Bachelor Thesis

Conception and Development of a Threat Modeling Tool

By

Tobias Reski

In

Medien und Informationswesen

Faculty

Medien und Informationswesen

Academic Supervision:

Prof. Dr. Andreas Schaad

Second Examiner:

Dipl.-Ing. Oliver Vauderwange

23.11.2018 – 23.02.2019

Hochschule Offenburg
Abstract

The development of secure software systems is of ever-increasing importance. While software companies often invest large amounts of resources into the upkeeping and general security properties of large-scale applications when in production, they appear to neglect utilizing threat modeling in the earlier stages of the software development lifecycle. When applied during the design phase of development, and continuously during development iterations, threat modeling can help in following a “Security by Design” approach. This approach allows issues relating to IT security to be found early during development, reducing the need for later improvement – and thus saving resources in the long term. In this thesis the current state of threat modeling is investigated. Based on this analysis, requirements for a new tool are derived. These requirements are then used to develop a new tool, called OVVL, which utilizes all main components of current threat modeling methodologies, as well as functionality not available in existing solutions. After documenting the development process and OVVL in general, this newly developed tool is used to conduct two case studies in the field of e-commerce and IoT.
Acknowledgements

Many people helped in shaping this thesis, and their support benefited this project greatly. To begin with, I want to thank my supervisor, Prof. Andreas Schaad, for his patient support, guidance, and advice he provided throughout the preparation of this thesis. His expertise in the field and continuous questions helped me greatly in understanding the many different concepts I came across during my research. Additionally, I would also like to thank the rest of the team, Janick Born and Benedikt Balint, whose skills and ideas were a crucial factor in making this project a reality. The combined effort in our team made the development and documentation a very comfortable environment to work in. Finally, I also want to thank Prof. Ralf Reski, for taking the time for proofreading, valuable suggestions and guiding me in writing less, saying more.
# Table of Contents

Abstract ........................................................................................................................................... I
Acknowledgements .......................................................................................................................... II
List of Figures ................................................................................................................................... 1
List of Tables ..................................................................................................................................... 3
1. Introduction .................................................................................................................................. 4
2. Background Threat and Risk Assessment ..................................................................................... 5
   2.1 Threat Modeling ..................................................................................................................... 6
   2.2 STRIDE ................................................................................................................................... 8
   2.3 CPE, CVE, and CVSS .......................................................................................................... 10
   2.4 Existing Threat Modeling Tools ........................................................................................... 11
      2.4.1 Microsoft Threat Modeling Tool (TMT) ...................................................................... 11
      2.4.2 OWASP Threat Dragon ............................................................................................... 13
      2.4.3 ThreatModeler, IriusRisk and securiCAD ................................................................. 14
3. Open Weakness and Vulnerability Modeler ................................................................................ 16
   3.1 Scope .................................................................................................................................... 16
   3.2 Team structure ...................................................................................................................... 17
   3.3 MVP Roadmap ...................................................................................................................... 17
   3.4 Foundation ........................................................................................................................... 19
      3.4.1 Current Threat Modeling State ..................................................................................... 19
      3.4.2 Requirements ............................................................................................................... 19
      3.4.3 Feature List ................................................................................................................... 20
   3.5 Concept ................................................................................................................................. 22
      3.5.1 3-A-Model .................................................................................................................... 22
      3.5.2 Conceptual Data Model ................................................................................................. 23
      3.5.3 Design .......................................................................................................................... 26
   3.6 Implementation ...................................................................................................................... 27
      3.6.1 Technology Stack ......................................................................................................... 28
      3.6.2 Deployment Architecture ............................................................................................ 31
      3.6.3 Class Structure and Data Transfer .............................................................................. 31
      3.6.4. Threat and Vulnerability Analysis ............................................................................. 33
      3.6.5 State Management ........................................................................................................ 34
      3.6.6 GitHub Version Control ............................................................................................... 35
      3.6.7 Hosting and Continuous Integration ............................................................................ 37
   3.7 MVP Overview ....................................................................................................................... 37
3.7.1 Implementation of the Conceptual Data Model ......................................................... 37
3.7.2 Feature Stack ............................................................................................................. 38
3.7.3 MVP Feature Showcase ............................................................................................ 39

4. Case Studies ................................................................................................................ 44
   4.1 System X .................................................................................................................... 44
   4.2 IoT System .................................................................................................................. 48
   4.3 Discussion of the Case Studies in Respect to the Initial Requirements ....................... 50
   4.4 Modeling the Case Studies in Microsoft’s TMT ......................................................... 51

