API is used for this project, and is described next: The Common Vulnerability Scoring System (CVSS) is maintained and updated by the Forum of Incident Response and Security team (FIRST)[9] is used for measuring the severity of a vulnerability. CVSS provides a way to score and rank the vulnerabilities on a number of factors that affect the vulnerability's overall impact.

[1] https://cve.mitre.org/

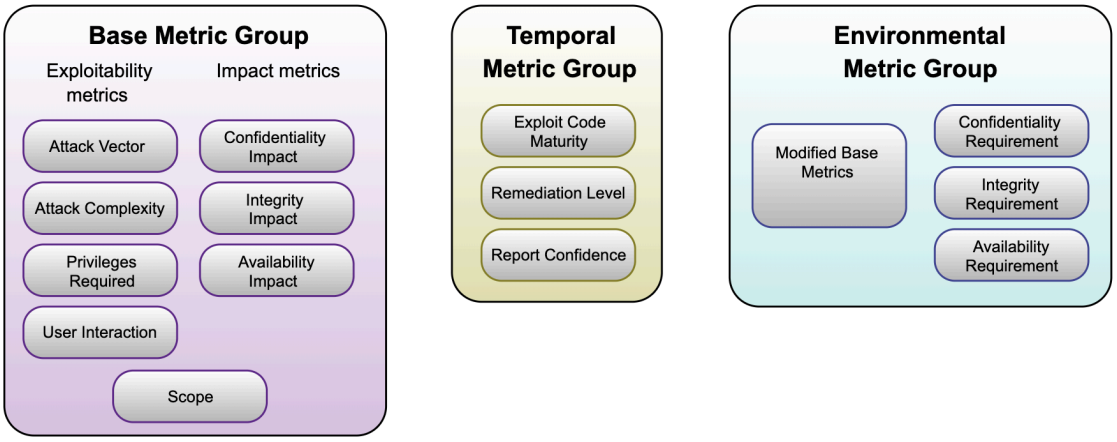


Figure 3.2: CVSS Metric Groups

Base metrics: the CVSS only considers static properties that do not change over time and do not depend on the environment in which the vulnerable software is running, it uses a set of base metrics to generate a base CVSS score. Colored boxes indidates the threat level, red is bad, yellow is mediare, green is good:

- **Attack Vector**: the means how an attacker can gain access to the vulnerability. The higher the metric value, the higher

  Network | Adjecent | Local | Physical

- **Attack Complexity**: the conditions outside of an attackers control which must exists for exploiting a vulnerability. If *none* conditions are required, its easier for the attacker

  NONE | LOW | HIGH

- **Privileges required**: the higher the privileges required to a system the harder it is for an attacker to exploit the vulnerability

  NONE | LOW | HIGH

- **User Interaction**: this metric describes if the users action is required in order for the attacker to exploit the vulnerability. E.g. a user must click some link and perhaps download malicious software before the attacker can participate.

  NONE | REQUIRED

- **Confidentiality**: this metric measures the level of disclosure of data and confidentiality.

  NONE | LOW | HIGH

- **Integrity Impact**: this metric measures if an attacker is able to modify or delete any data

  NONE | LOW | HIGH

- **Availability Impact**: this metric measures if an attacker is able to deny access to resources or reduce performance

  NONE | LOW | HIGH

- **Scope**: this metric is measured by the level a vulnerability can affect components beyond its own scope. E.g. a virtual machine, if the vulnerability can affect the host machine.

  UNCHANGED CHANGED

Temporal metrics builds on BASE and adds metrics that evolves over time:

- Exploit Code Maturity: this metric measures how likely it is that the vulnerability to be exploited by the current state of the exploit

  NOT DEFINED HIGH FUNCTIONAL PROOF-OF-CONCEPS

- Remediation Level: this metric is the level of availability of a solution

  UNAVAILABLE WORKAROUND TEMPORARY FIX OFFICIAL FIX

- Report Confidence: this metrics measures the credibility of the vulnerability details

  UNKNOWN REASONABLE CONFIRMED

Environmental metrics: is a set of parameters used to score the impact of a vulnerability on the target environment:

- **Modified Base Metrics**: this metrics allows for overriding base metrics taking into account any countermeasures or mitigation's already in place of the users environment

- **Confidentiality Requirements**: this metric measures the impact of a vulnerability that affects confidentiality. It measures to what degree the vulnerability may allow unauthorized access or disclosures of confidential information

  UNKNOWN REASONABLE CONFIRMED

- **Integrity Requirements**: this metric measures to what degree it may allow unauthorized modification of data or unauthorized access to data

  UNKNOWN REASONABLE CONFIRMED

- **Availability Requirements**: this metric measures to what degree it may cause a system or resourse to become unavailable or cause a system, to crash

  UNKNOWN REASONABLE CONFIRMED

Figure 3.3 illustrates how the metrics are assigned:

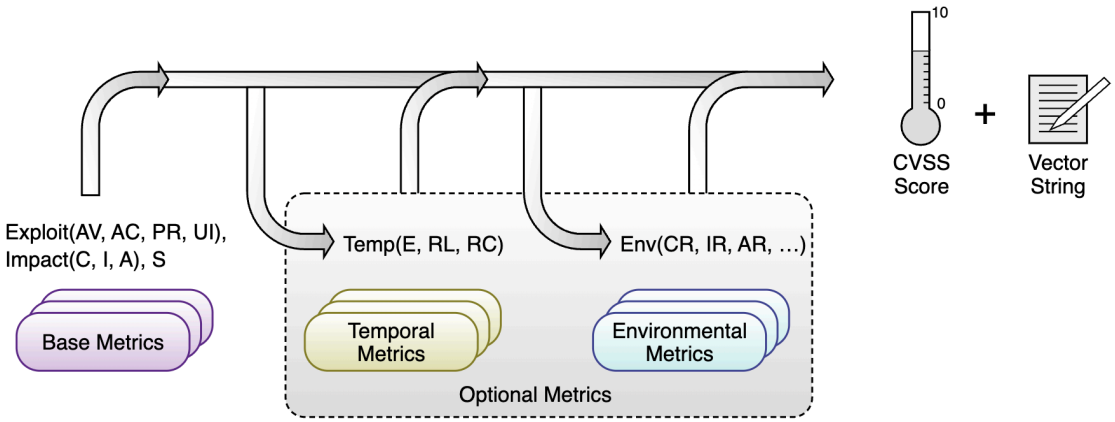


Figure 3.3: CVSS Scoring

3.2 Graphical User Interfaces

The software part of this project must contain a graphical user interface that will make it easy for users to build a model and form a threat picture of their system. Based on the results of the research in section 3.1 on vulnerabilities and modern systems, we have come to the conclusion that an integration of the CVE-API from the National Vulnerability Database aligns well with the goals of this project. Thus, it is important to take into account such an API when designing the graphical solution.

Initially thoughts were on developing a graphical desktop application for Java. Primarily because Java is a language I am well versed in, but also because it is a language that allows creating modular programs and reusable code[10], Java is platform independent and generally a robust language used by millions of developers.

With some degree of experience with graphical tools in regards to Java, it was a natural first choice to explore. Some of the well established of widely used tools includes: Java Swing and JavaFX[11]. These will be described next:

3.2.1 Java Swing

Background: Java Swing is based on technologies such as Java Foundation Classes (JFC) and Abstract Window Toolkit, AWT[12]. For better understanding of Java Swing, we will briefly describe these technologies. It all began with Java AWT developed by Sun Microsystems (later acquired and still maintained by Oracle Corporation) as part of the Java platform in 1995. This library provides a set of graphical user interface widgets that allows developers to build visual applications in Java. AWT is platform independent which means it can run on any operating system that supports the Java Virtuel Machine (JVM). AWT is utilizing the native GUI of the underlying operating system, this means that the application will have an appearance that matches the operating system it is run on. This also means that developers must be aware of the differences of the various operating systems when building their application. One example could be the *Close*-button, which is often found at the top of an application's window, can be located on the right, left or center depending on the platform, and in some cases it does not appear at all. There are, of course, quite a few differences between platforms, which is why it makes the development of cross-platform applications difficult for the developer. This lead to the development of the JFC in 1997 which dealt with these limitations. In addition, a wide range of new functionality was added, including the Swing toolkit.

Core Features: Java Swing provides GUI components such as buttons, labels, textfields,

checkboxes, radiobuttons, tables, menubars, toolbars and much more. Figure 3.4 is an example of a Swing application with some of these features. [13]. Java Swing mitigates the limitation in AWT descibed before with a *Pluggable Look-and-Feel*-support[13], which allows the developer to set a single appearance of the application across platform, e.g. a Windows-look. Swing also provides a *Java 2D API* which makes it possible to add 2D graphics, images and text into the application.

Unfortunately, Java Swing does not offer a natively visual builder of applications. The graphic elements must be described in code. However, there are plugins for many of the popular IDEs that offer a tool where you can drag-n-drop components and design the application visually. Some of these IDEs are: Eclipse, NetBeans, Intellij IDEA and Android Studio.

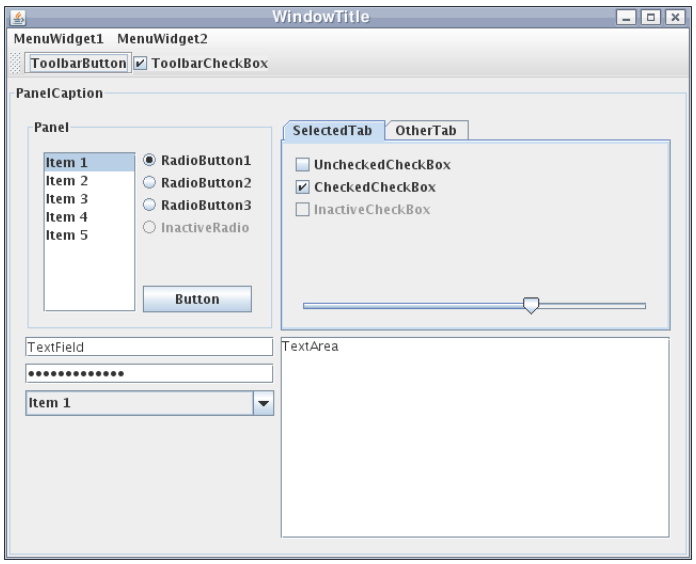


Figure 3.4: Java Swing - GUI Widgets

Get started with Swing: In order to use Swing in a Java application you will need to install the latest release of the Java SE (at least this is recommended due to improved performance, bug fixes and newest features)[14].

- Download: `https://www.oracle.com/java/technologies/downloads`
- Create a java application and import the Swing library: `import javax.swing.*`
- Compile and run the program.

Code listing 3.1 shows an example of a basic Swing application: we import the swing package at `line 1`, this gives us access to all of its classes, such as the label declared on `line 6` and the button declared on `line 10`. We can perform actions by adding an event listener to components, as seen on `line 11-15`. In this case the labels text is changed. On `line 19-22` we configure the settings for the application window. When the application is compiled and running a window will appear

```
1 import javax.swing.*;
2
3 public class SimpleSwingApplication extends JFrame {
4     public app() {
5         // Create a label
6         JLabel label = new JLabel("Hello, world!");
```

```
        add(label);

        // Create a button
        JButton button = new JButton("Click me!");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                label.setText("Button clicked!");
            }
        });
        add(button);

        // Set the frame properties
        setTitle("Simple Swing Application");
        setSize(300, 150);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String[] args) {
        new app();
    }
}
```

Listing 3.1: Simple example of a Swing Application

3.2.2 JavaFX

Background: Sun Microsystems introduces JavaFX in 2007[15], a completely new platform for developing graphical applications for both desktop and mobile. The latest version og JavaFX also supports development for embedded systems, and web applications with their Web component that renders HTML and JavaScript content inside Java applications. JavaFX is intended as a replacement for Java Swing and Sun Microsystems recommends that developers move to JavaFX. Despite this, JavaFX still supports the Swing library.

Core Features: Of course, JavaFX offers all the same elements as in AWT, JFC and Swing. In addition to that, JavaFX also offers Rich graphics and built-in support for both 2D and 3D, animations and other features used in Rich Internet Applications (RIA). Furthermore, JavaFX offers a visual scene builder tool, that enables developers to visually design their applications. An example of the scene builder is shown in figure 3.5

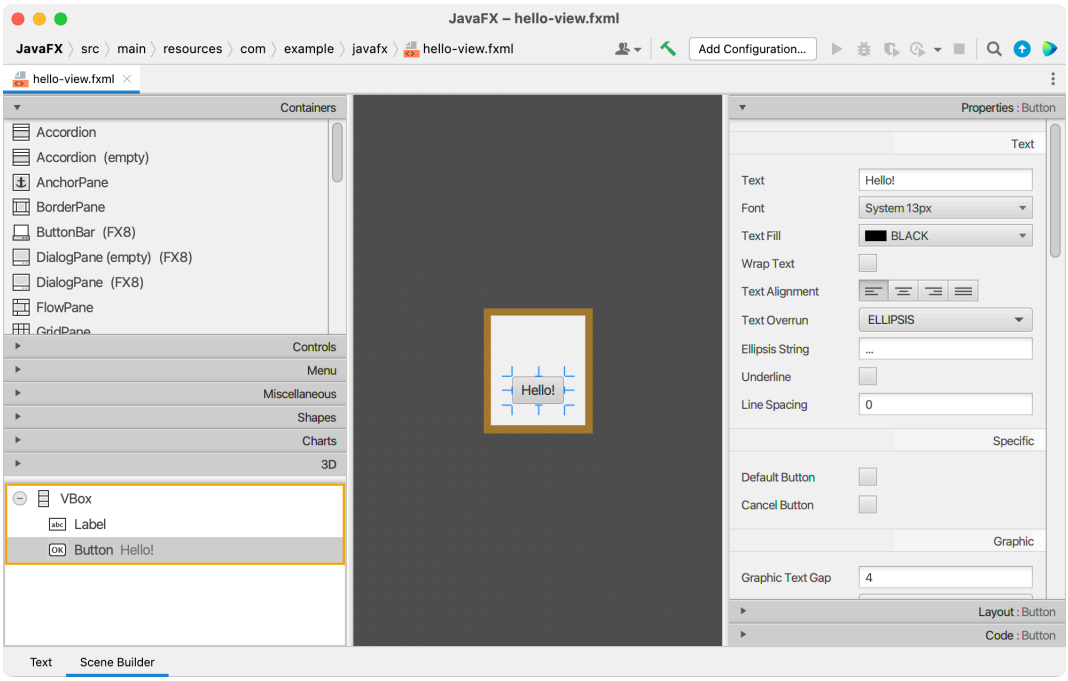


Figure 3.5: JavaFX - Scene Builder

Get started with JavaFX: Its basically the same procedure as in the Java Swing setup. The website for JavaFX offers great help on running the library via Maven, Gradle or in the different IDE's supported[16].

- Download and install JDK for your operating system: `http://jdk.java.net/19`
- Create a java application and import the JavaFX library: `import javafx.application.Application`
- Compile and run the program.

As illustrated in figure 3.6, a JavaFX application is built with a stage and a stage. Where stage is the actual window that is displayed and the stage is a contaner with all the content. The content of the scene is represented hierarchically as a graph of nodes.

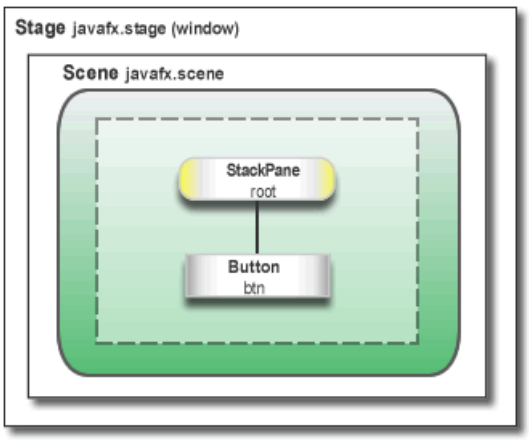


Figure 3.6: JavaFX - Scene Graph

Code listing 3.2 shows the example of a basic JavaFX application. The code is very

similar to the example from Java Swing. On `line 1` we import the JavaFX library itself, and from `line 2-7` the rest of the classes needed. Here we can set a title aswell. We do this on Stage, which is the application's window. `line 24` defines a StackPane as our root-node in the scene, thus we can add buttons and other components as child-nodes. Finally, we present the Stage window on line `line 27`

```
1  import javafx.application.Application;
2  import javafx.event.ActionEvent;
3  import javafx.event.EventHandler;
4  import javafx.scene.Scene;
5  import javafx.scene.control.Button;
6  import javafx.scene.layout.StackPane;
7  import javafx.stage.Stage;
8
9  public class HelloWorld extends Application {
10     public static void main(String[] args) {
11         launch(args);
12     }
13     @Override
14     public void start(Stage primaryStage) {
15         primaryStage.setTitle("Hello World!");
16         Button btn = new Button();
17         btn.setText("Say 'Hello World'");
18         btn.setOnAction(new EventHandler<ActionEvent>() {
19             @Override
20             public void handle(ActionEvent event) {
21                 System.out.println("Hello World!");
22             }
23         });
24         StackPane root = new StackPane();
25         root.getChildren().add(btn);
26         primaryStage.setScene(new Scene(root, 300, 250));
27         primaryStage.show();
28     }
29 }
```

Listing 3.2: Simple example of a JavaFX Application

3.2.3 GoJS

Background: GoJS is a JavaScript library developed by Northwoods Software Corporation[17] released in 2012. The GoJS library is a graphical tool that easily can be implemented in web applications. With GoJS it's possible to create graphical models and interactive diagrams with relational components. Unfortunately, it is not a tool that is completely free to use. It is allowed to use the tool for non-commercial and internal use. A license must be purchased to be able to use the tool freely.

Core Features: GoJS has built-in diagram elements like nodes, linking between nodes, grouping of nodes. The appearance and behavior of the graph, nodes and the links between them can be easily customized with, for example, changing colors, sizes, text, behavior on mouse hover, entry, exit just to name a few. With GoJS you can create your own elements and templates in the form of component forms, layouts and interactions. GoJS supports data binding, this means you can create interactive diagrams that automatically update when there is a change in data or by user input.

GoJS is compatible with all browsers that support the JavaScript framework. In addition, it supports touch and mobile devices.

Figure 3.7 is a simple example taken from the official website for GoJS[18]. This example illustrates a simplistic and pretty diagram containing four nodes that is linked together

by arrows. These nodes are fully interactive, a user can freely move them around in this example. There is, of course, great freedom in programming the behavior of the components. Their website has a really good documentation for their library as well as a huge number of examples to help in the development of an application[19].