5. Discussion .................................................................................................................... 53

6. Conclusion .................................................................................................................... 57

7. References .................................................................................................................... 58

Appendix A: Design Guideline OVVL ............................................................................. 61
Appendix B: System X Modeled in OVVL ........................................................................ 72
Appendix C: System X Modeled in Microsoft’s TMT ......................................................... 73
Appendix D: Comparison of CPE and CVE queries loading times in OVVL and the NVD [20] 74
Declaration of Independent Work ...................................................................................... 76
List of Figures

Figure 1: Risk Matrix. Modified after [4]. ............................................................................................................. 6
Figure 2: DFD of a fictitious software architecture using "Microsoft Threat Modeling Tool" [10]. .................. 7
Figure 3: Microsoft Threat Model Tool design view [10]. .................................................................................. 12
Figure 4: OWASP Threat Dragon desktop application DFD view. ................................................................. 14
Figure 5: OVVL Logo. ................................................................................................................................... 16
Figure 6: Roadmap outlining the development process of OVVL’s MVP. ..................................................... 18
Figure 7: The 3-A-Model showcasing OVVL’s data processing ................................................................. 22
Figure 8: UML diagram of OVVL’s Conceptual Data Model. ...................................................................... 23
Figure 9: Color definition of the interactor element in each of its possible states ...................................... 26
Figure 10: Abstracted component structure of our Angular application .................................................. 29
Figure 11: OVVL deployment architecture .................................................................................................. 31
Figure 12: Abstracted conversion of a "Process" to a "ProcessDTO". The datatypes of the displayed properties either follow the TypeScript notation or consist of custom interfaces .......... 32
Figure 13: DTO collection in a "ThreatModelDTO" object ............................................................................. 32
Figure 14: Transfer of the "ThreatModelDTO" to the backend. The properties of the corresponding "ThreatModel" Java object can be accessed by auto-generated getter and setter methods .......... 32
Figure 15: "Threat" class diagram .................................................................................................................. 33
Figure 16: Basic "Model-View-ViewModel" design pattern applied to abstract Angular application .. 34
Figure 17: Basic communication within an Angular application implementing Redux ................................ 35
Figure 18: OVVL web application GitHub [53] repository. ........................................................................... 36
Figure 19: Parts of the Conceptual Data Model discussed in section 3.5.2 currently implemented in the MVP ........................................................................................................................................ 38
Figure 20: OVVL working space. .................................................................................................................... 39
Figure 21: DFD of an arbitrary software system, created with OVVL .......................................................... 40
Figure 22: Zoomed in view of an arbitrary DFD in OVVL ............................................................................. 40
Figure 23: Specification of a process in OVVL ............................................................................................... 41
Figure 24: Updated DFD elements in OVVL .................................................................................................. 41
Figure 25: CPE-Selection in OVVL ................................................................................................................ 41
Figure 26: Found STRIDE threats within an arbitrary system in OVVL ....................................................... 42
Figure 27: Highlighting of DFD elements and displaying of additional information after a STRIDE threat has been selected in OVVL ................................................................................................. 42
Figure 28: Found CVE-References ranked by their CVSS impact score in OVVL ........................................ 43
Figure 29: Highlighting of a DFD element and displaying of additional information after a CVE-Reference has been selected in OVVL .................................................................................. 43
Figure 30: Technical Deployment of System X web application [58, p. 9] .................................................... 45
Figure 31: Zoomed in view of System X’s DFD, created in OVVL ............................................................... 45
Figure 32: Set properties and selected CPE in the Apache Web Server specified in System X’s documentation ................................................................................................................................. 46
Figure 33: Highlighted spoofing threat in System X. ......................................................................................... 46
Figure 34: Oracle Database affected by two vulnerabilities .......................................................................... 47
Figure 35: DFD of the IoT system discussed in section 4.2 ......................................................................... 49
Figure 36: Threat analysis of System X in OVVL (left) and Microsoft’s TMT (right) reports 62 threats in both tools. Both DFD’s set the same properties ........................................................................ 52
Figure 37: System X (Section 4.1) modeled in OVVL ..................................................................................... 72
Figure 38: System X (Section 4.1) modeled in Microsoft’s TMT ................................................................. 73
Figure 39: Loading time of the CPE query "Windows 10 64-bit" in OVVL ...................................................... 74
Figure 40: Loading time of the CVE query of the CPE of "Windows 10 64-bit" in OVVL....................... 74
Figure 41: Loading time of the CPE query "Windows 10 64-bit" in the NVD [19]............................... 75
Figure 42: Loading time of the CVE query of the CPE of "Windows 10 64-bit" in the NVD [19]......... 75
List of Tables

Table 1: STRIDE to DFD elements matrix [16, p. 1105]. ................................................................. 9
Table 2: OVVL Feature Prioritization and Comparison................................................................. 20
Table 3: Comparison of the loading times of both the “Windows 10 64-bit” CPE and the derived CVE search query between OVVL and the NVD [20]. Values are displayed in milliseconds (ms) and rounded to the nearest integer. ........................................................................................................ 55
1. Introduction

With the globalization of software infrastructure and the resulting increase in user numbers, designing and developing secure software and software architectures is of ever-increasing importance. When neglected, overlooked errors can result in severe financial and reputational losses. Because the sophistication of attacks is increasing, designing software stacks with security concerns already in mind from the beginning onwards is crucial. In this context, threat modeling can be a useful tool for conducting an informed analysis of the security risks inherent to a software system. In an ideal setting, threat modeling is utilized in parallel to the development lifecycle. While many resources detailing the threat modeling process are available, free tools offering threat modeling support are few and far between, as well as lacking in several areas. As a result, integrating threat modeling in the development process might seem for many developers as tedious, unnecessary, and generally hard to do.

Improving the current state of threat modeling and lowering the barrier of entry for its integration in software projects is the goal of this thesis. In this context, the lacking areas of available tools are analyzed and suggestions for possible improvements are made. Additionally, the suggested improvements are showcased by means of implementing them in form of a new threat modeling tool.

This thesis is split into three parts. Initially, we provide a general overview over threat modeling. Here, the general background behind risk assessment in symbiosis with threat modeling is given. Additionally, existing tools are analyzed in context of their feature palette and design. The second part specifies its scope and the structure of the whole project further and continues with the discussion of the development of a new tool. This discussion builds onto the analysis of the first part and includes derived requirements for a new tool, its general concept and implementation structure. Also, an overview over the new tool in its current development state is given. In the third part, the new tool is utilized to conduct two case studies, thus evaluating how the new tool can aid in the development process.
2. Background Threat and Risk Assessment

Designing and developing secure software architectures and applications is of increasing importance. "Security by Design" is an approach modern software development should adopt as part of a dedicated Secure Development Lifecycle (SDLC). In order to ensure that a software system cannot be exploited once it is in production, implementing dedicated threat and risk assessment processes during the overall system lifecycle is crucial.

Risk assessment is meant to provide a systematic and structured approach to identifying and evaluating possible threats and risks a system faces. By using this approach, the impact of a potential threat on a system can be estimated [1, pp. 1-21]. Generally, the basic concepts for a software risk assessment can be broken down into the following artefacts[2]:

- **Asset.** Anything that needs to be protected within a system is described as an asset. Assets can be assigned with a value to assess the impact of a system compromise.
- **Threat.** The potential circumstance in which an asset and thus a system can be compromised.
- **Vulnerability.** A vulnerability describes a weakness in an asset, which, if exploited, may give rise to a threat.
- **Risk.** The possibility of losing something of value.

To calculate the risks a system faces, the impact and probability of a threat need to be defined. The impact describes the effect on the software system, should a threat be realized, while the probability describes the likelihood of the threat realization. Defining these factors is often guesswork, but doing so allows for the derivation of the following formula [3]:

\[
\text{Impact} \times \text{Probability} = \text{Risk}
\]

Mapping this formula to a risk-matrix visualizes how risks – and thus threats – could be prioritized (Figure 1):
By applying risk assessment in the early development stages and applying it iteratively during the development process, threats can be identified, prioritized and mitigated before significant resources are used for implementations that otherwise need to be refactored or improved upon. While no system can be perfectly protected from malicious actors, acknowledging and mitigating known vulnerabilities and threats can help to keep a systems threat susceptibility to a minimum and reduce the overall attack surface. Risk and threat assessment allows for development resources to be applied to relevant issues within a system, which in turn makes the development process more efficient, reduces the system downtime, and increases the return of investment (ROI) [5].