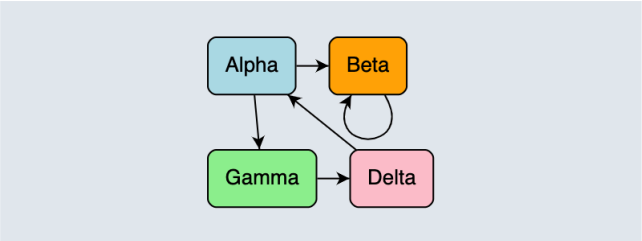


Figure 3.7: GoJS Diagram

Get started with GoJS: Basic knowledge of HTML and JavaScript is required to develop with GoJS. That said, it's pretty simple to set up: either download[20] the library directly and place and reference the library in the folder of your web project. The library can also be installed via the packet manager npm[21] for Node projects: `npm install gojs --save`

The code implementation of figure 3.7 is seen in code listing 3.3. The GoJS library is imported on `line 5` in the body-tag of the HTML-file. From here we can reference the library anywhere in the file. We can declare a diagram by referencing the go-object as seen on `line 11`. From `line 15-22` we are defining a template for the nodes of the graph. In this case we define the shapes of nodes to be rounded rectangles with a color data-binding that we can reference later. Furthermore we define the nodes to have a text-block with curtain properties. On `line 24-37` we create a list of nodes: Alpha, Beta, Gamma, Delta, and assign a color to each of them. Then we define the linking between the nodes.

```
1  .
2  .
3  .
4  <body>
5      <script src="https://unpkg.com/gojs@2.3.1/release/go.js"></script>
6  </body>
7  .
8  .
9  .
10
11 const myDiagram =
12   new go.Diagram("myDiagramDiv",
13     { "undoManager.isEnabled": true });
14
15 myDiagram.nodeTemplate =
16   new go.Node("Auto")
17     .add(new go.Shape("RoundedRectangle",
18       { strokeWidth: 0, fill: "white" })
19       .bind("fill", "color"))
20     .add(new go.TextBlock(
21       { margin: 8, font: "bold 14px sans-serif", stroke: '#333' })
22       .bind("text", "key"));
23
24 myDiagram.model = new go.GraphLinksModel(
25   [
26     { key: "Alpha", color: "lightblue" },
27     { key: "Beta", color: "orange" },
```

```
    { key: "Gamma", color: "lightgreen" },
    { key: "Delta", color: "pink" }
  ],
  [
    { from: "Alpha", to: "Beta" },
    { from: "Alpha", to: "Gamma" },
    { from: "Beta", to: "Beta" },
    { from: "Gamma", to: "Delta" },
    { from: "Delta", to: "Alpha" }
  ]);
```

Listing 3.3: Simple example of GoJS

3.2.4 X6

Background: X6 were created by Alibaba Group in 2016. AntV Group is Alibaba's data visualization team. They are responsible for developing and maintaining, among others, the X6 library. It is an open-source graph visualization libraries that allow developers to create interactive graphs and diagrams. Figure 3.8 is an example of one way to style and interlink nodes in the graph. There exists a large number of built-in examples that are free to use, such as: lane diagrams, flow chats, UML class diagram, expandable tree structures, DAG graphs, and the list goes on.

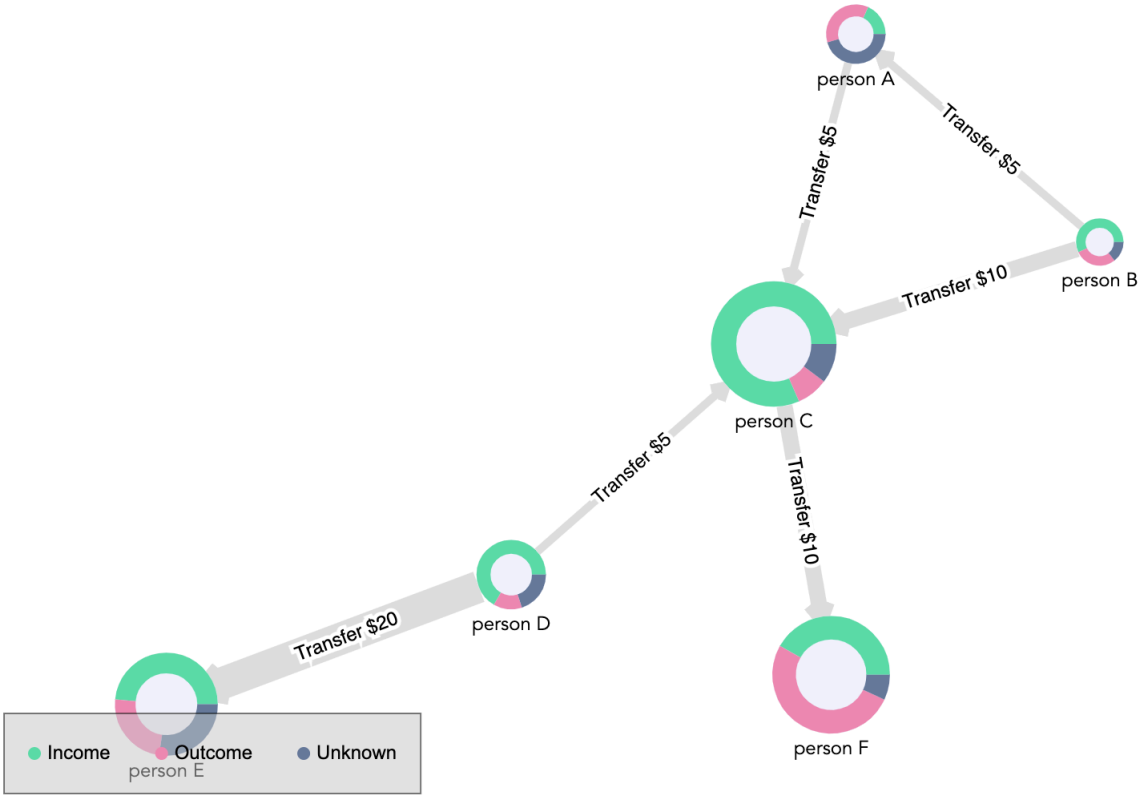


Figure 3.8: X6 Example of a Graph Diagram

Core Features: In X6, you can define a graph with nodes[22] that can interact with each other. A simple graph is defined is the code example below. There is of course great amount of attributes that can be set. The container of the graph is the reference point to the DOM element named *container* in the HTML file:

```
const graph = new Graph({
  container: document.getElementById("container"),
  connecting:{
    allowNode:true,// boolean
  },
});
```

There are predefined nodes such as circles, rectangles, ellipse and polygon, images to name a few. You can also customize a node as the X6 engine itself is based on HTML and Scalable Vector Graphics (SVG). The code example below shows how you can easily and quickly add a node with shape, size and position to a graph:

```
graph.addNode({
  shape: "rect",
  x: 100,
  y: 40,
  width: 100,
  height: 40,
});
```

As already explained, nodes can have connections to each other. This is done via edges[23]. An edge must at least have a source and target node defined. In addition, edges are highly customizable both in terms of shape, colors, text but also the route between the nodes, these can be normal, orthonal, manhatten, metro to name a few. The code example below shows the very simple way to define an edge:

```
graph.addEdge({
  shape: "edge",
  source: "node1",
  target: "node2",
});
```

X6 allows event listeners[24] on both the canvas, nodes and edges. There are a large number of mouse events here: click, double click, right click, mouse down, mouse enter (when the mouse is inside a node, for example), mouse out (when it leaves the node again). We can define such events directly on the graph object, as shown in the code example below. There are event listeners for far more than just mouse events, for example you can listen to if a node moves or is near a certain coordinate. You can listen to whether an edge join/unjoin or node deletion/add/modify occurs:

```
graph.on("cell:click", ({ e, x, y, cell, view }) => {});
graph.on("node:click", ({ e, x, y, node, view }) => {});
graph.on("edge:click", ({ e, x, y, edge, view }) => {});
graph.on("blank:click", ({ e, x, y }) => {});

graph.on("cell:mouseenter", ({ e, cell, view }) => {});
graph.on("node:mouseenter", ({ e, node, view }) => {});
graph.on("edge:mouseenter", ({ e, edge, view }) => {});
graph.on("graph:mouseenter", ({ e }) => {});
```

The graph model objects data can be exported to json-format[25]. The exported data consists of cells that holds all the nodes and edges along with their metadata. The graph can import a json representation of a model as well. Below is an example of such an json object. The json object is rather large even with only few elements, so only the top part shown in the example:

```
{
  "cells": [
    {
      "position": {
        "x": 40,
        "y": 40
      },
      "size": {
        "width": 100,
        "height": 40
      },
      "attrs": {
        "text": {
          "text": "Hello"
          .
          .
          .
```

Get started with X6: X6 works in both simple HTML applications and is compatible with a large number of JavaScript frameworks such as React, VueJS and Angular.

To quickly get started and try out X6 in a simple HTML setting you will need to create a HTML file, as seen in the code example below. First install the library with npm:

`npm install @antv/x6 --save`

or download it from their website:

`https://unpkg.com/@antv/x6/dist/index.js`

On `line 1` we create a div-tag with an identifier id. This is the area in the document the model canvas will be rendered. We import the library on `line 2` and `line 4-11` we declare and set the graph itself.

```
<div id="container"></div>
import { Graph } from "@antv/x6";

const graph = new Graph({
  container: document.getElementById("container"),
  width: 800,
  height: 600,
  background: {
    color: "#F2F7FA",
  },
});
```

3.2.5 Comparison and conclusion

	Licence	Graph visualization	Complexity*	Docs	Visual tool	D/M/E**
Java Swing	none	no	Hard	Okay	Yes	Y/N/N
JavaFX	none	yes	Medium	Good	Yes	Y/Y/Y
GoJS	needed	yes	Low	Good	No	Can be
X6	none	yes	Low	Okay	No	Can be

*Complexity is based on how hard it is to develop a graphical solution with graph visualization and relations. Hard meaning that the framework is not natively supporting graph visualization thus a developer needs to implement it himself.

Target **Desktop / **M**obile / **E**mbedded Systems. Java Swing is primarily for desktop applications where JavaFX can target all three platforms. GoJS and X6 are JavaScript libraries that, depended on which application they are used in, can be used on all three platforms. With other words, if the framework its used in supports JavaScript it will work.

Documentation: X6 has a rather large documentation, it is however only in Chinese. Even with a translator plugin for the browser its hard to get the right translation sometimes. Other than that X6 provides a very comprehensive documentation.

After experimenting with the Swing package, where we tried to build a larger application that included components with the functionality to be moved from one window (stencil) to another (model canvas), and where these components should have the ability to form mutual relationships in the form of arrows (parent node, child node relationships). We realized how difficult it was to implement in this package. Swing is not a good fit for entity relations and those kinds of models. Java Swing is a powerful tool, however its designed for a more traditional, form-based user interfaces as described earlier with buttons, labels, tables etc. Although it is possible to create a graphical user interface with components that have entity relations in Swing, it is an unnecessarily difficult task that can be handled much better with another package, which we will get into later in this section.

The next logical step was of course to look at JavaFX since, as described earlier, this is meant to be the successor to Java Swing. We experimented again with creating an application, and tried to create a way to build a graph with nodes that can have relationships with each other. Although we managed to create something basic as a starting point, we had to realize that the problem was again to create a solution that was in any way sensible. A basic graph system would have to be implemented from scratch, which alone is too big a task in relation to this project. We therefore began to investigate other possibilities. Simultaneously developmening a test application in JavaFX, we also tested existing threat frameworks, including Threat Dragon. This framework works online in web browsers, which inspired us to think along those lines instead.

In our search for a graph visualization tool that could be used for web pages we first found the GoJS library. This was a huge step forward compared to Java Swing and JavaFX in terms of graph visualization. With very few steps we had a simple web page set up with a single HTML-file with a linking to the GoJS library. It was incredibly easy to set up a simple node graph and create the necessary connections between them. After a few days of work, it was also possible to create a more complex system where components could be drag-n-dropped from a stencil window to the main canvas. The only problem was that GoJS is not an open source tool but instead requires a license. So back to searching for another framework that could be used on web pages. Examining the documentation

for Threat Dragon, we could see that they used an open source graph visualization tool called G6 from the AntV group. This was naturally the next tool to be tested.

The conclusion of testing the various tools has led to the conclusion that X6 is the tool that best matches this project. X6 is open source under the MIT license and extremely easy to develop in. Although the documentation requires a few translations here and there, it is excellent especially because it is full of useful examples. Furthermore, X6 can be developed for both desktop, web, mobile and embedded, as long as there is a JavaScript framework it can run on.

3.3 Threat modeling Frameworks and tools

Within threat modeling there are many tools and frameworks, in this section I will describe some of them that I have found and found useful. Especially OWASP[26] which is a non-profit organization that works to improve the security of software. They do this through tools created by a strong community of several thousand people who create open source projects and through education and training.

3.3.1 OWASP Threat Dragon

Threat Dragon[27] is a free, open source threat modeling tool from OWASP and follows the principle of the Threat Model Manifesto[28]. Much like in the CORAS-example, its an graphical tool where you build a threat model based on the methodology of either CIA, LINDDUN, STRIDE or a generic model, and decide on how to mitigate them. The Entities of Threat Dragon is shown in picture 3.10, where *Actors* are the actors of a system, users, admins, hackers, etc., *Processes* can be servers (app, web, mail), *Stores* can be databases etc, a dotted line is a thrust boundary, which is a zone where components within trust each other and do not have to question each others integrity. Arrows between are data flows.

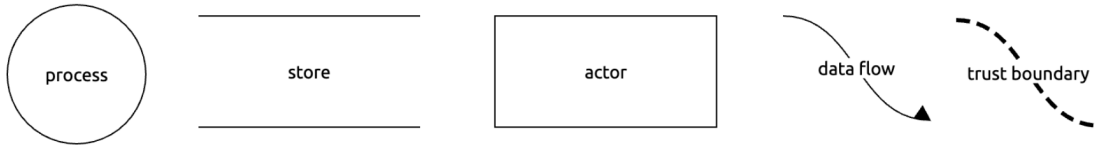


Figure 3.9: Threat Dragon Entities

In figure 3.10 an example of a system is shown. Users and admin is within a trusted boundry. They have a dataflow to a webserver that has a dataflow to a database. Lets say we chose to build a STRIDE threat model, we can, for example, click the web server-process and add threats to it based on STRIDE. For example add spoofing as an threat or what mitigation we are taking against the threat.

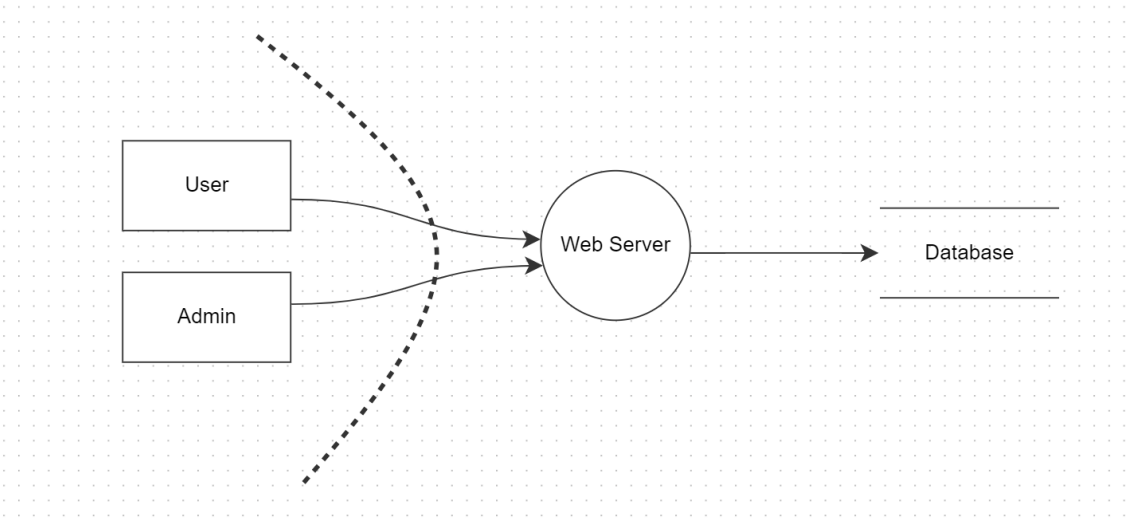


Figure 3.10: Threat Dragon Example

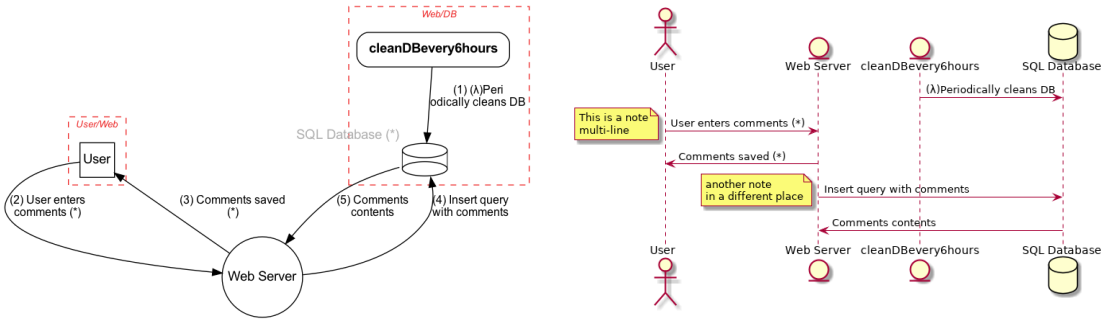
3.3.2 OWASP pytm

OWASP pytm is a threat modeling framework in python aimed at developers. It is not a completely finished tool that people can use out of the box, as we have seen with the other examples in this section, but a tool that the developers can use to describe a system through code, and thus start a dialogue and build a more secure system.

Based on the users definitions, it can generate data flow diagram, sequence diagram and the threats to the users system. However, it is not necessary to be a developer or to know Python as such since you build your model as shown in the code snippet below. In other words, it is not a framework where you build your threat model into an existing code base

```
from pytm.pytm import TM, Server, Datastore, Dataflow, Boundary, Actor

tm = TM("my test tm")
tm.description = "another test tm"
tm.isOrdered = True

User_Web = Boundary("User/Web")
Web_DB = Boundary("Web/DB")

user = Actor("User")
user.inBoundary = User_Web

web = Server("Web Server")
web.OS = "CloudOS"
web.isHardened = True
web.sourceCode = "server/web.cc"

db = Datastore("SQL Database (*)")
db.OS = "CentOS"
.
.
.
tm.process()
```

Compared to the other tools described in this section, this is a more complicated one. Here you must first have python installed and import pytm, and write everything in code.



(a) Data Flow Diagram

(b) Sequence Diagram

Figure 3.11: OWASP pytm diagrams

One can argue that this forces users to work and think more about their system, but it also requires far more working hours, especially on larger systems.

Once you have described your system, you can generate a report, either in the form of a data flow diagram or a sequence diagram, see figure 3.11

3.3.3 CORAS

CORAS is a model driven approach to risk analysis developed by SINTEF[29]. It provides a graphic language tailored to threat and risk modeling using a diagram editor that works directly off an browser or as downloadable executable. CORAS is a language that can be used to identify potential security threats, as well as the vulnerabilities, incidents, and assets that may be involved. It is particularly useful for modeling threat scenarios and discussing possible mitigation strategies during a risk assessment.