2.1 Threat Modeling

Threat modeling is a parallel activity to risk assessment. It is a systematic approach to build a structured representation of a software system and its required security properties, resulting in a general overview over potential weaknesses a system faces. This is done by modeling and analyzing the logical entities of a system, after which potential vulnerabilities and threats can be identified [6]. By rating their severity and impact, these threats can then aid in the risk assessment process.
One possible way to build a threat model is by applying the following steps [6] [7]:

1. Decomposing an application into modular entities by identifying its assets (e.g. Web-Application, Database).
2. Creating a data flow diagram (DFD) outlining the structure and communication flow of assets by breaking them down into their sub-components, if applicable. The resulting elements are displayed in the DFD as interactors, processes or data stores [8, p. 336], though an extensive system model might define them more precisely (e.g. Browser Client, Web Server)\(^1\). Sections within the DFD, in which data processing changes its trust level, are visualized in the model as trust boundaries [8, p. 336, 9, p. 13].
3. Identifying and modeling of all possible threats (e.g. by applying STRIDE as discussed in section 2.2), even if they cannot be exploited [10, p. 1].
4. Prioritizing threats by rating their severity.
5. Deriving steps that can be taken to mitigate threats.
6. Continuously improving the model and its derived threats, depending on changes in a systems architecture.

Due to its modular approach, threat modeling can be applied not only to simple, but also to complex systems. Since about 50% of security issues arise from flaws in the initial design of an application [12, p. 53], applying threat modeling before or during the design phase can help finding security issues early in the development process. This ensures that resources

---

\(^1\) The DFD does not have to be constructed to its lowest possible level to get a gist of a systems security properties. Still, the accuracy of both the DFD and the resulting threat model correlate.
can be assigned more effectively during the actual development, since it is more cost efficient to resolve issues before a system is in development then after deployment. Identifying threats early also helps to develop “realistic and meaningful security requirements” [10, p. 1].

2.2 STRIDE

STRIDE is a threat modeling approach developed by Microsoft [13] and is based on the assumption, that threats software architectures are susceptible to can be clustered [14, p. 5]. The STRIDE model divides IT-Security threats into the following 6 categories [13, p. 1].

Spoofing of user identity

Spoofing happens when an attacker breaches a user’s authentication data, either through having access to personal information or through being able to replay the authentication procedure [13, p. 2]. By impersonating users with extensive access rights to a system, an attacker might be able to compromise the system in many different aspect (e.g. access and manipulation of data, weakening of the security structure).

An example of a spoofing threat is not using encrypted authentication and/or authorization protocols. User credentials can be intercepted by an attacker, who in return is able to impersonate the user. A strong, encrypted authentication system can prevent this type of threat.

Tampering with data

Tampering is a threat which describes an attacker being able to manipulate “system or user data with or without detection” [13, p. 3]. These types of attacks can range from changing the price of an item in an online-shop to deleting a production database. Similar to addressing spoofing threats, a software system could use encryption and digital signatures in order to prevent tampering.

Repudiation

Repudiation describes an attacker not being able to be traced or an attack not being attributable to them. It is not an attack itself, but the manipulation of data in order to hide traces of an attack. For example, an attacker might be able to spoof his identity when
performing an unauthorized action and/or tamper with security logs in order to cover up an attack [13, p. 4].

**Information disclosure (privacy breach)**

An information disclosure threat is realized when a perpetrator is able to gain access to confidential data. In this case, data can be anything from files to network traffic. A tampering threat differs from a spoofing threat in that an attacker can access data directly [13, p. 9].

**Denial of Service (D.o.S.)**

Denial of Service describes a service being unavailable due to a lock up of resources as a result of an attack. These threats can be hard to prevent, since they are usually realized by flooding the target system with requests from different physical locations [15], e.g. by the use of so-called “botnets”.

**Elevation of privilege**

Elevation of privilege describes an attacker being able to gain privileged access, which in return gives them the ability to compromise the entire system [13, p. 6]. This is especially dangerous if the perpetrator can gain access without detection. When an elevation of privilege threat is realized, an “attacker has effectively penetrated all system defenses and becomes part of the trusted system” [13, p. 6].

For each type of element within a DFD, certain threats are applicable [14, p. 5]. Thus, STRIDE threats can be mapped to a system abstracted by a DFD (Table 1).

*Table 1: STRIDE to DFD elements matrix [16, p. 1105].*

<table>
<thead>
<tr>
<th>Element</th>
<th>Spoofing</th>
<th>Tampering</th>
<th>Repudiation</th>
<th>Information Disclosure</th>
<th>Denial of Service</th>
<th>Elevation of Privilege</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interactors</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processes</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Data Stores</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Data Flows</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
It must be noted that the aforementioned mapping of STRIDE to DFD elements is applied to generic elements. When applied to more specific elements and communication scenarios, this matrix (Table 1) should be fine-tuned [14, p. 5]; e.g. a data store might not always be susceptible to denial of service attacks, but to repudiation when implementing a log service.