In order to build the threat model and make your analysis, you will define a context and build your model be your assets, the threats to these assets, vulnerabilities of your system and possible treatments or action you are already taking to mitigate them. Some of these elements are illustrated in picture 3.12

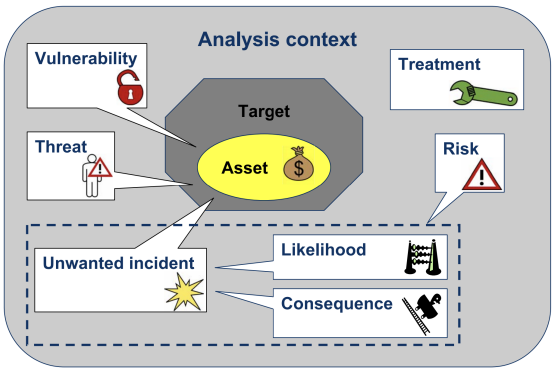


Figure 3.12: Elements of the analysis

In picture 3.13 an example from is illustrated from their book[30] pg. 48, showing a simple example of the scenario where virus protection program is not up to date and a virus might exploit the server.

Figure 3.13: Risk scenario

3.3.4 Threagile - Agile Threat Modeling

Threagile is a toolkit for agile threat modelling that can run as either a command-line interface or a server with REST API. As with 3.3.2, this is not a plug-n-play tool where you quickly build a model. This is an advanced tool aimed at developers. You must have knowledge of YAML and agile system development. The idea of this type of threat modelling is to bridge the gap between classic threat modelling and agile development teams. The framework uses declerative YAML-files containing; data assets, components, communication links and trust boundaries. The YAML-file, even for a small example are huge, see this example: https://run.threagile.io/threagile-stub-model.yaml. Below is a snippet of just one asset:

```
data_assets:
  Some Data Asset:
    id: some-data
    description: Some Description
    usage: business # values: business, devops
    tags:
    origin: Some Origin
    owner: Some Owner
    quantity: many # values: very-few, few, many, very-many
    confidentiality: confidential # values: public, internal, restricted,
        confidential, strictly-confidential
    integrity: critical # values: archive, operational, important, critical,
        mission-critical
    availability: operational # values: archive, operational, important,
        critical, mission-critical
    justification_cia_rating: Some Justification
.
.
.
```

3.3.5 SeaSponge

SeaSponge[31] is yet another threat modelling tool. Its open sourced and client-side browser running in HTML5, developed for Winter Of Security 2014[32]. Much like in the previous examples, you build a threat model and use this model to assess threats to your system and how to mitigate them. The entities of SeaSponse are: *Boundary*, *External*, *Process* and *Store*. As seen in picture 3.14, the framework seems a bit clumsy. There are no manual, not sure how to apply a trust boundary.

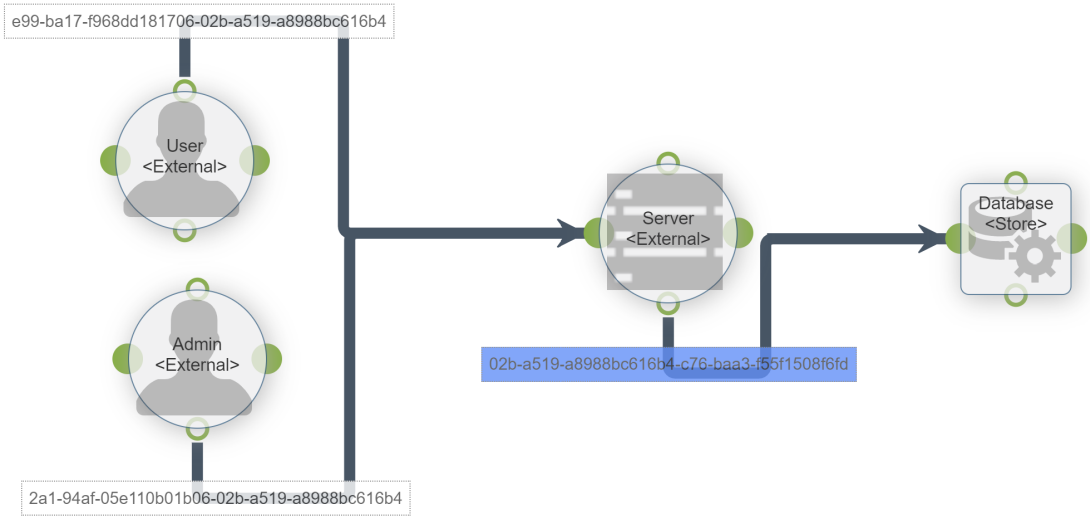


Figure 3.14: SeaSponge Example

3.3.6 Microsoft TM Tool

The tools presented so far in this section are all limited to tools for building a threat model and helping users to analyze and communicate the security design of their systems. Microsoft TM Tool[33] goes a step further, their tool can also help by making suggestions on how to mitigate security issues. Although the principles are basically the same, their tool offers quite a few more entities, both within data stores, external actors and processes, which can help create a more detailed and accurate model of your system. In addition, it is possible to generate a report in HTML format which works quite well. Although it is a free tool, it only works for Windows systems. It must be downloaded and run as an application in Windows.

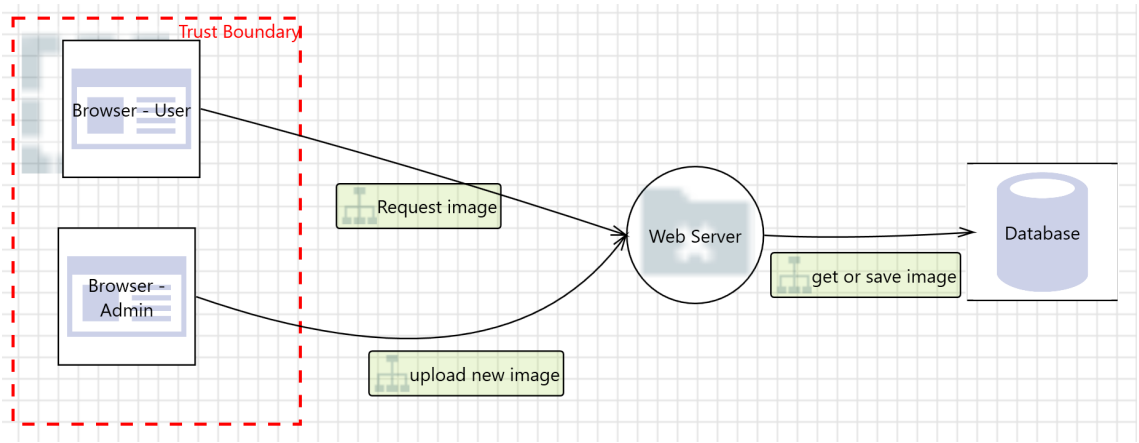


Figure 3.15: Microsoft TM Tool Example

The application has two modes; design and analysis. In design-view we build the model, in the same way as the other tools mentioned, this is illustrated in figure 3.15. When the model is built, we can switch to the analysis view, see figure 3.16. This provides a list of possible threats based on STRIDE. If you click on one of these threats, you will see a more detailed description of the threat and possible mitigations

Threat List

	Diagram	Last Modified	Title	STRIDE Category	Description	Interaction	Possible Mitigation(s)	Severity	SDL Phase
0	Diagram 1	Generated	N	An adversary can leverage the weak scalability of	Denial of Service	The default cacl	Request image	Override the default Identi	High	Design
1	Diagram 1	Generated	N	An adversary may sniff the data sent from Identi	Information Disclosure	An adversary m	Request image	Ensure that all traffic to Ide	High	Design
2	Diagram 1	Generated	N	An adversary can bypass authentication due to n	Spoofing	An adversary ca	Request image	Use standard authenticatio	High	Design
3	Diagram 1	Generated	N	An adversary can get access to a user's session du	Spoofing	An adversary ca	Request image	Implement proper logout w	High	Implementatior
4	Diagram 1	Generated	N	An adversary may issue valid tokens if Identity se	Spoofing	An adversary ca	Request image	Ensure that signing keys ar	High	Design
5	Diagram 1	Generated	N	An adversary may guess the client id and secrets	Spoofing	An adversary m	Request image	Ensure that cryptographica	High	Implementatior
6	Diagram 1	Generated	N	An adversary can leverage the weak scalability of	Denial of Service	The default cacl	upload new ima	Override the default Identi	High	Design
7	Diagram 1	Generated	N	An adversary may sniff the data sent from Identi	Information Disclosure	An adversary m	upload new ima	Ensure that all traffic to Ide	High	Design
8	Diagram 1	Generated	N	An adversary can bypass authentication due to n	Spoofing	An adversary ca	upload new ima	Use standard authenticatio	High	Design
9	Diagram 1	Generated	N	An adversary can get access to a user's session du	Spoofing	An adversary ca	upload new ima	Implement proper logout w	High	Implementatior

Figure 3.16: Microsoft TM Tool Threat List

3.4 Reflection

It is our belief that all these frameworks just described can solve the problem of building systems graphically and create a basis for a threat assessment. Common to these frameworks is their approach to threat assessment, they let users build a more abstract picture of the system, understood in the way that these frameworks are intended as a way of describing systems and setting up a discussion among those who are responsible for examining the threat picture for a given system. In other words, it requires greater expertise in the area of IT security to be able to define the systems and identify potential vulnerabilities, as one must have knowledge of basic elements within IT security such as the CIA model, STRIDE, processes, actors, trust boundaries and how they together form a system. Even if you have this basic knowledge, you still need to be up to date with the current threat picture, which vulnerabilities have been published and how to incorporate these into the model structure.

Although the Microsoft TM Tool goes a step further and provides possible solutions to mitigate the security problems for one's model, it is still a tool that does not reach the huge group of people who do not have the necessary skills to use these tools and it is still a quite complex framework.

In section 4, we will explain the basic idea of a system that addresses the target group, a tool that is open source and easy to use for the very common person without deep technical skills and a way to formulate a model of a system much more concrete. Where the aforementioned frameworks set out to create a discussion among IT security experts within companies or as individuals, we try here to create a very concrete tool where you simply build systems as they are with the components it contains.

4 The idea

4.1 Core idea

The main idea is to have a way to formulate a system of any kind through a graphical interface that is intuitive and easy to use for developers or procurers of a system. The term system is an abstract concept as it can cover everything from a large network of computers, a single PC or just a mobile phone. A system can also cover an online service, a database for example or even something as simple as a Hue bulb that can be connected to the internet. A system is therefore the entity we choose to focus on. To give meaning to the system, we must define the system's inputs, outputs, its states and the control mechanisms that apply in the system:

- **Inputs** are in simple terms what goes into the system. Examples of this can be a keyboard plugged into a PC, connected by a USB cable or a router with an Ethernet cable plugged in to the network interface of a PC or the data from a sensor wired over Bluetooth.

- **Outputs** are the exact opposite of the inputs. Here we define everything that exits the system. This could be a printer connected to a PC, loudspeakers, a monitor, the data leaving a database, or anything else you can define as going out of your system.

- **Control** is all the mechanisms in the system that have an active influence on the behavior of the system and its processes. It can be user commands or instructions, rules in the system, allocation of resources such as CPU time or memory

- **State** can be a snapshot of the state of the entire system at a given point in time, the state of a program with current variable values, or the status of a process.

The idea is that you choose the system you have a need to uncover. Let's use a simple PC as an example. This PC has a keyboard and a mouse as input. A screen and a printer as output. To connect these components, a USB or perhaps HDMI cable is used. In addition, it has a huge number of states it can be in while the PC is on, and depending on the user's actions, these states also change.

An overview of a general system is illustrated in figure 4.1. As a starting point, the system is a square box surrounded by the four basic elements input, output, state and control. When the user starts building the system, he adds components to the model, gives them a context in the form of a name and a connection to one of the four elements. As seen in the figure we have defined a `Component 1` to `Component n` for the `input` element. This is to illustrate that it should be possible to add as many components as desired to the input element and to the others as well.

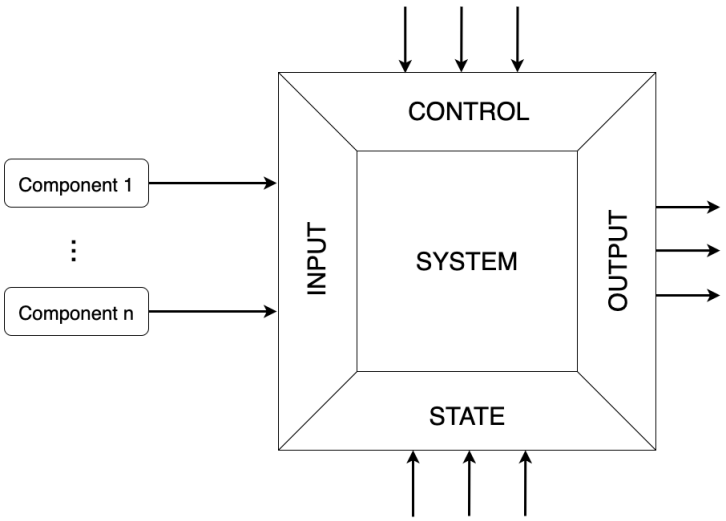


Figure 4.1: Overall picture of the system

Now we have a basic understanding of what a system is with the four elements input, output, state and control. But when can we say with certainty that we have formulated a system? If we go back to the example of a PC, which in itself can be quite a large system to formulate, then this PC can perhaps form a small part of a large network you are trying to define or It may also be that you only need to examine a subset of the system we call PC. Let's assume that the PC has a router connected via Ethernet cable as input, and it is only of interest to examine this part for vulnerabilities. Then the definition of the system changes to the router. That is, the router IS the system, the internet and the cable into the router is the input and the cable out of the router is our output. State and Control must now be defined for this system.

The idea is that you choose your own system, however small or large it may be. It goes without saying that the larger a system you try to build, the more complex and difficult it is to formulate. Therefore, the ideal solution is to be able to build smaller systems that can be combined with other systems in the same model. Let's take the example of the router. We have defined a router with its inputs, outputs, states and controls. This is a fairly simply task. Now you save this system and define a new one, a PC perhaps. Import the router as use this as an input.

This project is an attempt to make the start of a total system that can do just that. We strive to create the foundation of this system where users can model a basic system with inputs. Output, state and control are therefore not the focus of this project, but the thoughts and considerations made on these three will be elaborate on in this section.

As described in section 2, this project is aimed at people who do not necessarily have great expertise when it comes to IT security, vulnerabilities and how to build a threat model of a IT system with the traditional tools. Therefore, this project focuses on ease of use and simplicity with a system where you just need to describe the components in your system.

Based on the research in section 3, we have come to the conclusion that the solution will be browser-based and online. In this way, users do not have to download and install software before using the system. Figure 4.2 illustrates what a user sees when entering the website for this projekt. Left side is a stencil window where the user can select components from one of the four elements: Input, Output, State and Control. Of course

only Input is available so far. A component has a name, e.g. `Keyboard` or `Router`. These components are loaded in from a database. The user has the ability to create new components from a sub-page on the website. The idea here is that the user can drag and drop components from the stencil window into the model canvas and attach the components to their respective elements of the model. The user then has the option to right-click on components to add child components or to inspect a component. When the users model is ready they click the `Run`-button which sends the current state of the model to a back-end server which in return send a list with all the associated vulnerabilities found. These vulnerabilities are displayed in the footer threat list.

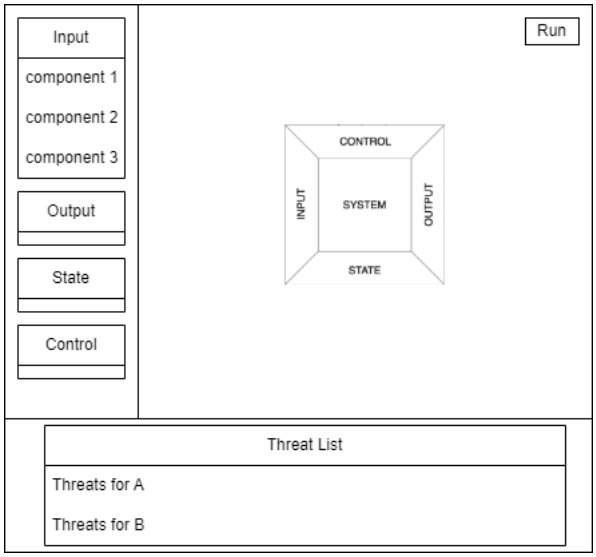


Figure 4.2: System prototype

4.2 Input, Output, Control, State

So far, we have explained the overall idea behind the system. What is interesting, however, are the four elements in the system and their potential interplay. The ambition of this project is to investigate whether it is possible to create such a system and then create the basis for it. First we will explain the four elements and then provide our theory of how they can work as a whole, and finally reflect on whether it is a sustainable idea for further research and development.

4.2.1 System Inputs

The inputs of a system can be defined as information or data sent into computer for processing[34]. Examples of these can be: keyboard, mouse, microphone, webcam, touchpad, scanner and many more.

Figure 4.3 is an illustration of an input component. In this case a keyboard in inspection mode, where we are able to see its child components. A keyboard might have some drivers and firmware installed, as seen here. These are called products and are added by the user by right clicking the keyboard-component where he can search on products based on either a `vendor`-name and/or a `product`-name.

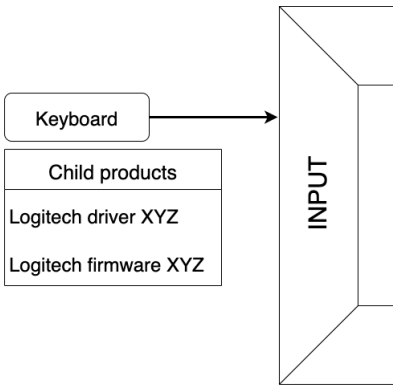


Figure 4.3: Example of input component

This is how a user builds input components by naming them and adding associated products to the component and linking it to input. The user can attach as many components to input as needed, however they need to be added one by one. This is of course trivial and only a first step towards a more advanced system where the idea is to expand these components. For example, if, as in figure 4.4, there was a router component that is input to the system itself has a component as input, so we get the tree-like structure of input components that are related. This should of course be presented in a better we, as a single component where the user could view it in a tree structure.

The threat evaluation for input components are based on a CVE database lookup on all of the products. If a product has any vulnerabilities associated it is reported back to the system. This is also a trivial process and can fairly easily be modified backend to adapt to a more complex input functionality as discussed in figure 4.4.

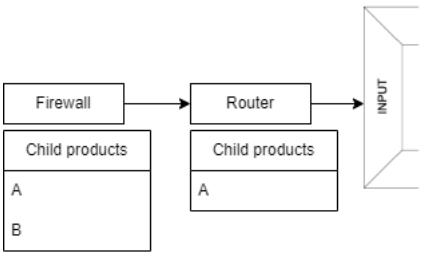


Figure 4.4: Possible idea to input component

4.2.2 System Outputs

The output can be defined as all the information or data sent out of a system[35]. Examples of these can be video output to a monitor, audio output to a speaker, text and images sent to a printer or even data written to a database.

The output of the system is not implemented in this project, however here is our take on how the output could be incorporated and what impact defining the output components will have on the overall threat evaluation. Figure 4.5 is an illustration of how we could model the output of the system. This is very similar to the way we model the inputs because both input and output of a system is rather simply to define.

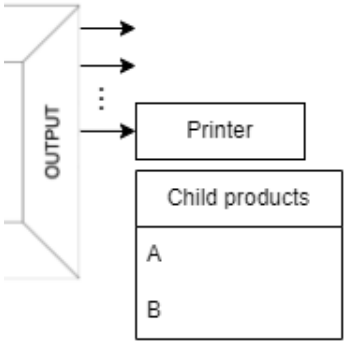


Figure 4.5: Example of output component

We could treat both input and output as single objects and find all vulnerabilities in both of them and just print a big list of the system's vulnerabilities. The problem, however, is that some input components can also be defined as output components. For example, a router can be a input component to the system with data read from the internet, however it can also be a output component because request and data are sent to the website we are trying to access. You can of course just model this in the structure of your system, so that there is an input router and an output router with corresponding child components.

This creates two additional problems though: we start to have a model with redundant components that only differentiate themselves by whether they are input or output to the system. This means that larger models of a system can risk becoming unmanageable and harder to understand. The second problem is that cases may arise where an input component has the same vulnerability or vulnerabilities as an output component and there will therefore potentially be a number of redundant vulnerabilities in the final report.

One way to avoid the problem with redundant vulnerabilities is illustrated in figure 4.6. Before we do invocations of the methods that administrates the rest calls to the CVE API, we can do a comparison on the components. If a component in the set of inputs is identical with a component in the set of outputs, then we simply discard one of them, in this case component `c1` from the set of `outputs` is discarded.

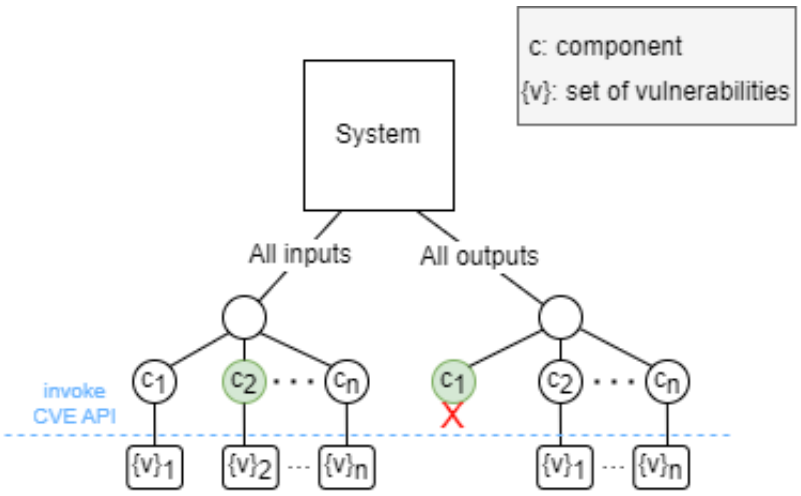


Figure 4.6: Example of output component

However, this does not fix the problem of redundant components in the model. We still

need to define, for example, a router for input and output. A solution here could be to be able to make a connection from the component to both input and output. It does, however, present some graphical challenges that must be considered. Another solution could be to add an option to be able to tick the output in the form of a radio button on the component itself. In this way, it visually appears to be input, but implicitly it also belongs to output.

4.2.3 System Control

So far, we have looked at input to the system, which in itself is relatively simple to define and formulate in a system. The same was true for output, but when we start combining the two, the complexity quickly grows and leaves a lot of design choices to be made. In this section, we will try to explain control in a system, how we can possibly map control and get it to interact with the rest of the system.

Controls in a system are the mechanisms that handle and regulate the system and its processes. If we take a PC as an example, it has a CPU (central processing unit), which is responsible for executing the instructions in computer programs and making calculations. In other words, it controls the data flow of a computer system by fetching instructions from memory and executing them. The chip design of the CPU can be seen as one large algorithm that follows a set of rules or instructions that determines how the system should behave.

A PC's operating system, on the other hand, is responsible for handling and controlling the system's hardware and software components. It controls file management, memory management, processes and devices. However, Control can also be a dedicated chip specially designed for the behavior of a robot that automate some work task in a company, or a software that determines the behavior of light signal in traffic. Control can also refer to access control and security policies to regulate who or what is allowed to access specific resources in a system.

So even if we can define very precisely with words what control in a system is, it becomes a bigger challenge to define it in a project like this. In order to formulate control and use it in a model of a system, we must find a way to generalize the concept such that it becomes building blocks for the model as we have seen with input components and output components.

One way to achieve this could be by defining some templates for an operating system, a cpu, access control, etc. respectively, where the user can choose those that are interesting for the system. Each of these templates must then be able to be filled in with information, for example, the exact operating system or CPU used or a template where you can choose between a number of scenarios in relation to access control. In other words, a comprehensive solution would have to be created for every conceivable definition of control.

4.2.4 System State

Now we are moving into an area where a description of a system becomes more technical. For input and output, we simply had to know the names of the components to be used and add them, but here the user must be able to define a snapshot of a given area in the system. An example could be a user who has defined his system as being a program, where the input to the system is user inputs typed into a terminal, and the output of the system is the response from the program written in the console. Here, a state component in the model could be a description of the program, where at a given moment in the program's instructions, a specific value is assigned to a variable in the program.

4.3 Assumptions

In a final solution for such an online tool that communicates with backend APIs and databases, it would be necessary to implement security measures such as authentication of users. Right now anybody with a link to the website could add and delete components in the database. Its assumed that such measures are implemented. Furthermore we also assume that the users do not use the current solution with bad intentions. There is no measures to prevent users from making a million REST call to the backend, which in return would make even more calls to external API's.

4.4 Reflections

It is our impression that the complexity of each step of this project; input, output, control, state increases exponentially. We see both input and output as something that can be realized without problems and even built upon. These are also still at a level where ordinary people should be able to understand and use the solution. State and Control, on the other hand, is a far more advanced approach to such a system. Even if we were to succeed in implementing a system that we have thought about, we believe that it is starting to move away from the original goal; a framework for ordinary people, and towards the IT security experts again.

5 Technical solution

The software project is divided into two separate systems: A backend and a frontend. The frontend part refers to the user interface, which is what the user of the system interacts with. The backend part handles all server-side processing and data management. It is often a good idea to separate a larger software project, which this one has potential for, as this makes the code modular and easier to maintain as you clearly separate the responsibility of the tasks. The code therefore becomes easier to handle and easier to understand. This is especially important if further research is to be done on this project or if new functionality is to be added. At the same time, it also makes it easier to handle the load on the system, as you can more easily decide whether it is the frontend or the backend that must handle certain things.

In this section we will first dive into the frontend part of the project. This includes the use of the VueJS framework and the graphical library X6. VueJS is a framework that has only been around since 2016 but has gained huge traction and increased in popularity over the past few years. VueJS provides a robust and reactive user interface, which works well with X6, therefore a natural choice for the technical solution. We also get into the different libraries that are used, for example the network library for handling REST requests.

Next we will get into the backend part of the system. We will examine the use of the NodeJS framework and ExpressJS, which together form an exceptional framework for developing scalable network applications. We will also explore the integration of a database solution with MySQL. Furthermore we will discuss the CVE API and the algorithms designed to handle the data exchange between the National Vulnerability Database and the our system.

This section will therefore provide a detailed insight and understanding of the entire code base, the architecture and the design choices we have made along the way.

5.1 System architecture

Figure 5.1 shows an abstract representation of the total system. The users interactions are limited to the frontend, while the backend operates behind the scenes. The communication between the frontend and backend is done through REST calls, and in response, the backend interacts with the database and the REST API of the National Vulnerability Database.

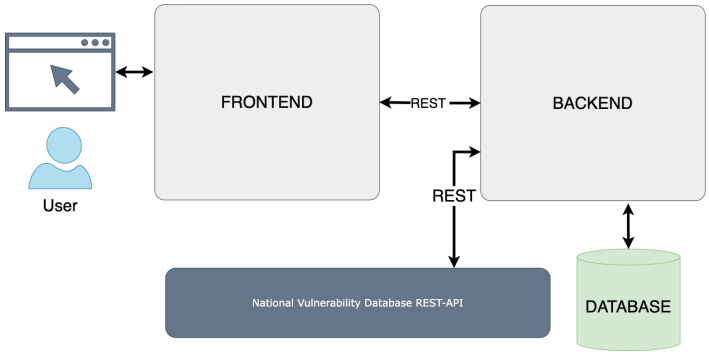


Figure 5.1: Overall picture of the system

5.2 Frontend

5.2.1 The Vue framework

Before we get into the details of the frontend codebase, we will provide an overview of the VueJS framework. This will provide a foundational understanding of the core functionality and thus make it easier to explain the design choises made.

VueJS is a framework that has grown in popularity in a few years and has proven to be extremely fast and robust, it outperforms both React and Angular. It is one of the most downloaded and used JavaScript frameworks with over 1.5 million users[36]. It is designed to be lightweight and easy to use which makes it ideal choice for this project. One of the key features of Vue is reactive data binding which allows the framework to automatically update components in response to data changes.

```
<template>
  <div>
    <h1>{{ message }}</h1>
  </div>
</template>

<script>
export default {
  data() {
    return {
      message: 'Hello from Vue3'
    }
  }
}
</script>

<style>
<!-- CSS styles here -->
</style>
```

Listing 5.1: Basic Vue3 JS Example

A simple Vue 3 single-page project typically consist of the main app component that acts as the root component of the application. This component would typically have a header, a navigation menu and the main content component. Each of these components is defined as a .vue file and includes the three building blocks of a Vue compenent: Template, script and style. This is illustrated in code listing 5.1.

- **Template**: is the section where you define the structure and layout of the compenent. This is done in HTML and a special syntax that allow for data binding between the template and the script. As seen in the code snippet we use the special syntax *{{ message }}* to bind the message variable exported from the script and display it directly in the DOM. The template section allows for much more functionality other than reading single variables, such as doing conditional logic, looping through arrays and much more. This will be explained in greater details later.
- **Style**: is the section where we define the css-styling of our component, just as in regular HTML/CSS.
- **Script**: is the section where all the logic and behavior of the component is defined. As seen in code snippet 5.2, the way we export data; variables, constants, arrays, etc., and expose them to the template is by defining *'export default {}'*. Within this export we can define and invoke methods like we normally do in a JavaScript application. We also have access to different parts of the components life-cycle which is

specific points in time where methods are executed, such as when the app is first created or mounted, or when its updated. The way we can hook into these points in time is illustrated in 5.2.

```
export default {
  data() {
    return {
      message: 'Hello from Vue3'
    },
  created(){
    // do stuff when app is created
  },
  mounted(){
    // do stuff when app is mounted
  },
  updated(){
    // do stuff when app is updated
  },
  methods: {
    myFunction_a(){},
    myFunction_b(){}
    }
  }
}
```

Listing 5.2: Example of functionality of export default

Figure 5.2 shows the life cycle of a Vue component with the timeline going from top to bottom. All Vue components in a project go through this series of steps and allow us to add code to specific stages. All the white boxes with a red border are hooks we can access directly in our script (including the setup stage)[37]:

- **beforeCreate** is called when the instance is initialized. This is before options such as data() or computed is processed.
- **created** is called after the instance has finished setup states like reactive data, computed properties and methods
- **beforeMount** now all reactive dafa, computed properties and methods has been set up. DOM nodes has not yet been created.
- **mounted** all synchronous components have been mounted and the DOM tree has been created
- **beforeUpdate** is called right before the component is about to update the DOM tree due to some reactive state change
- **updated** called after the DOM tree is updated due to the reative state change
- **beforeUnmount** called before a component is unmounted. The compnonent instance is still fully functional at this stage
- **unmounted** all child component and associated reative effects is stopped. This hook can be used for clean up.

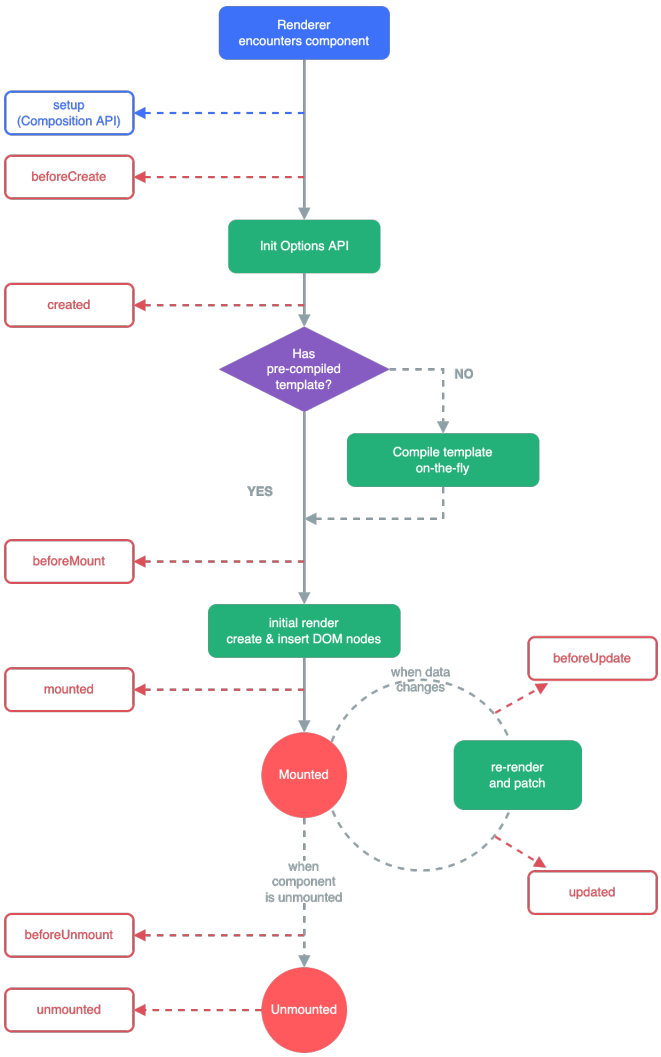


Figure 5.2: Vue3 Lifecycle

5.2.2 Frontend Overview

Now that we have a better understanding of the basic elements of Vue we can continue with the actual code base for the frontend part of the project. Notice that we will explain the codebase with a mixture of the real code and pseudo code. This is necessary because some of the code is really large and sometimes JavaScript is intertwined with HTML and special Vue syntax, which results in rather huge HTML trees. We will let the reader know when pseudo-code is used. Real function names are used in pseudo-code such that the viewer can look it up in sourcode.

Figure 5.3 is a diagram of the frontend system as a whole. The gray box represents root-level of the code-base, the orange boxes are folders: *Views*, *Components*, *Router* and *Public*, and blue boxes are the source-files.

Index.html is the public file a users browser is requesting. This in turn communicates with the *main.js*-script that is mounted in *App.vue*, which is the top-level HTML file. At the moment the web-application has a main view for modeling the graph, a database view that displays the components from the database, a view for creating new components to the database and an about view. When the user navigates between these view he is

invoking the *router*-script which is responsible displaying the correct view to the user.

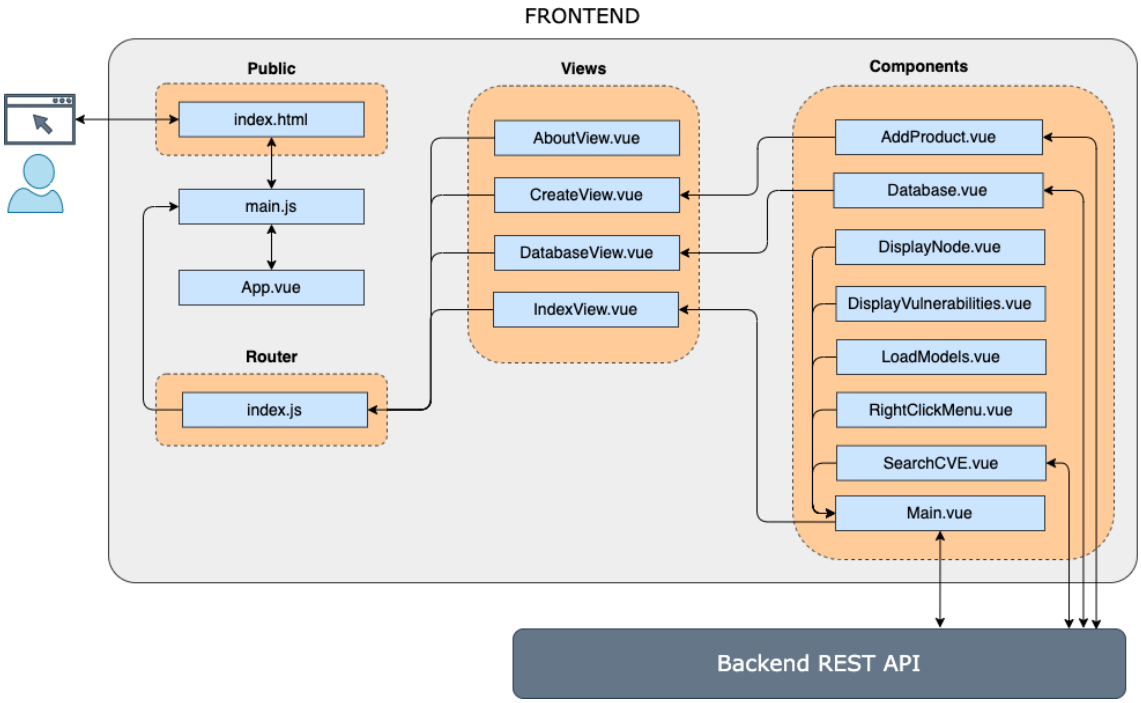


Figure 5.3: System architecture with focus on the frontend

Both Views and Components have the same structure as described in example 5.1. The idea in dividing it into these two categories is that Views handles the top layer of what the user sees. Usually some basic HTML and structure, where components are smaller and reusable code used in Views. An example could be if you have a component that has searching functionality, then the UI and possibly logic for this could be used in different places in the application. By separating this, you create a more manageable codebase that is easier to make changes to and code that takes up less space.

The entry point for the user into the whole application is the public index page, as seen in listing 5.3 it contains the app identifer used by Vue to inject the compiled vue project.

```
<div id="app"></div>
   <!-- built files will be auto injected -->
```

Listing 5.3: Public > Index.html

The root of the application is the main javascript file seen in listing 5.4. This script is responsible for creating the vue app instance, importing the router (for navigation between component views) and mount the application instance with *#app* to the DOM element *app* in listing 5.3.

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './router'

createApp(App)
   .use(router)
   .mount('#app')
```

Listing 5.4: root > main.js

The first View rendered to the user is *IndexVue*, which loads in the *Main*-component. This is a rather large component that maintains the to a large degree the logic and behavior of the whole application. All of the Views components in this project has no other function than importing their respective components. The design choice for this is just that for future development it will be easier to separate the concerns of the basic components and the structural views. Listing 5.5 is the View-component for the Main component. The views for About, Create and Database is similar to this code.