STRIDE, when fine-tuned and used in combination with an extensive system model, aids in giving a security overview over a software system. While it does not cover every possible attack explicitly, its more general approach at identifying threats is a good starting point for identifying the security weaknesses of a software system [14, p. 1].

2.3 CPE, CVE, and CVSS

While STRIDE is used to gain a general threat overview, knowing about explicit software vulnerabilities improves a threat model further. Elements in a DFD can be defined more clearly by specifying a certain software make, for which information about known vulnerabilities is stored and accessible in public databases. Found vulnerabilities can then be mitigated before they are exploited.

CPE

CPE stands for “Common Platform Enumeration” and is a “structured naming scheme for information technology systems, software and packages” [17]. CPE’s are meant to name software products in a standardized manner. One CPE can be linked to only one Software. For example, “Windows 10 1607 64-bit” CPE is defined as [18]:

\[ cpe:2.3:o:microsoft:windows\_10:1607:*:*:*:*:x64: *\]

For each CPE, known software vulnerabilities can be found in the form of CVE-References (“Common Vulnerabilities and Exposures” [19]).

CVE

CVE is a standardized system for referencing known software vulnerabilities, as provided by the NVD [20]. CVE-References include, amongst other things, a summary of the vulnerability, the date the vulnerability was published, one or multiple CPE-References and a CVSS (“Common Vulnerability Scoring System” [21]) score. A CVE-Reference might be named in the following way [22]:

\[ CVE - 2018 - 8505 \]
CVSS

CVSS “provides a way to capture the principal characteristics of a vulnerability and produces a numerical score reflecting its severity” [21]. This score describes the impact of the vulnerability. The severity of a vulnerability is based on aspects like its attack complexity or its impact on integrity and confidentiality. Each CVE-Reference includes a CVSS score, which enables the ranking of vulnerabilities.

2.4 Existing Threat Modeling Tools

Many aspects of threat modeling can be automated or supported by tools, which can make it easier to integrate threat modeling into the development lifecycle. During the course of this project, we performed an extensive (informal) analysis of the free existing tools, focusing on their features and on their user experience (UX) design. By doing so, we aim to derive requirements for our own tool, and thus offer an approach to threat modeling which significantly improves upon the existing tools. Currently, there are two free and several commercial tools available.

2.4.1 Microsoft Threat Modeling Tool (TMT)

Microsoft provides a free tool in the form of a Windows desktop application [11]. Its threat analysis is based on the STRIDE methodology. TMT’s main features include [23]:

- Extensive documentation.
- Creating DFDs manually.
- Setting properties of DFD elements and adding custom properties.
- Listing potential STRIDE threats following an automated DFD analysis.
- Creating custom threats.
- Manually prioritizing threats.
- Exporting a CSV file of found threats.
- Creating a threat report in form of a HTML file
TMT provides a wide array of features and covers most use cases related to threat modeling, but it is lacking in several aspects. One drawback is the static behavior of custom threats and element properties. Once set, they are only applied to the respective element. Custom properties do not influence the threat analysis. Tracking of different model iterations is also not supported, which makes tracking of threat mitigations not possible within Microsoft’s tool. Additionally, only making the tool available for Windows platforms might constitute a barrier of entry for some projects [24]. The tool cannot be further customized or adopted by the open-source community.

Viewed from a UX standpoint, Microsoft’s TMT might seem dated to some users. While our analysis of the tools design was not based on defined UX acceptance criteria, we were able to note the following impressions:

- A “Getting Starting Guide” can help with understanding the different aspects of a tool and of threat modeling in general.
- Drag & Drop to place elements is an intuitive approach to DFD modeling.
- Linking found threats to their respective elements by highlighting them aids in understanding the threat model better.
• The different DFD elements and their sub-components are displayed in one long, by default unfolded list. This can make it hard to find certain elements during the modeling process.
• The working area seems to be boxed in by too much information and settings.
• A lot of text is used to explain features and properties. While this generally helps with understanding certain aspects of the tool, it can be confusing to get an overview over the current process.
• Many sub-windows take up space unnecessarily by sometimes containing no information. This happens when a feature usually managed from a certain window is currently not in use.
• Unclear, repeating icons and an unintuitive navigation structure can make it hard to create a threat model in a timely manner.
• While multiple diagrams can be created in one project file, they cannot be linked to construct complex systems.
• Changing between design and analysis view is not intuitive.