```
<template>
  <div>
    <Main />
  </div>
</template>

<script>
// @ is an alias to /src
import Main from '@/components/Main.vue'
export default {
  name: 'Index',
  components: {
    Main
  }
}
</script>
```

Listing 5.5: Views > IndexVue.vue

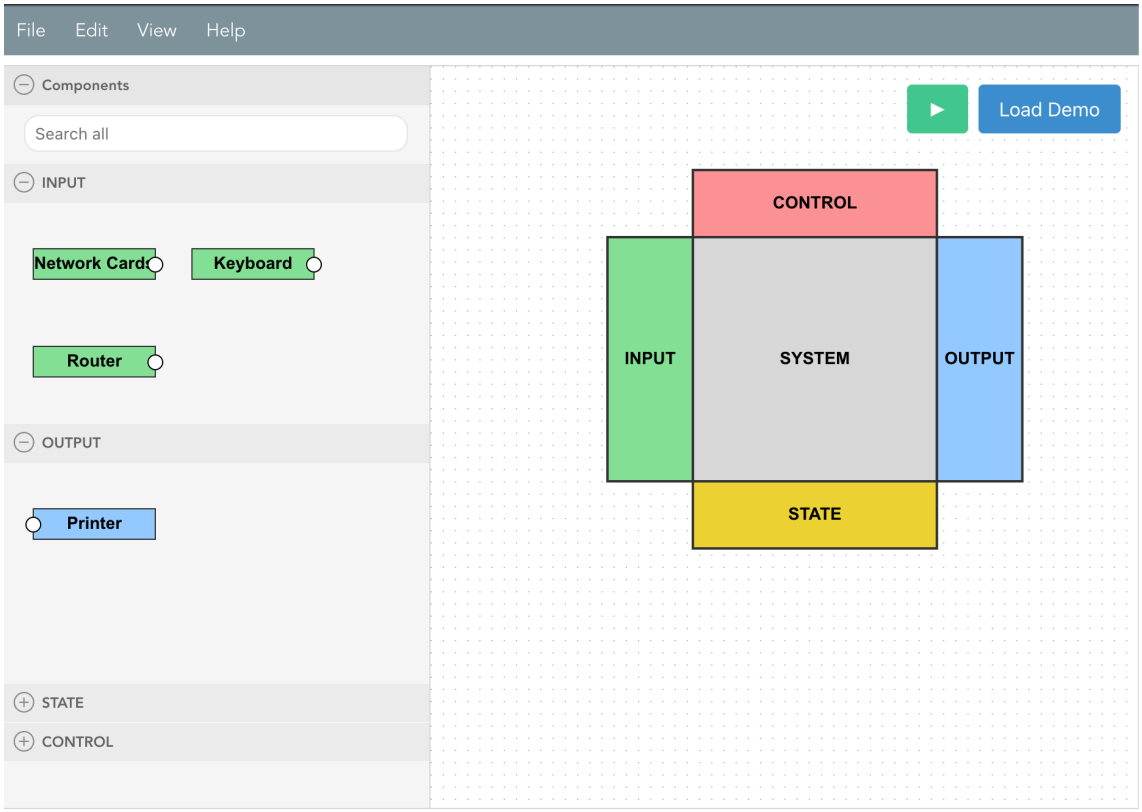
When users initially enters the website of the application, they are met with a screen similar to figure 5.4. The header is a is a clickable menu with the menu-items: File, Edit, View, Help. Right now only View and Help has any functionality. When one of the items are clicked an sub-menu is extended. View has the option to navigate between the main view and database view. Help has the option to navigate to an about view and tutorial view. File and Edit are just placeholders for future work.

The left side of the screen is the stencil area, which displays the main components from the database: Input, Output, State and Control. The right side of the screen is the model canvas. This canvas is loaded with default components that makes a general system without any child-components.

Users can drag and drop components from the stencil window into the main canvas. Each component has a port attached, the white circle. When the component is placed on the canvas window one can extend an arrow (link) from the components port to its respective parts of the system. E.g. In the figure one could drag in a Network Card component and create a link from that to the Input of the model, but not to the Output because they do not share the same category.

On top of the main canvas is two floating buttons located. The 'Play' button which will take the current state of the model and send it to the backend. When pressed the button will become un-clickable until a server-response is received. Besides the load button is a Load Demo-button, which opens a new window where users can load predefined models for a quick test of the system.

In the footer of the screen is the system threat list displayed. Initially this has no values. This list will be updated when a response is received from the server.



System threat list:

Component		Vulnerabilities	Severity
inputs		0	LOW

Figure 5.4: Screenshot of the main view

5.2.3 Components/Main.vue

The way we are displaying the main screen can be seen with the pseudo-code in listing 5.6 a basic HTML table is defined with a row for the stencil and a row for the canvas On `line 1-5`. Then we are defining the two floating buttons. The floating position of the buttons are handled with CSS in the style-section. The buttons are using Vues special syntax for if-conditions with *v-if* where *!waiting_for_response* is a boolean value that is set to true when the button is clicked by invoking the function *requestVulnerabilitiesFrom-Server*. The boolean is set to false again, when this function gets a response from the server. This way we can hide or show the button. `line 10` is simply a button that appears when the Run-button is clicked, to give the user a visual way of knowing the program is processing.

```
1 start table
2     start row
3       stencil window
4       canvas window
5     end row
6
7     define floating buttons:
8     <button v-if="!waiting_for_response" @click="
          requestVulnerabilitiesFromServer" id="btn"..> RUN </button>
9
```

```
10      <button v-if="waiting_for_response" id="btn" disabled>processing...</
            button>
11
12      <button id="btn-demo" @click="toggleLoadModelWindow">Load Demo</button>
13  end table
```

Listing 5.6: main.vue - template-section. line 1-14

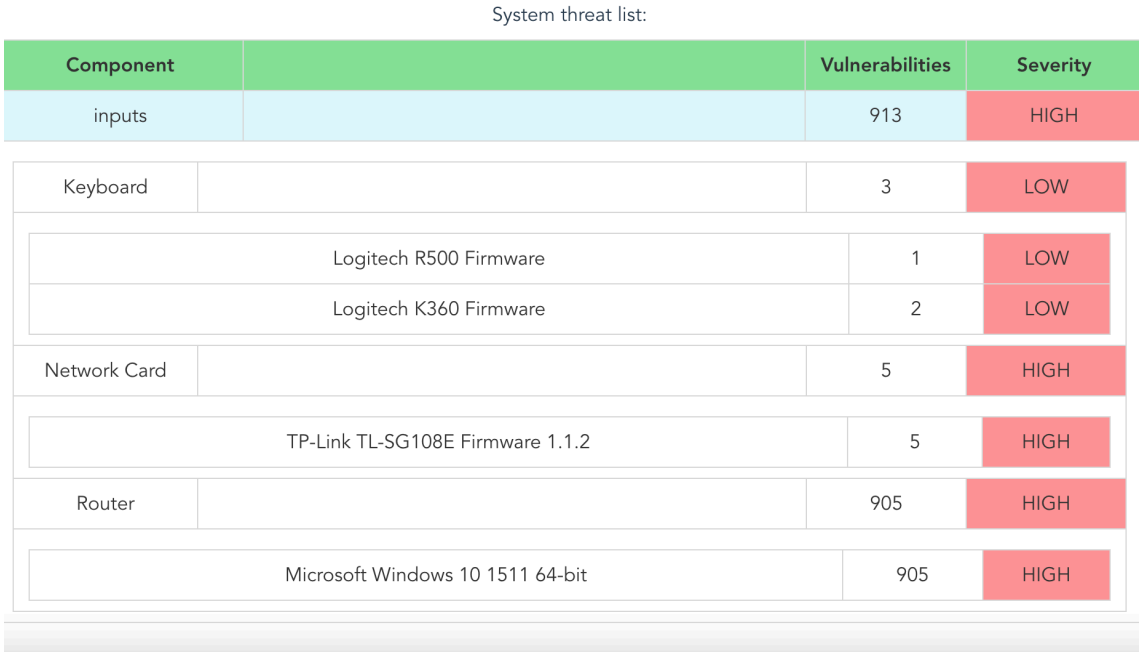
Besides assigning the boolean value true or false, the *requestVulnerabilitiesFromServer*-method is responsible for creating the rest call to the backed and send the backend the current state of the model. When it receives the response it will invoke another method: *response_to_display()* which is responsible for dissecting the json object it received. To see an example of such an json-object, see appendix A.1. When dissecting the json-object it places all the vulnerabilities and metadata into global variables that can be accessed by the template components, these variable is seen in listing 5.7. When the json-object recieved from the backend it is stored in the *response*-variable. From here we extract the different elements. *components* is child components, e.g. a Keyboard, of the main components (Input, Output, State and Control, and *products* is the associated products of child components, e.g. the driver for Network Card or Printer. Severity is the overall severity of the system.

In order to create a table that is expandable with child objects, we are maintaining three lists to keep track whether the table or child table are expanded or not: Opened, Opened2, Opened3. This is explained more detailed later in this section.

```
1
2   data() {
3       return {
4       ...
5           // response from server:
6               total_vulnerabilities: 0,
7               waiting_for_response: false, // switches 'run' / 'processing' on
                    button
8               components: [],
9               products: [],
10              severity: "",
11              opened: [],
12              opened2: [],
13              opened3: [],
14              response: [{
15                  "id": 1,
16                  "name": "inputs",
17                  "components": []
18              }]
19      ...
20
21      }
22  }
```

Listing 5.7: main.vue - script-section. line 156-170

For the footer of the screen we have the list of threats to the system. The way we display them is going through each main component in the response variable seen in listing 5.7 (Inputs, outputs, States, Controls). For each of these components it would display the name of it, total vulnerabilities of all of its component-children (as an integer) and the overall severity of all the children. Each of these child-components it displays the associated products along with number of vulnerabilities and the severity too. An example of a of such an display is seen in figure 5.5


System threat list:

Component		Vulnerabilities	Severity
inputs		913	HIGH
Keyboard		3	LOW
Logitech R500 Firmware		1	LOW
Logitech K360 Firmware		2	LOW
Network Card		5	HIGH
TP-Link TL-SG108E Firmware 1.1.2		5	HIGH
Router		905	HIGH
Microsoft Windows 10 1511 64-bit		905	HIGH

Figure 5.5: Screenshot of vulnerabilities of the system

The pseudo-code for this is illustrated in listing 5.8. To see the corresponding sourcecode look at `line 15-68` in main.vue

```
start table
    for each main_component in response:
        * create mouseclick event that expands the main component
        for each component in components:
            > display component name
            > display component total vulnerabilities
            > display component overall severity
            * create mouseclick event that expands child-products list
            for each product:
                > display product title
                > display product total vulnerabilities
                > display product overall severity
                * create mouseclick event that open vulnerabilities window
            end inner loop
        end middle loop
    end outer loop
end table
```

Listing 5.8: main.vue - template-section. line 15-68

The user is building the model and adding components from the stencil into the canvas and linking them to one of the four main components. However these components are just empty containers with a name. For instance *Keyboard*. The component needs to have at least one child product attached. This is done by right clicking the component and choosing *Add child product*, which opens up the search window as seen in figure 5.6

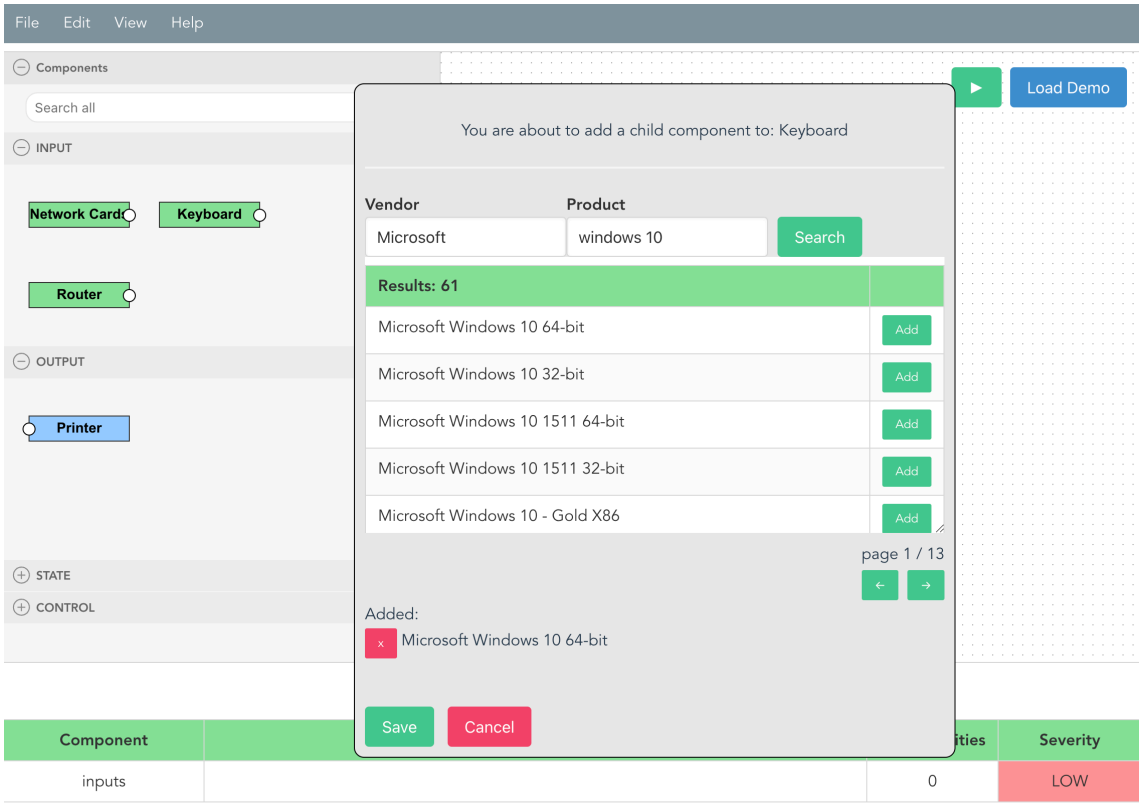


Figure 5.6: Screenshot of the search window

We import the component *SearchCVE*, as seen in listing 5.9 on `line 2`. When we import vue components we are able to pass along arguments, in this case to the search window, and return values. *:node_title* on `line 3` is the argument we pass in, in this case a title for the window. *@close=* at `line 4` and *@data* at `line 5` are listeners that listen for a specific return value. For instance if we close the window we return the value *Close* and invoke the function *toggleSearchWindow*, and if we pass any data from the window it will be passed to the function *add_child_nodes_to_graph*. The code for the SearchCVE component will be explained later.

```
1 <div v-if="displaySearchWindow" id="menu_search">
2     <SearchCVE
3         :node_title="focused_node_title"
4         @close="toggleSearchWindow"
5         @data="add_child_nodes_to_graph" />
6 </div>
```

Listing 5.9: main.vue - template-section. line 70-78

When right clicking the menu we also have the option to see what products are attached to the components. We pass along as argument the list of products, as seen in listing 5.10. The code for the DisplayNode component will be explained later.

```
1 <div v-if="displayNodeWindow" id="menu_node">
2     <DisplayNode
3         :title="focused_node_title"
4         :nodes="focused_node_children"
5         @close="toggleNodeWindow" />
6 </div>
```

Listing 5.10: main.vue - template-section. line 86-89

The right click menu is a vue component that takes as argument a title. The return values from clicking the menu triggers either a window where the user can search for products which can be added to the right-clicked compenent, or a window that displays the current content of the component. See listing 5.11.

```
1     <!-- display menu when right clicking nodes -->
2     <div v-if="displayMenu" id="menu_rcm" ref="menu_ref" style="left:99999px">
3         <RightClickMenu :rightClickMenuTitle="focused_node_title"
              @openSearchWindow="toggleSearchWindow"
4             @openNodeWindow="toggleNodeWindow" @openLoadModelsWindow="
                  toggleLoadModelWindow" />
5     </div>
```

Listing 5.11: main.vue - template-section. line 80-85

When the user clicks the Load Demo button, the window for loading graph models appears. If the window returns a model then the *loadModel*-method is invoked. The code for the DisplayLoadModels component will be explained later.

```
1 <div v-if="displayLoadModelsWindow" id="menu_load">
2     <DisplayLoadModels
3         :title="focused_node_title"
4         @close="toggleLoadWindow"
5         @model="loadModel" />
6 </div>
```

Listing 5.12: main.vue - template-section. line 91-94

When the user has runs the model and receives the vulnerabilities from the backend, it is displayed in the footer section. Because some products can contains up to several thousands vulnerabilities, some with a lot of information, it is hard to display it directly in the footer table. Thus, the user can click any product and a vulnerability window appears.

```
1 <div v-if="displayVulnerabilitiesWindow" id="menu_vuln">
2     <DisplayVulnerabilitiesWindow
3         :title="focused_product"
4         :nodes="focused_vulnerabilities"
5         @close="toggleVulnerabilityWindow"
6         @model="loadModel" />
7 </div>
```

Listing 5.13: main.vue - template-section. line 98-101

The following code explanation is a mixture of pseudo and real code, we will reference to the actual line numbers in listing 5.14. From the previous code explanation we have explained the different parts of the template section, now we dive into script section of the code. This is where we import the components used in the template section.

export default: the way we export variables, methods and such from the script-section is with *export default* as seen in 5.14 first line. On `line 2-9` we export and expose the imported vue components to such that the template can use them.

props: we define a prop-value n `line 10-11`, this is how Vue handles arguments to external components. E.g. we pass a title string to several of the components.

data(): this is where we return stored or computed data in variables which is reachable to both the script with: "this.\$variable name" and from the template section by referencing the variable name directly in vue syntax, e.g. " ".

async created(): as explained earlier with the vue lifecycle, we have the ability to tap into different stages of the vue component. In this stage we are calling different methods that can be computed before any DOM elements are ready. We load all the components from the database and create the graph and stencil object with their intial settings. Then we build the base stencil and model graph and setup any listeners to the graph, such as the logic and behaviour when users are interacting with the graph. Lastly we just initialise an empty vulnerability table in the footer of the website. We are define the created() stage to be asynchronously because reading in data from an external database can take some time.

mounted: when everything from the created() stage is ready and we reach the mounted stage where all DOM elements are created, we can setup the mouse listeners on the DOM. We use this mouse listener to set the coordinates `mousePosX` and `mousePosY` and in `data()`. These coordinates helps placing the right click menu where ever the user clicks the canvas.

methods: here we declare all of our methods. To access or call methods from one and another or access/set variables, we use `this.` + method or variable name.

- `openVulnerabilities(title, vulnerabilities)`: this methods is responsible for displaying the window with greater details on a specific products vulnerabilities.

  `loadModel(value)`: this method reads the return value from `DisplayLoadModels` and sets replaces the current graph state. Notice, right now we read only the first element `value[0].model` which always is *INPUT*. This is needs to be changed to iterate through all of the main components, when the system is capable of that.

- `determine_system_severity(system_severities)`: this is simply a helper function that takes in a list of severities and returns the most severe from the list: order: `'HIGH' > 'MEDIUM' > 'LOW'`

- `response_to_display()`: this methods is responsible for extracting the different elements of the json object that is received by the backend when a user has pressed the run button. We maintain a top level list to hold all system severities read from the object. Then, for each of the components (e.g. a 'Keyboard') in the object, we add them to the global list: component. For each of the components, we add their products (e.g. a driver for the keyboard) to the global list: products. Inside this for-loop we also count up the global vulnerability counter: total_vulnerabilities. This way its easy to display a total count in the footer of the web-page. For each of the product, we loop through all of its vulnerabilities and add each severity count to the top level list of severities. When all vulnerabilities are accounted for we use `determine_system_severity(system_severities)` to determine the highest graded severity. This is way the footer can display the overall severity to the system.

- `toggleMenu()`: this is just a method that flips the bit responsible for showing or hiding the window. Similar functionality exists for `toggleLoadWindow()`, `toggleVulnerabilityWindow()`, `toggleLoadModelWindow()`, `toggleSearchWindow()`. Considered they are doing the same logic, this can be refactored.

- `toggle(id), toggle2(id), toggle3(id)`: is helper function that makes it possible to extend or close the table rows in the footer vulnerability table. This is done by maintaining a list with current elements to show: `opened` With the help of the JavaScript method `splice`.
- `get_child_nodes()`: this is a helper function that read the current nodes (e.g. a driver) of the graph from a specific component (e.g. Keyboard). When finished reading it sets the global variable `focused_node_children` with this value.
- `add_child_nodes_to_graph(value)`: when a user right clicks on a component (e.g. Keyboard), opens the window for searching products, adds a product (e.g. a driver) and closes it, a value consisting a list of components is returned to this method. Then it iterates through the graph and updates the relevant component with the new set of products.
- `async requestVulnerabilitiesFromServer()`: this method is responsible for doing the rest call to the backend and sending along the current state of the model. In return it receives a json object with all the vulnerabilities of the system. All global variables that is related to the response is cleared. Then we call `response_to_display()` to display the new data.
- `get_components_from_database()`: this is a simple rest call to the backend that request the components from the database. It places the components in `items` which is read by the stencil to display the components.
- `create_graph()`: in this method we create and define the graph object from X6. The graph has some basic attributes it needs to know before it can be initialised: `container: this.$refs.model_ref`, is the pointer to the div-identifer in the template section. We also set the size of the graph along with some behavioral settings such as snap-radius which is when a user creates the link between a component and a main component (e.g. link between Keyboard and Input), then upon drag it instantly snaps to the main component. We also here set a `validateConnection` that makes sure that, for instance, a component from the Input category in the stencil is only allowed to be linked to the main component named Input.
- `create_stencil()`: in this method we create and define the stencil object from X6. It too, as with the graph object, needs some basic settings, such as a reference to the graph. This is needed in order to have the drag-n-drop behavior from stencil to canvas. It needs some size settings too. We have hard-coded the categories: Input, Output, State and Control and then reads in the respective components from the global variable `Items`. For each of the categories Input, Output, State and Control, we define the look and meta data. This metadata consists of shape, label, id, type, size, port settings (where do we drag the linking arrow from), colors, etc. Lastly we load all of the components into the stencil.
- `create_base_model()`: the base model consists of five nodes: Input, State, Control, Output and System. The define the look and metadata as well for these and add them to the graph.
- `setup_graph_listeners()`: Last thing we do is to create listeners on the graph. We want to know when a user is right clicking a component in the canvas such that a context-menu can appear. We also want to know when the user is hovering either a component (node) or the link between nodes (edge). When hovering we display an erase icon, and upon leaving the node or edge this icon disappears.

```
export default {
    components: {
        CVE,
        RightClickMenu,
        SearchCVE,
        DisplayNode,
        DisplayLoadModels,
        DisplayVulnerabilitiesWindow
    },
    props: {
        rightClickMenuTitle: String
    },
    data() {
        return {
            // graph, stencil,
            // booleans that determines if a window is open or closed,
            // title, id, children of focused product etc,
            // mouse position,
            // response from server,
            // placeholder lists
        }
    },
    mounted() {
        // setup mouse event listeners
    },
    async created() {
        // load components from database
        // create graph and stencil
        // build base model
        // setup graph listeners
        // setup empty vulnerability table

    },
    methods: {
        openVulnerabilities(title, vulnerabilities){
            // sets title and add vulnerabilities to focused product
            // displays vulnerability window
        },
        loadModel(value) {
            // set global graph from the loaded in model
        },
        determine_system_severity(system_severities) {
            // helper function to determine highest value in list
            // 'HIGH' > 'MEDIUM' > 'LOW'
        },
        response_to_display() {
            // dissect the response json object and places
            // all components, child components, vulnerabilites data
            // into the respective global variables

            Pseudocode:
            for each component in response:
                > push to global component list
                for each product in component:
                    > push to global product list
                    > products len. of vuln. is added to global count
                    for each vulnerability in product:
                        > save the vulnerabilys severity in a temp list
            calculate overall serverity from temp list
        },
        toggle_windows(){
```

```
            // flips the bool that shows/hide the window
        },
        toggle(id){
            // helper function to create the ability to expand/close
            // the footer vulnerability table's rows
        },
        add_child_nodes_to_graph(value) {
            // add products (value) to specific component in the graph
            // value is the return value from SeachCVE

            for each cell in graph:
                if node == focused component:
                    > products += value
        },
        async requestVulnerabilitiesFromServer() {
            // REST call to backend: POST the state of the model
            // gets the vulnerability model in return
            // clears all response variables in data()
        },
        async get_components_from_database() {
            // Rest call to backend: GETs list of components in return
        },
        create_graph() {
            // creates the X6 graph object with basic settings
        },
        create_stencil() {
            // creates the stencil object with basic settings and
            // loads in components from database
        },
        create_base_model() {
            // creates base canvas model:
            // (input, output, state, control, system)
        },
        setup_graph_listeners() {
            // sets up listeners in the canvas:
            // node:mousedown, node:contextmenu, node:mouseenter,
            // node:mouseleave, edge:mouseenter, edge:mouseleave
        },
    },
```

Listing 5.14: pseudo-code: main.vue - script-section

It is a bit trivial to go through all of the components imported to Main.vue because the basic methods of importing and passing arguments and return values is the same. Instead we will provide the pesudo-code with an explaination of each, and you as a reader can dig into the sourcecode if needed:

5.2.4 Frontend: Components/AddProduct.vue

This is a very simple component that is responsible for creating new products to the database. The user provides a name and a description via textfields and the category from a dropdown as seen in figure 5.7. method `saveProduct()` is invoked when the save-button is clicked. This makes the rest post request to the backend: `url + '/components'` along with the name, description and category.

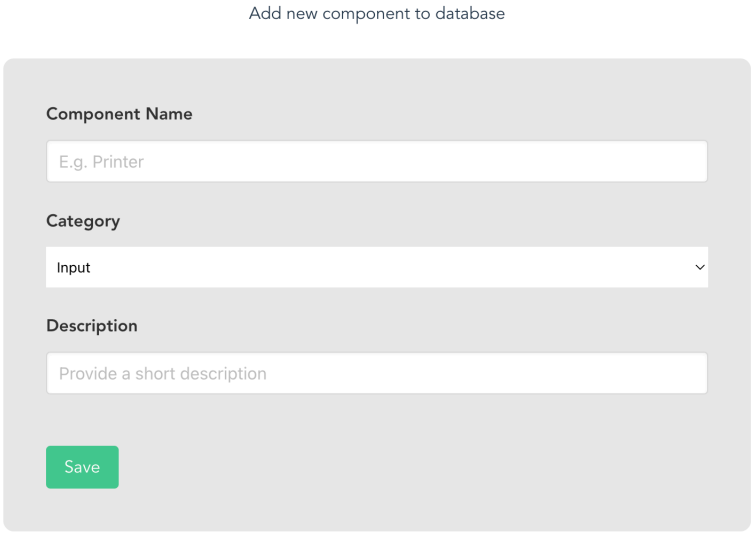


Figure 5.7: AddProduct.vue

5.2.5 Components/Database.vue

The database component displays a table with the name, description and category of all components in the database, and a button for deletion of specific components as seen in figure 5.8.

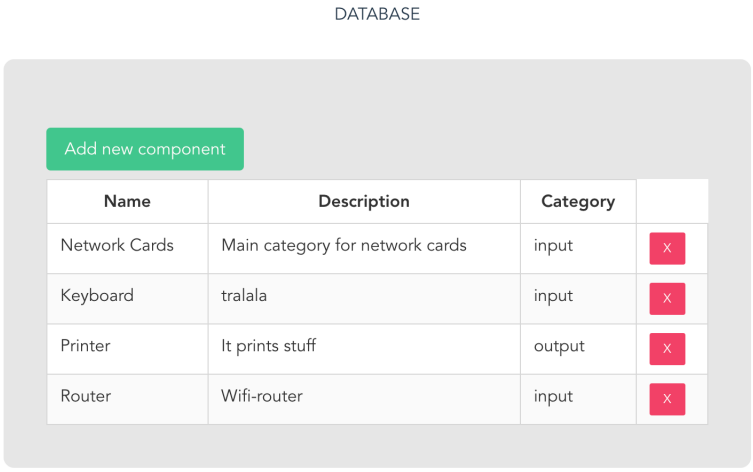


Figure 5.8: AddProduct.vue

The template part is illustrated with pseudo code in listing 5.15. On create-stage of the vue component cycle we call `getProducts()` which is the method which makes a rest get request to the backend to retrieve the components: `url + 'components'`. The return data is placed in the global variable `Items`. When delete-button is pressed, the `deleteComponent(id)` is invoked, making the rest call delete request: `url + '/delete/id'` deleting of a specific component by its id.

```
button: "Add new component"

for each product in products display:
    > name, description, category, delete button
```

Listing 5.15: Pseudo code: Database.vue - template-section

5.2.6 Components/DisplayNode.vue

The DisplayNode is the window that appears when users right click a component, e.g. Router, and inspects the comoponent. It displays in a table all associated products, as seen in figure 5.9, some firmware for a network card. Its possible to delete components from the table.

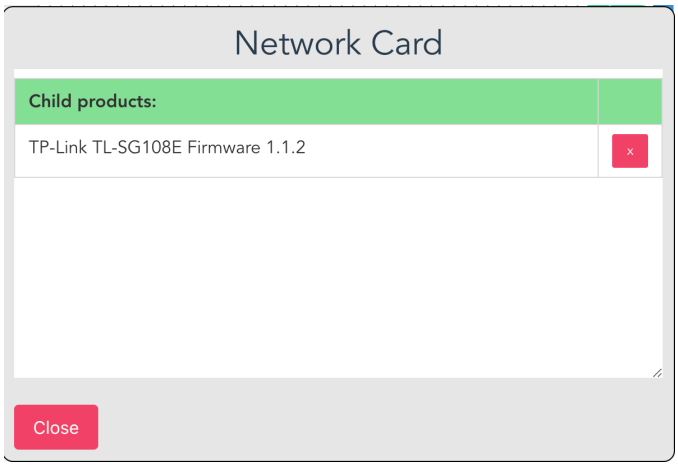


Figure 5.9: DisplayNode.vue

If we delete a component from DisplayNode, then we need to emit this back to the graph model in main.vue. This is done by removing the particular component by invoking `remove(index)`. This method simply removes the component from the `nodes`-list, as seen in listing 5.16.

```
1       remove(index) {
2          this.nodes.splice(index, 1);
3       },
```

Listing 5.16: Pseudo code: DisplayNode.vue - script section

5.2.7 Components/DisplayVulnerabilities.vue

This is a very simple component that simply takes in as argument all vulnerabilities associated with a product, as seen in figure 5.10, displaying two vulnerabilities for a keyboard firmware. There is no further functionality other than extended information on the vulnerabilities and the ability to close the window.

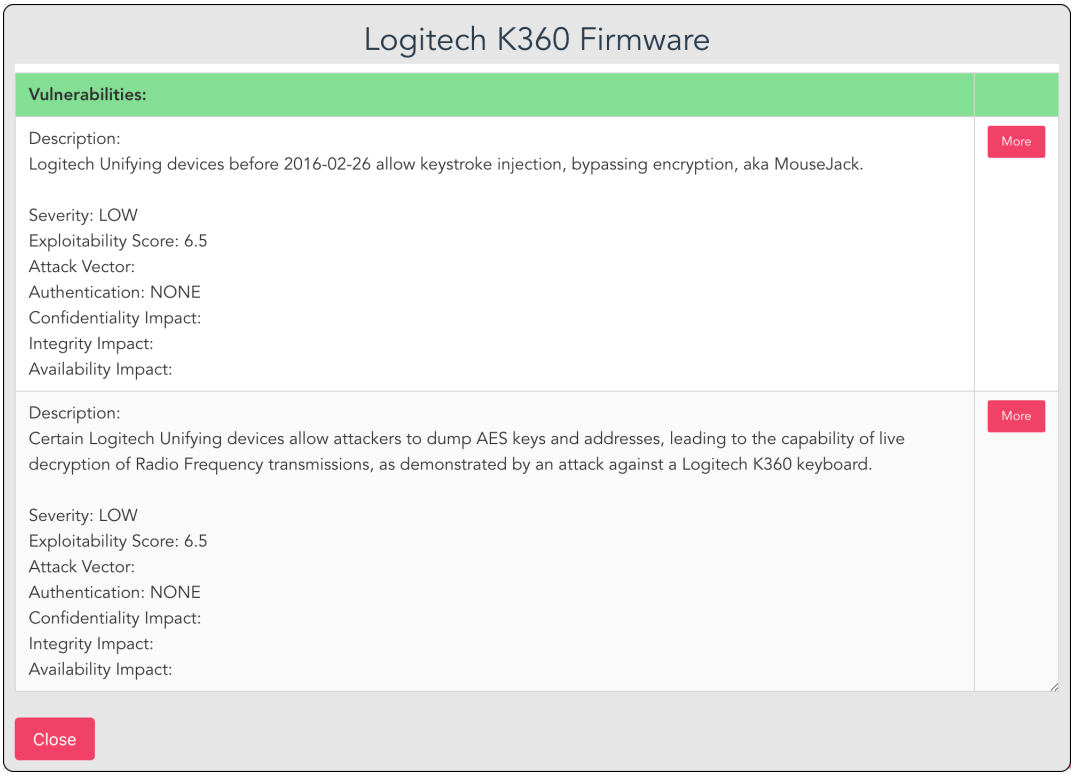


Figure 5.10: DisplayNode.vue

5.2.8 Components/RightClickMenu.vue

This is the component that displays the right click menu when users right click in the canvas. It is entirely created with simple HTML and CSS to have the appearence of a menu, see figure 5.11.

Right now it has to links: Open the window to inspect a component or the window for adding products to the a component. This is a simply boolean value that is emitted back to main.vue to show one of the windows.

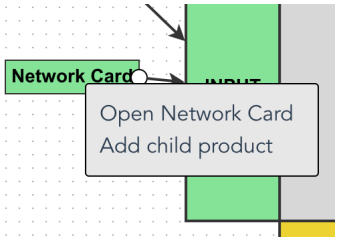


Figure 5.11: DisplayNode.vue

5.2.9 Components/SearchCVE.vue

This is where users search for products for a given component. E.g. find a specific driver and firmware for a keyboard. The figure 5.6 is an example a search on vendor: Microsoft with the product: windows 10. This yields a large amount of results. So there was a need for pagination, such that the user can browse through the results as pages. Listing 5.17 is pseudo-code for pagination. We have a global list `myList` and global variable that contains the current 5 results for a given page. `page` is to control which page we are on. `line 4` is how we do the pagination, with the JavaScript method slice, which returns a

portion of an array `slice(start, end)`. To change page we simply invoke `prev` or `next` method.

```
paginator(){
for response.length:
    > add response title and id to myList

result = myList.slice((page - 1) * 5, page * 5)
}
prev(){
    if page > 1:
        > page--
        > paginator()
}
next(){
    if page < response.length / 5:
        > page++
        > paginator()
}
```

Listing 5.17: Pseudo code: SearchCVE.vue - Paginator()

In order to search, a user can type in a vendor and/or a product. This invokes the `searchProduct()` method, the as illustrated with pseudo-code in listing 5.19. We make sure that the user cannot search with empty string. If a users search word contains white spaces we remove them an replaces them with an underscore, e.g. `'windows 10'` becomes `'windows_10'`. This is the syntax accepted from the NVD api, which will be explained in greater depth in the section for the backend system. When products are added they are emitted back the main.vue with `this.$emit("data", this.picked_products);`