The fact that Microsoft’s TMT offers more features compared to OWASP’s Threat Dragon (discussed in section 2.4.2), made its design and feature analysis crucial for deriving requirements for our tool. These requirements are discussed in section 3.4.2.

2.4.2 OWASP Threat Dragon
As described in its documentation [25], Threat Dragon is an open-source threat modeling tool developed by OWASP and currently in the early stages of development. It is freely available in the form of a web application and as a standalone desktop app for both Windows and MacOS. Its main features include [25]:

• Creating DFDs manually.
• Setting properties of DFD elements.
• Adding custom STRIDE threats.
• Manually prioritizing threats.
• Linking threat models to GitHub Repositories.
OWASP Threat Dragon Discussion

Being open source and platform independent, Threat Dragon is a tool that could be integrated in most development lifecycles without much effort. Its clean design implementation aids in giving a clear overview over architectures of varying complexity.

Threat Dragon’s community driven development approach constitutes a few major drawbacks when it comes to feature availability. Compared to Microsoft’s TMT, Threat Dragon currently offers neither automatic threat analysis, exporting of threat data, nor the automatic generation of threat reports. DFD element properties are currently very limited and no custom properties can be set. With Threat Dragon’s last commit to its master branch being January 20, 2018 [26], its ambitious goals defined in the roadmap still seem far away from completion [27]. In its current state, Threat Dragon is still missing several key features for it to be considered a valid tool for threat modeling.

2.4.3 ThreatModeler, IriusRisk and securiCAD

ThreatModeler [28], IriusRisk [29] and securiCAD [30] are enterprise software solutions offering risk and threat assessment on a larger scale. Since neither of these tools are available for free, we were not able to do an extensive analysis of these applications, though their respective websites give an overview over their features [28–30]. Compared to Microsoft’s TMT and Threat Dragon, these tools offer a wider array of functionality, like complex diagram creation, ticket creation for agile teams, multiple threat analysis options and customer support.
ThreatModeler, IriusRisk and securiCAD Discussion

While these extensive features can be immensely valuable for large software projects, we think that it might cause threat modeling to be viewed as a separate project, instead of it accompanying the usual development lifecycle – which, combined with their commercial nature, might make it hard to justify integrating these tools into software projects developed by smaller teams. We want to follow a different approach and offer a lightweight tool with the most crucial features for threat modeling. By doing so, we want to make threat modeling accessible for smaller teams and projects, as well as big projects, where a threat analysis can be used as a starting point for further risk assessment.
3. Open Weakness and Vulnerability Modeler

3.1 Scope

In the context of this thesis, the following goals had to be achieved:

1. Analysis of the current threat modeling state, including the comparison of existing Threat Modeling tools.
2. Deriving requirements for a new threat modeling tool, which we call “Open Weakness and Vulnerability Modeler” (OVVL).
3. Deciding on a feature list for OVVL, including features that are not offered by other tools.
4. Development of OVVL as a “Minimum Viable Product” (MVP), which implements the core features.
5. Analyzing how OVVL utilizes threat modeling, aiding in the development process.

When viewing the project as a whole, its scope can be divided into two sections: Development of an MVP and the development of the full product. As such, the whole project spans further than the context of this thesis, during which the MVP is built. While the MVP does not implement every feature required for the final product, it still can be considered a working prototype for which valuable user feedback can be acquired. The development of the final product can then respond to the feedback, while building upon the foundation created by the MVP.

Because this thesis lays the groundwork for the actual product, the discussion in the following sections is not limited to only the scope of the MVP. Some aspects, such as the base requirements discussed in section 3.4.2 and the feature list discussed in section 3.4.3, are relevant for the development continuing after the completion of this thesis as well. Still, the main focus of this thesis is the MVP development. Thus, when discussing OVVL in the following sections, generally the MVP is referred to.

---

2 The VV stands for a W and V. “Open Weakness and Vulnerability Modeler”. 
3.2 Team structure

OVVL was designed and built in a team effort. The team was composed of:

- A project lead coordinating the project and providing the background knowledge and industrial expertise.
- One architect (myself) responsible for the development and documentation of the tool.
- One engineer assisting in the development of the UX and general design.
- One assisting engineer, responsible for the development and documentation of an additional attack tree modeling tool, later to be integrated into OVVL.

We followed an agile approach throughout the course of this project, in order to ensure the timely mitigation of issues arising during the development process. This includes the iterative development of the different aspects of OVVL. During weekly meetings, these iterations were designed and implemented after elaborate analysis and discussion of the current development state in relation to the planned feature implementation. By following this approach, we were able to respond to new requirements arising throughout the project, while keeping the core concept in mind. In an ideal agile setting, the role of the project lead should have been split into a product owner and a project manager [31].

3.3 MVP Roadmap

A brief roadmap outlining the development process visualizes how the insights and results discussed in the following sections came to live. During the second half of October 2018, the general threat modeling state was analyzed. This served as a baseline from which existing tools could be compared – and their shortcomings summarized. From these shortcomings, requirements and features for a new tool could be derived. Additionally, analyzing existing tools from a design perspective opened up the discussion for our own tools design. The process leading up from the completed analysis to an extensive concept for OVVL spanned from November to December 2018. During this time, development on the actual tool was started simultaneously. At first, the right technology stack had to be chosen and a rudimentary application skeleton built. As we finalized the feature list, features we prioritized got implemented in the tool. After the design conception for the scope the MVP covered was finalized in the second half of January 2019, the tool got rebuilt with the new
design, while keeping all the features already implemented. On the 22.01.2019, a presentation of this thesis outlining our project had to be held. Thus, this date served as a deadline for constructing a working MVP including the updated design. In February 2019, case studies utilizing the finished MVP were conducted, bugs present in the tool mitigated, the tool hosted and finally this thesis submitted. Spanning the whole course of the MVP development, results gathered were documented in the context of this thesis. These results are discussed in the following sections.

Continuing forward, we first plan on gathering feedback on the MVP during March 2019. Based on this feedback, features in our backlog can be re-prioritized, new features can be added to the backlog, and flawed aspects of current features implementation can be improved. Development will continue until the end of August 2019, at which time we plan on delivering a fully functional product.

![Roadmap outlining the development process of OVVL's MVP](image)

*Figure 6: Roadmap outlining the development process of OVVL’s MVP.*
3.4 Foundation

3.4.1 Current Threat Modeling State

In order to set meaningful requirements for the development of a new threat model tool, first the present state of the threat modeling field needs to be discussed. Members of SAFECODE, a “global, industry-led effort to identify and promote best practices for developing and delivering more secure and reliable software, hardware and services” [32, p. 3], came to the following conclusions [32, p. 13]:

- While the demand for useful threat modeling tools is ramping up, existing solutions do not meet the requirements set by security specialists to a sufficient extend.
- Only a few tools exist and come with a limited guidance availability. This can make it harder for teams to get started with threat modeling.
- Integrating threat modeling into existing development processes can be challenging.
- Since most security issues only become a concern when exploited, insight gained by threat modeling might not immediately be seen as useful.

Even though this discussion took place in 2015, the state of threat modeling stayed seemingly unchanged. While the demand for threat modeling is increasing, only two free solutions are offered. Microsoft’s TMT and Threat Dragon both are helpful tools for threat modeling but are missing some essential features. Commercial tools offer a wide range of features but are harder to integrate into the development lifecycle due to their complexity. As a result, the growing need for threat modeling is not met in the required extend, an issue which we want to solve with OVVL.

3.4.2 Requirements

After a careful analysis of the existing tools as discussed in section 2.4, as well as taking the suggestions from SAFECODE [32, pp. 13-14] into account, we derived several high-level requirements for the final version of OVVL:

- A user must be able to visualize complex communication systems.
- A user must be able to analyze the system for threats and vulnerabilities.
- A user must be able to customize model elements and threats.
- A user must not be limited by their utilized platform.
- Found issues, their mitigation status and model iterations must be traceable.
The system and its threats must be presentable in a visually appealing way.

The current threat model must be storable both online and offline.

Multiple users must be able to work on one threat model.

The threat modeling process must be intuitive, guiding the user when necessary.

These requirements must be met before we consider OVVL to be a fully-fledged product. Therefore, they serve as our baseline, from which a feature list can be derived. When we design features or discuss their current implementation of our tool, these requirements are kept in mind and thus serve as a guide towards the final product.