```
if search fields are not empty:
    > replace white space with an underscore
    > make a rest post request to backend
    if no result:
        > display message "no results"
    else:
        > update global variale with response data
        > paginator()
```

Listing 5.18: Pseudo code: SearchCVE.vue - SearchProduct()

Listing 5.19: Pseudo code: SearchCVE.vue - SearchProduct()

5.2.10 Components/LoadModels.vue

Right now this component only has one hardcoded model of a system places directly in a variable. When the user clicks `load`, as seen in figure 5.12, the model is emitted back to main.vue with: `this.$emit("model", this.models);`.

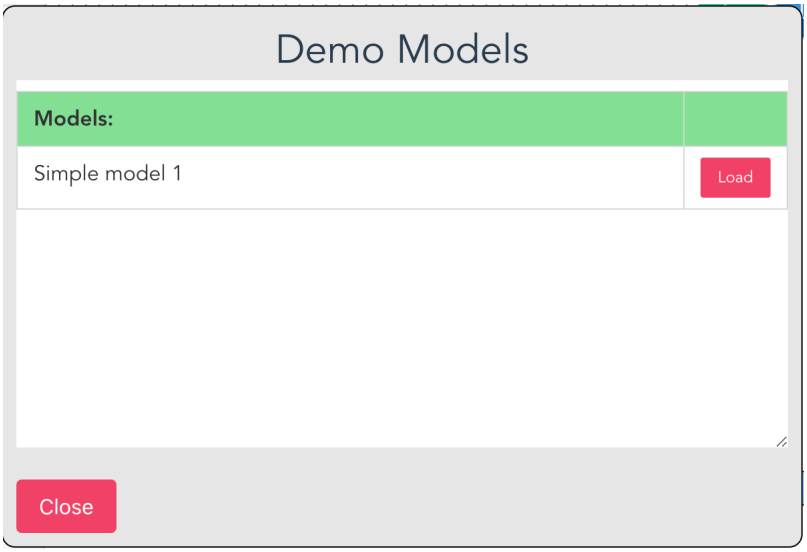


Figure 5.12: LoadModels.vue

5.3 Backend

5.3.1 The NodeJS & ExpressJS framework

Before we get into the details of the backend codebase, we will provide an overview of the NodeJS framework along with ExpressJS and Nodemon, which together makes it very easy for developers to create an backend server that can handle multiple connection concurrently[38].

NodeJS is an open source, cross-platform JavaScript framework designed to build server-side network applications. ExpressJS is a framework build on top of NodeJS which simplifies server-side routing, handling of requests and responses (middleware) and debugging[39]. Nodemon is a useful tool that automatically restarts the NodeJS application when a change is detected in the sourcecode under development. This saves alot of time during development.

Installation & run-guide for the backend part is found outside this report, see: `Installation-guide.pdf`.

5.3.2 Backend Overview

The backend server consists of an entry point, the `index.js`-file, which is where we define and set up our server with express, routing, middleware and cors policies. When the frontend application is making a REST request to the backend, it is recieved by `routes.js` which, depending on the request, will execute the respective backend method and create an response. As seen in figure 5.13, we have a overview of the different REST calls to the backend going in the blue box `routes.js` and the corresponding `controllers` methods invoked. The backend is separated into controllers and models, the idea behind this is to improve readability and maintainability be letting the controllers be responsible of handling incoming requests from the backend. We define the logic for handling specific routes and actions here, and let the models represent the business logic of the backend, such as database queries and rest call to external APIs. We also define a config file for the database `config\database.js` that is responsible for setting up the database connection.

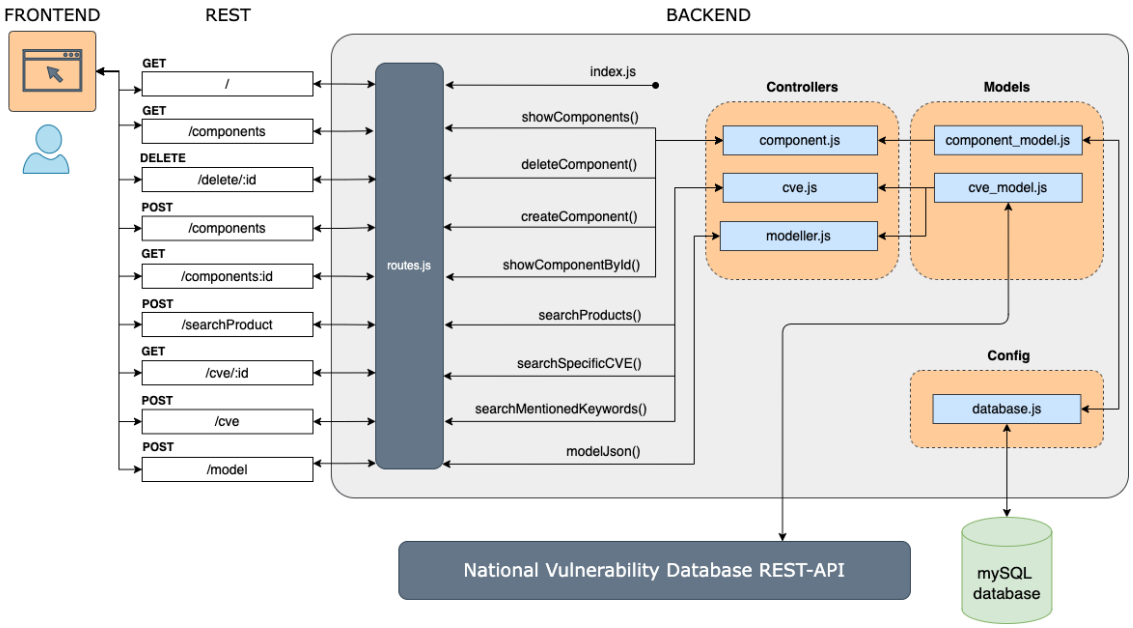


Figure 5.13: System architecture with focus on the backend server

We import express, CORS and router in this file. Most browsers will block request made to other domains than what the user is visiting, that is why CORS is needed, short for Cross-origin resource sharing**??**. It allows us to define an origin address which is accepted by the server. In listing 5.21 we are using the `localhost` at port `8080`, this is for local testing, and can be changed to any live address where a frontend is deployed. Next, the express options are set up, we add the cors policies, json and the router, and lastly we make the server listen on the port defined on `line 1`. We are using the `dotenv` package, which allows us to define environmental settings outside the source-code. This is good if you have a live production and do not want to have, e.g. password and username for database connection stored in the source-code. For local testing this environment file is located at root level named `.env`. Its common practice with git-repositories to include this file in gitignore.

5.3.3 index.js

```
1  const PORT = process.env.PORT || 3030;
2
3  // https://dtu-tir.netlify.app
4  var corsOption = {
5     origin:
6        "http://localhost:8080",
7     optionSuccessStatus: 200
8  };
9
10 const app = express();
11 app.options("/*", function (req, res, next) {
12    // set access control options
13    res.sendStatus(200);
14 });
15
16 app.use(cors(corsOption));
17 app.use(express.json());
18 app.use(Router);
19
20 app.get('/', (req, res) => res.send("Server is up and running!"));
```

```
app.listen(PORT, () => {
   console.log("Server is running ..");
});
```

Listing 5.20: index.js

Routes is a simply file that is responsible for invoking a specific method in components based on the rest request. As seen in 5.21, the rest calls and their respective methods are matching those from picture 5.13.

5.3.4 Routes/routes.js

```
// handles the rest endpoints from the frontend
const router = express.Router();
router.get("/");
router.get("/components", showComponents);
router.get("/components:id", showComponentById);
router.post("/components", createComponent);
router.post("/model", modelJson);
router.delete("/delete/:id", deleteComponent);
router.get("/cve/:id", searchSpecificCVE);
router.post("/cve", searchCVEMentioningKeywords);
router.post("/searchProduct", searchProducts);
```

Listing 5.21: routes/routes.js

5.3.5 Controllers/compenent.js

The `components.js` file has four methods, two are shown in listing 5.22 because of similarity. The first method `showComponents` is taking in as argument the rest request and invoke `getComponents`. In return it receives an response, either an error message or a json object with all the components in the database (e.g. Keyboard, Network Card, etc.), this is returned back to the frontend. The second method is similar with the exception it has a id parameter which is used to search a specific component. The two last methods now shown are `createComponent`, which is responsible for creating a new component, and `deleteComponent` which is responsible for deleting a component in the database.

```
// returns all components from the database (e.g. 'router', 'keyboard', '
    printer')
export const showComponents = (req, res) => {
   getComponents((err, results) => {
      if (err) {
         res.send(err);
      } else {
         res.json(results);
      }
   });
};

// returns a single component from the database based on its id
export const showComponentById = (req, res) => {
   getComponentById(req.params.id, (err, results) => {
      if (err) {
         res.send(err);
      } else {
         res.json(results);
      }
   });
};
.
.
```

```
.
```

Listing 5.22: controllers/component.js

5.3.6 Models/component_model.js

Again, two of the four methods are shown in listing 5.23. This is where do actual queries to the database is done. The first method is simply returning all components from the database with the SQL-query: `SELECT * FROM components`, and the second method is returning a single component based on an id with:
`SELECT * FROM PRODUCT WHERE component_id = ?`

```
export const getComponents = (result) => {
   db.query("SELECT * FROM components", (err, results) => {
      if (err) {
         console.log(err);
         result(err, null);
      } else {
         result(null, results);
      }
   });
};

export const getComponentById = (id, result) => {
   db.query("SELECT * FROM PRODUCT WHERE component_id = ?", [id], (err,
       results) => {
      if (err) {
         console.log(err);
         result(err, null);
      } else {
         result(null, results[0]);
      }
   });
};
```

Listing 5.23: models/components_model.js

5.3.7 Controllers/cve.js

This file handles the request on searching for information on either a specifiv CVE based on a CVE-ID, and returns either an error message or a json object with the information on the CVE, or a broader seach on either a vendor, a product or both. Here the json return object is a list of all the products matching the search terms.

```
// returns a json-object of an specific CVE based on its CVE-id
export const searchSpecificCVE = async (req, res) => {
   const id = req.params.id;

   searchByID(id, (err, results) => {
      if (err) {
         res.send(err);
      } else {
         res.json(results);
      }
   });
};

/*
   return all products matching the terms based on:
   vendor name  (e.g. 'Microsoft),
   product name (e.g. 'windows 10')
*/
```

```
export const searchProducts = async (req, res) => {
   const vendor = req.body.vendor;
   const product = req.body.product;

   searchProductByVendorAndProductNames([vendor, product], (err, results) => {
      if (err) {
         res.send(err);
      } else {
         res.json(results);
      }
   });
};
```

Listing 5.24: controllers/cve.js

5.3.8 Controllers/model.js

This file has two main methods: `modelJson`, seen in listing 5.25, which is responsible for taking in a json object containing the state of users graph model, from the frontend, and returning all vulnerabilites back to the frontend. This is done by calling the second method: `getAllVulnerabilitiesInJsonFormat`, which is in pseudo-code in the code-listing. This method creates and returns a nicely formatted json object of all the vulnerabilites of all the products (e.g. a router) of the main component `input`. The way we display any output in the frontend is by tables that is looping through indexes, this is why we increment a count when looping through each of the components and its children.

- We run through each of the sub-components in the graph. Example of sub-components could be a `Router`, `Keyboard`, etc. Right now, we are only addressing sub-components for `input` and not, `output`, `state`, or `control`. These main-nodes are needed to be included, if this project is to be further developed on.
- Then we invoke the `getAllVulnerabilitiesFromCPEName` method on all of the child-products of the the current sub-component. Example of child-products could be `Drivers`, `Firmware`, `Operating systems`, etc. This method return all vulnerabilities associated with each child product. For each of the child products, we keep track on the number of vulnerabilities it has. We also assign an index to the child product with `product_count++`. The reason for this is to make it easier for the frontend to loop through the json object by indexes.
- For each of the vulnerability found in each of the child products, we determine the highest ranking severity for the current child product, and adds this severity to a global list and a local one. That way we have an easy way for the frontend to read; what is the severity for a given child product, and what is the overall severity of the system.
- Then we create a list with all the child-products, including local severities, indexes, names, titles, vulnerabilities.
- When all child products are created, we create the sub-component wich includes the highest ranking severity, count of vulnerabilities, an index, the name (in this case its only `input`, and lastly adding the list of child-products.
- Now the json object is complete and is returned to the frontend.

```
export const modelJson = async (req, res) => {
   const graph = req.body.graph;
   await getAllVulnerabilitiesInJsonFormat(graph, (err, results) => {
      if (err) {
```

```
          res.send(err);
       } else {
          res.json(results);
       }
    });
}
.
.
.
getAllVulnerabilitiesInJsonFormat(graph, result){
    for all cells in graph:
        for all sub-components:
            if main_node = "input":
                for all children of sub-component:
                    > getAllVulnerabilitiesFromCPEName(CPEname)
                    > count up number of sub-component vulnerabilites
                    > increment count for current product
                    for all vulnerabilites in sub-component:
                        > add severity to global list
                        > add severity to local list
                    > determineSystemSeverity(local severities)
                    > add highest local severity, id, name, title, vuln., etc
                      ., to child-node
            > highest severity = determineSystemSeverity(global list)
            > add highest severity, vulnerabilities count, index , comp. name,
                products
}
```

Listing 5.25: controllers/model.js

5.3.9 Models/cve_model.js

This is the file where we do the actual REST request to the NVD database API. We will explain two of the methods we are using, the first one is `getAllVulnerabilitiesFromCPEName` we invoke in 5.25. This method is shown in listing 5.26. This is a simple REST request that, based on a CPE name return all associated vulnerabilities. In our code we only include metadata such as, descriptio, metrics, weaknesses, configuration and references. There is much more information that can be included, if needed. The second method is `searchProductByVendorAndProductNames` which takes in as arguement a vendor name and/or a product name. E.g. vendor: `Microsoft`, product `windows 10`. This method return all products that matches these terms.

```
export const getAllVulnerabilitiesFromCPEName = async (cpeName, result) => {
     try {
        const response = await axios.get(`https://services.nvd.nist.gov/rest/
            json/cves/2.0?cpeName=${cpeName}`);
        const vulnerabilities_data = response.data.vulnerabilities;
        const len = vulnerabilities_data.length;

        let vulnerabilities = [];
        for (let i = 0; i < len; i++){
         let vulnerability = {
            "description" : response.data.vulnerabilities[i].cve.descriptions,
            "metrics": response.data.vulnerabilities[i].cve.metrics,
            "weaknesses" : response.data.vulnerabilities[i].cve.weaknesses,
            "configurations": response.data.vulnerabilities[i].cve.
                configurations,
            "references" : response.data.vulnerabilities[0].cve.references
         };
         vulnerabilities.push(vulnerability);
        }
```

```
        result(null, vulnerabilities);
    } catch (err) {
        result(err, null);
    }
};

export const searchProductByVendorAndProductNames = async (keywords, result)
    => {
 try {
    const response = await axios.get(`https://services.nvd.nist.gov/rest/
        json/cpes/2.0?cpeMatchString=cpe:2.3:*:${keywords[0]}:${keywords
        [1]}`);
    const products = response.data.products;
    const final_response = [];

    for (let i = 0; i < products.length; i++) {
        const cpeName = products[i].cpe.cpeName;
        const cpeNameId = products[i].cpe.cpeNameId;
        const cpeTitle = products[i].cpe.titles[0].title;

        const product = {
            cpeName: cpeName,
            cpeNameId: cpeNameId,
            cpeTitle: cpeTitle
        };
        final_response.push(product);
    }
    result(null, final_response);
 } catch (err) {
    result(err, null);
 }
};
```

Listing 5.26: models/cve_model.js

5.3.10 Config/database.js

This is the configuration file to establish a connection of the database. As seen in listing 5.27 we have initialised a local configation with a connection limit of 1000, the host, username and password of the database and the database name. In comments, we have shown how to setup a connection with environmental settings.

Installation & run-guide for the backend part is found outside this report, see: `Installation-guide.pdf`. Here is also a guide on how to setup the database settings for a live server.

```
// local config
var db_config = {
    connectionLimit : 1000,
    host: "localhost",
    user: "root",
    password: "",
    database: "mydb"
};

// SERVER config
/*
var db_config = {
  connectionLimit : 1000,
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
```

```
  database: process.env.DB_NAME
};
*/

var connection = mysql.createPool(db_config);
export default connection;
```

Listing 5.27: config/database.js

6 Proof of work

Unfortunately, we did not manage to get more than five people to answer the questionnaire and test the system. Based on the people who participated, they all have very different backgrounds. The questionnaire starts by letting the participants try out the program without any prior knowledge to test whether they can understand what it is all about. They are then asked a series of questions about their experience. After this, we explain to them what the idea of the system is and how it should be used, after which they are allowed to answer a series of questions again. All answers can be seen in Appendix.

7 Further work

7.1 Immediate practical additions

- The X6 framework supports built-in functionality for `copy`, `paste`, `cut`, `undo`, `redo`, and `zooming` in the graph. This would provide the user for a much better experience.
- Support for saving a model to and reading from local storage can be achieved either in the frontend or backend. For frontend Vue support this with `localStorage`[40]. For the backend this can be archieved with `fs`[41].
- The graph and its component could be optimized on several points: right now its possible to move the main components, such as `input`, `output`, `state`, `control` and even `system` around. This is a bit confusing. These should either be lock in place or grouped together such that they could be moved as one object. This is possible to do within X6.
- From the user feedback it is clear that the testers do not automatically read the help-page, instead they jump right in to try the model out. For the most testers this results in them not finding out all the functionality. As suggested verbally from one of the testers, a mandatory tutorial could be introduced when a user is first presented with the website. A popup window that introduces the user to the basic functionality of the site.
- The `System threat list` in the footer of the website could be improved upon. Right now it displays all the threats in a table with severity for `High`, `Medium`, `Low`. These are all in red. A nicer way would to make the respectively red, yellow and green.
- When clicking a product of a component in the threat list, the user is shown a window with all vulnerabilities associated. A lot more information than shown is actual pass along. A `More` button is added but not used. This could be used for displaying additional data on a particular vulnerability.
- For the X6 Graph we have only designed the behaviour and look of input and output components. For input the components are green and the port is placed on the right side (the white circle where user extracts the arrow). For output the color is blue and the port is placed on the left side. This needs to be done for both state and control as well.
- Right now the `Load Demo` is just a hard-coded model of a template that is loaded into the graph. This could be changed to a functionality that reads a list of templates from the backend. Possible let the users be able to save their own templates to the backend, which can be used by other users.
- When users click the send button it becomes inactive untill the server response with an answer. Further work in this process could be some notification on the status. This could be an alert box that appears for a few seconds with ”Now processing...”, ”Finished processing!” and ”Some error occurred, try again”. There exists several packages for Vue that supports this.

7.2 Extend on the input

- When users are building their model and adding products to components they are doing so by searching on a vendor-name and/or a product name. This is a fairly

broad search than often yields many results. The CVE API supports rather complex search abilities which our current solution in the backend and frontend could adapt to. These search queries are described in section 3.1.

7.3 Fixing current errors

This project has been tested locally on both Ubuntu, Kali Linux and MacOS and has no errors when runned locally. However a crucial error is happening in the backend when the project is moved to a live server.

The usecase: A user builds his model, lets say five different components are added with multiple products attached. He clicks the send-button and receives all vulnerabilities associated with his system. Now, if the user changes anything in the model, adds or deletes components or products, and clicks the send-button, then the live server response with the error:

```
[ERR_HTTP_HEADERS_SENT] Cannot set headers after they are sent to the client.
```

This is an error that errors in NodeJS, when dealing with asynchronous functions, where locally the framework handles request and responses so fast that its not a problem, but on a live network the a function can be finished executing but some asynchronous methods has yet to be finished. This results in a behavior where the function responsible for responding data to the frontend is no longer in use, but some asynchrounous methods are still trying to use it. Ultimately this lead to server crashes, even though we catch any errors with proper try-catch statements. A hacke solution to this is simply forcing a return statement into the function that is responsible for sending data back to the frontend. A return statement that is fired immediately when the function has sent its load the first time. However, this creates another problem where the frontend is not receiving all of its data. Thus a user has to resubmit the model multiple times until it do. Here are two sources that explains the problem more clearly with possible solutions[42] and [43].

This is clearly a crucial error that needs to be fixed at some point, if further development of this project is to be done.

7.4 Moving on to output, state and control

Output is the logical next step to look into. This is the area that reminds the most of what has already been implemented. Similar functionality should be implemented with consideration for the interplay and results from input and output, as discussed in section 4.1. Next would to research in depth the area of state and control in regards to systems, and from this begin the steps toward implementing the basics for the two.

8 Conclusion

The idea came from Christian D. Jensen, who proposed this project, which was to investigate whether there is a basis for building a system aimed at ordinary users who do not have the technical skills that IT specialists have nor the money to buy services from specialists, and then building a graphical software solution.

This thesis has investigated the current scene of threat modeling systems, the modern approach with different frameworks and tools, that are used by private individuals and companies. This thesis also covered different developer tools for building visual graph and different public databases for searching on known vulnerabilities.

A software solution has been realised based on the this research and the agreed upon goals to develop the initial steps towards a complete system, where non-technical users of the system are able to graphically model a system of their choice in a easy and intuitive way by defining the input, output, state and control of the system. The first steps towards such a solution has been implemented; a browser-based frontend system which the users interact with and defines the input to their system, and a backend server that handles databases, external APIs and all the logic. The system is capable of providing vulnerability information based on the users model in a nice manner where the user can see which components of their system has vulnerabilities and read the associated metadata.

What has not been realised, is output, control and state. These are beyond the scope of this thesis and ground for further work along with a critical error for using the system in a live production. The theory on these subjects has been covered in this thesis though.

Based on the research, user response and knowledge gained from implementing the first steps of the system, we conclude that there is basic for further research and development towards a complete system. However, though this new approach to threat modeling, we suspect that such a system might become a bit too complex to target the ordinary and non-technical people. If the focus changed from this target group, and a more broad group of people with a bit more technical knowledge, it might be of interest still.

Bibliography

[1] History Computer Staff. *The First Computer Virus of Bob Thomas*. URL: https://history-computer.com/the-first-computer-virus-of-bob-thomas/.

[2] Darpa.mil. *ARPANET*. URL: https://www.darpa.mil/about-us/timeline/arpanet.

[3] Wikipedia. *Creeper and Reaper*. URL: https://en.wikipedia.org/wiki/Creeper_and_Reaper.

[4] MITRE. *Common Vulnerabilities and Exposures. CVE*. URL: https://www.cve.org/About/Overview.

[5] National Vulnerability Database. *NVD*. URL: https://nvd.nist.gov/.

[6] Coordination Center Vulnerability Notes Database. *CERT/CC*. URL: https://www.kb.cert.org/vuls/.

[7] The National Institute of Standards and Technology. *NIST*. URL: https://www.nist.gov/.

[8] U.S. Department of Commerce. *USDC*. URL: https://www.commerce.gov/.

[9] First. *FIRST About*. URL: https://www.first.org/about/mission.

[10] IBM. *Advantages of Java*. URL: https://www.ibm.com/docs/en/aix/7.1?topic=monitoring-advantages-java.

[11] INFIRAISE Dhruv Kayasth. *What Is The Best Java Framework For Desktop Applications?, aug, 03, 2022*. URL: https://www.infiraise.com/what-is-the-best-java-framework-for-desktop-applications/.

[12] David Flanagan. *Java Foundation Classes in a nutshell*. O'Reilly.

[13] Oracle. *About the JFC and Swing*. URL: https://docs.oracle.com/javase/tutorial/uiswing/start/about.html.

[14] Oracle. *Compiling and Running Swing Programs*. URL: https://docs.oracle.com/javase/tutorial/uiswing/start/compile.html.

[15] Oracle. *JavaFX Frequently Asked Questions*. URL: https://www.oracle.com/java/technologies/javafx/faq-javafx.html.

[16] Oracle. *Getting Started with JavaFX*. URL: https://openjfx.io/openjfx-docs/#introduction.

[17] Northwoods Software. *About Northwoods*. URL: https://nwoods.com/about.html.

[18] Northwoods Software. *GoJS minimalistic example*. URL: https://gojs.net/latest/samples/minimal.html.

[19] Northwoods Software. *ntroduction to GoJS Diagramming Components*. URL: https://gojs.net/latest/intro/.

[20] Northwoods Software. *Download link for GoJS library*. URL: https://gojs.net/latest/download.html.

[21] NPM. *Npm packet manager*. URL: https://docs.npmjs.com/.

[22] AntV Group. *X6 - Documentation for nodes*. URL: https://x6.antv.antgroup.com/tutorial/basic/node.

[23] AntV Group. *X6 - Documentation for edges*. URL: https://x6.antv.antgroup.com/tutorial/basic/edge.

[24] AntV Group. *X6 - Documentation for events*. URL: https://x6.antv.antgroup.com/tutorial/basic/events.

[25] AntV Group. *X6 - Documentation for data*. URL: https://x6.antv.antgroup.com/tutorial/basic/serialization.

[26] OWASP.ORG. *OWASP*. URL: https://owasp.org.

[27] Threatdragon.com/. *Threat Dragon*. URL: https://www.threatdragon.com/.

[28] See complete author list: threatmodelingmanifesto.org/#about. *Threat Modeling Manifesto*. URL: https://www.threatmodelingmanifesto.org.
[29] *The CORAS Method*. URL: https://coras.tools/.
[30] Ketil Stølen Mass Soldal Lund Bjørnar Solhaug. *Model-Driven Risk Analysis: The CORAS Approach*. URL: https://www.amazon.com/Model-Driven-Risk-Analysis-CORAS-Approach-ebook/dp/B00F75RASI.
[31] *SeaSponge. Joel Kuntz, Sarah MacDonald, Mathew Kallada, Glavin Wiechert*. URL: https://mozilla.github.io/seaspong.
[32] Mozilla.org. *Winter of Security 2014*. URL: https://wiki.mozilla.org/Security/Automation/WinterOfSecurity2014.
[33] Microsoft. *Micosoft TM Tool*. URL: https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool.
[34] Computer Hope. *Input definition*. URL: https://www.computerhope.com/jargon/i/input.htm.
[35] Computer Hope. *Output definition*. URL: https://www.computerhope.com/jargon/o/output.htm.
[36] VueJS. *Frequently Asked Questions*. URL: https://vuejs.org/about/faq.html.
[37] vuejs.org. *Vue3 component life-cycle*. URL: https://vuejs.org/api/options-lifecycle.html.
[38] OpenJS Foundation. *About Node.js*. URL: https://nodejs.org/en/about/.
[39] Besant Technologies. *What is Express.js?* URL: https://www.besanttechnologies.com/what-is-expressjs.
[40] Graham Morby. *Persist Data With Vue3*. URL: https://dev.to/grahammorby/persist-data-with-vue-3-38pc.
[41] NodeJs Team. *Writing files with Node.js*. URL: https://nodejs.dev/en/learn/writing-files-with-nodejs/.
[42] Gajus Kuizinas. *Handling unhandles promise rejections in async functions*. URL: https://gajus.medium.com/handling-unhandled-promise-rejections-in-async-functions-35acfd1f2f57.
[43] Prosper Opara. *Understanding Node Error [ERR$_HTTP_HEADERS_SENT$]*. URL: https://www.codementor.io/@oparaprosper79/understanding-node-error-err_http_headers_sent-117mpk82z8.

A Appendix

Response json-object from server


```
{
 "id": 1,
 "name": "inputs",
 "components": [
  {
   "severity": "MEDIUM",
   "vulnerabilities": 1,
   "id": 1,
   "component": "Keyboard",
   "products": [
    {
     "severity": "MEDIUM",
     "id": 1,
     "name": "cpe:2.3:h:logitech:cordless_freedom_itouch_keyboard
         :-:*:*:*:*:*:*:*",
     "title": "Logitech Cordless Freedom iTouch Keyboard",
     "vulnerabilities": [
      {
       "description": [
        {
         "lang": "en",
         "value": "Logitech iTouch keyboards allows attackers with physical
             access to the system to bypass the screen locking function and
             execute user-defined commands that have been assigned to a button
             ."
        }
       ],
       "metrics": {
        "cvssMetricV2": [
         {
          "source": "nvd@nist.gov",
          "type": "Primary",
          "cvssData": {
           "version": "2.0",
           "vectorString": "AV:L/AC:L/Au:N/C:P/I:P/A:P",
           "accessVector": "LOCAL",
           "accessComplexity": "LOW",
           "authentication": "NONE",
           "confidentialityImpact": "PARTIAL",
           "integrityImpact": "PARTIAL",
           "availabilityImpact": "PARTIAL",
           "baseScore": 4.6
          },
          "baseSeverity": "MEDIUM",
          "exploitabilityScore": 3.9,
          "impactScore": 6.4,
          "acInsufInfo": false,
          "obtainAllPrivilege": false,
          "obtainUserPrivilege": false,
          "obtainOtherPrivilege": false,
          "userInteractionRequired": false
         }
        ]
       },
       "weaknesses": [
```

```
        {
         "source": "nvd@nist.gov",
         "type": "Primary",
         "description": [
          {
           "lang": "en",
           "value": "NVD-CWE-Other"
          }
         ]
        }
       ],
       "configurations": [
        {
         "nodes": [
          {
           "operator": "OR",
           "negate": false,
           "cpeMatch": [
            {
             "vulnerable": true,
             "criteria": "cpe:2.3:h:logitech:cordless_freedom_itouch_keyboard
                 :*:*:*:*:*:*:*:*",
             "matchCriteriaId": "30DA30B5-BE51-49C7-BA8A-EC8DB05CDCBF"
            },
            {
             "vulnerable": true,
             "criteria": "cpe:2.3:h:logitech:cordless_itouch_keyboard
                 :*:*:*:*:*:*:*:*",
             "matchCriteriaId": "8EE6D6C5-3672-4F90-86FB-9A306CA8B590"
            },
            {
             "vulnerable": true,
             "criteria": "cpe:2.3:h:logitech:itouch_keyboard:*:*:*:*:*:*:*:*",
             "matchCriteriaId": "001799FF-4C8E-4401-AB70-CB50F4DD8F72"
            }
           ]
          }
         ]
        }
       ],
       "references": [
        {
         "url": "http://www.securityfocus.com/bid/4662",
         "source": "cve@mitre.org"
        },
        {
         "url": "https://exchange.xforce.ibmcloud.com/vulnerabilities/8994",
         "source": "cve@mitre.org"
        }
       ]
      }
     ]
    }
   ]
  }
 ]
}
```

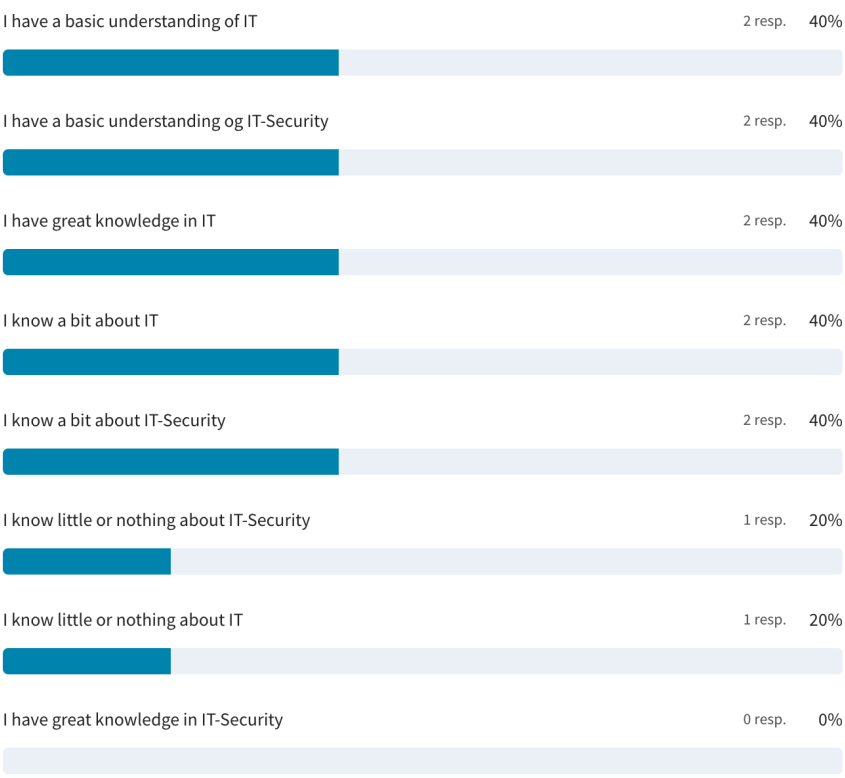

Listing A.1: Response JSON-object

Threat Identification & Response

5 responses

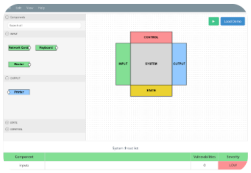
How would you describe yourself?

5 out of 5 answered



What are your profession and/or education?

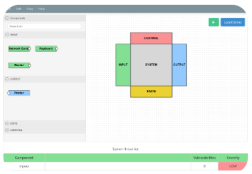
- theology
- Financial controller
- Economics Student
- Designer cand des
- network technician



This was the first thing you saw entering the website.

5 out of 5 answered

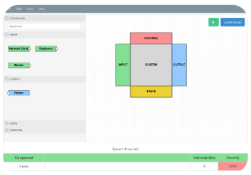
- It reminded me of a computers systems
- What is this :)
- At jeg ikke ved hvad der vises på skærmen, at man kan rykke rundt på blokkene, at der ikke umiddelbart skete noget, når jeg trykkede ”Load Demo” eller pilen ved siden af.
- That the colors match the boxes of the system. That its a plug-in function, where colors guide the interaction and function.
- what should I use it for?



What where your first actions?

5 out of 5 answered

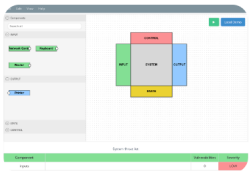
- I pulled the small cards(router, printer etc.) unto the component i thought they would fit under. So keyboard over to ”control”, networkd cards to ”state”, Router to ”input” and printer to ”output”. After that i did the same thing but by color instead, so all the green cards to input and the blue card to output.
- I were clicking several buttons
- Jeg begyndte at flytte rundt på boksene og trykkede på ”Load Demo” samt pilen ved siden af
- First i tried to drag the colored boxes from the left onto the system colored boxes to the right to match colors. Then i tried to press play, but nothing happened. Then i figured out that I could take the system colored boxes apart from the grey system box and wondered what the function of this could be, other than like a rough starter template, I could work from if i had the IT knowledge to support the usage of the program.
- was trying to figure out how to connect the boxes



What do you think it's about?

5 out of 5 answered

- I think they represent a computer, or perhaps a software system.
- Computer or server
- Noget med modtagelse af inputs (keyboard, Router, Network) og hvordan der sker en proces i systemet, som leder til output.
- The input/ output I think is like the data flow from start to end. Like the keyboard functions as an input, the printer is the last state of the output. They are placed oposite and represents a connection. The input section is green, witch is the color used as an information of ”go”, ”on” or another positive in contrast to the red. However the Output is not the opposite color of the input with its light blue and therefor still represents a positive or neutral color for me. The Control and State are also opposite - so they must have a connection of some sort - Control being in the top, must be the main control of the system; maybe a sort of main system input, that can variate depending on who is using the system. State could be the State of the system; if it’s offline, active, processing etc.
- A computer



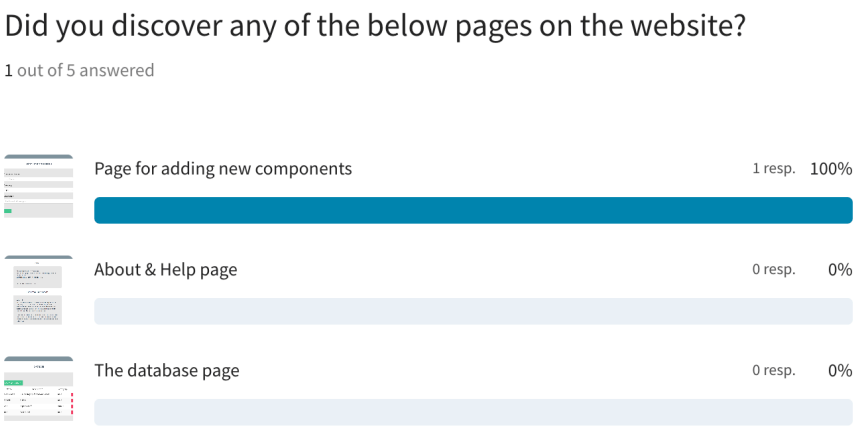
Tell us about the stencil

5 out of 5 answered

- I think the stencil pane is components that one can choose to add to the system. For example if it is a software program, then one can choose to add a network card in order to upgrade that software.
- Computer or server
- De overordnede inddelinger af de enkelte elementer i processen
- They represent allready pre-programmed systems and additions, that can be plugged into the system. Like an IT pottatohead. The colors signals where they need to connect, in relation to the system boxes. There is one printer, but there could be many,

depending of the size of the user; where as all the printers contain a different setting depending on their printer number - as in used within a larger company, with different printers on different levels in a building for instance?

- To drag the ”hardware type” where it supposed to be in the box



Did you discover you can right click components? If not, how can we make it more clear?

- I did not. Perhaps a small panel on the right side where one can see some of the basic keybindings.
- No
- Left click eller lav en boks, hvor der står at man kan højre klikke.
- Yes, but there were only two options and i didn’t get far from there
- Text it somewhere

Did you succeed in creating a basic model and running it?

- After going back i noticed one could pull arrows toward the fitting systems parts. Then i pressed the top right green arrow, and alot of information appeared. Beneath were a system thread, where it said ”Vulnerabilities” that said 801 and ”Severity” that said HIGH. I then was able to read about som of the systems attached to each components.
- No
- Tror jeg ikke
- No i didnt - and after trying again i learned that the white circles becomes an arrow and can be dragged. This could maybe be more clear if they are an arrow from the beginning - og just a pointing triangle, to indicate their ability to connect?
- Different amount of vulnerabilities

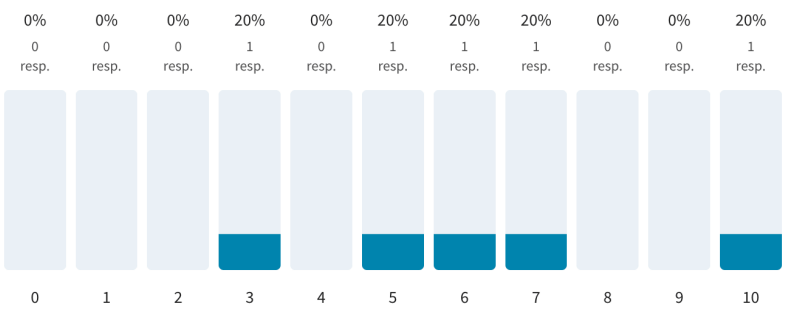
Do you think there is a need for a tool like this for non-technical people?

5 out of 5 answered

How interested would you be in using a tool like this to improve a system you own?

5 out of 5 answered

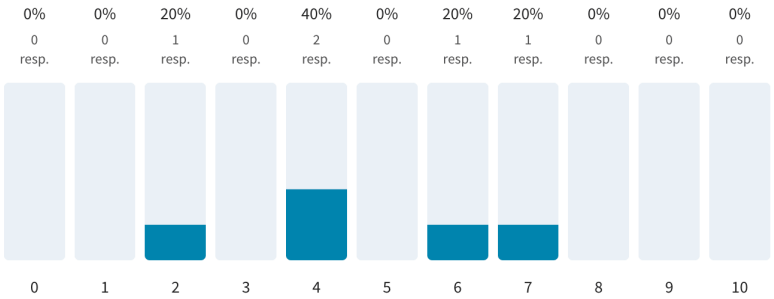
6.2 Average rating



How intuitive did the website feel to you

5 out of 5 answered

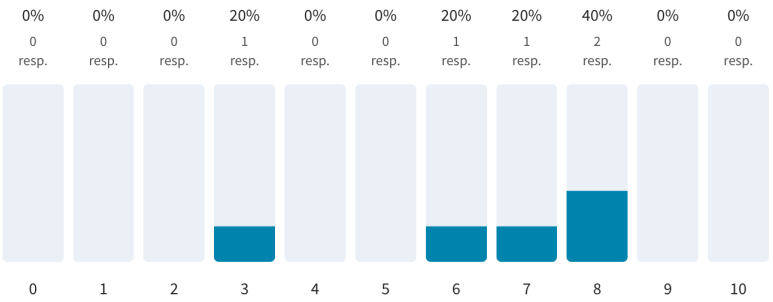
4.6 Average rating



How would you rate the design of the website?

5 out of 5 answered

6.4 Average rating



Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Technical
University of
Denmark

Richard Petersens Plads, Building 322
2800 Kgs. Lyngby
Tlf. 4525 3031

www.compute.dtu.dk