### 3.4.3 Feature List

The core features we derived from the aforementioned requirements are listed in Table 2 and ordered according to their priority within the project. Additionally, we compare the features to both Microsoft’s TMT and Threat Dragon’s current feature implementation. Through this comparison, we want to showcase how we plan on improving upon the existing state of threat modeling.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Microsoft TMT</th>
<th>Threat Dragon</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Architecture Modeling</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drawing DFDs.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Automatic Threat Analysis</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Based on the STRIDE model, dynamically dependent on the drawn model.</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td><strong>DFD Element Subtypes</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>General DFD elements must be distinguishable further in subtypes. Manual naming of DFD elements independent of their type.</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td><strong>DFD Element Properties</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set properties influence the threat analysis.</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Linking CPE to DFD Elements</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Setting certain software makes for DFD elements.</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td><strong>CVE Vulnerability Analysis</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analyzing the model for known software vulnerabilities.</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

*Table 2: OVVL Feature Prioritization and Comparison.*
<table>
<thead>
<tr>
<th>Feature</th>
<th>Available Options</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vulnerability Ranking</strong></td>
<td>![Image]</td>
</tr>
<tr>
<td>Automatic prioritization of CVE-References depending on their CVSS score.</td>
<td>![Image]</td>
</tr>
<tr>
<td><strong>Threat Prioritization</strong></td>
<td>![Image]</td>
</tr>
<tr>
<td>Manual prioritization of found threats.</td>
<td>![Image]</td>
</tr>
<tr>
<td><strong>Saving and Loading Models</strong></td>
<td>![Image]</td>
</tr>
<tr>
<td>Saving and loading threat models both locally and online.</td>
<td>![Image] Only locally Web-app online, desktop application locally</td>
</tr>
<tr>
<td><strong>Automatic Threat Reports</strong></td>
<td>![Image]</td>
</tr>
<tr>
<td>Automatically generating threat reports for presentation.</td>
<td>![Image]</td>
</tr>
<tr>
<td><strong>Custom Threats</strong></td>
<td>![Image]</td>
</tr>
<tr>
<td>Custom threats can be defined and are taken into account during analysis.</td>
<td>![Image] Custom threats are set after the analysis No Analysis, only custom threats can be defined</td>
</tr>
<tr>
<td><strong>Model and Mitigation Tracking</strong></td>
<td>![Image]</td>
</tr>
<tr>
<td>Version management for the threat model.</td>
<td>![Image]</td>
</tr>
<tr>
<td><strong>Custom Properties</strong></td>
<td>![Image]</td>
</tr>
<tr>
<td>Setting custom properties for DFD elements.</td>
<td>![Image]</td>
</tr>
<tr>
<td><strong>Data Export</strong></td>
<td>![Image]</td>
</tr>
<tr>
<td>Exporting of both DFD and threat data, to enable further processing in other tools.</td>
<td>![Image]</td>
</tr>
<tr>
<td><strong>User Accounts</strong></td>
<td>![Image]</td>
</tr>
<tr>
<td>Account management to allow multiple users to work on the same model.</td>
<td>![Image]</td>
</tr>
<tr>
<td><strong>Project Tracking Integration</strong></td>
<td>![Image]</td>
</tr>
<tr>
<td>Linking threat mitigations to external project tracking software like JIRA.</td>
<td>![Image]</td>
</tr>
<tr>
<td><strong>Model Hierarchy</strong></td>
<td>![Image]</td>
</tr>
<tr>
<td>Allows systems subcomponents to be modeled in the same DFD.</td>
<td>![Image]</td>
</tr>
</tbody>
</table>
Additionally, we consider a clear focus on UX and design in general to be crucial, since the different functionalities we plan on offering might otherwise make our application seem too complex. By following a clean design approach and guiding the user when needed, we aim to develop a mostly self-explanatory and easy-to-use product. The design of our application is discussed further in section 3.5.3.

While all features must be implemented before we consider the tool a fully functional product, our iterative development approach allows us to offer a stable, useable tool utilizing the features currently implemented. The features implemented in the MVP are discussed in section 3.7.2.

3.5 Concept

In order to assure a structured and productive development, we decided on a clear concept before starting development. This allowed us to have a common goal in mind, while working on different aspects of the tool. The features listed in section 3.4.3 are part of the core concept, though this section focuses on the actual data and architecture structure. While concept iterations were developed when necessary, the models discussed in this section describe the current implementation of OVVL.

3.5.1 3-A-Model

The core functionality of our tool can be summed up in a conceptual model we named “3-A-Model”. It describes OVVL’s process of handling data — aggregation, automation and allocation.

![Figure 7: The 3-A-Model showcasing OVVL’s data processing.](image)
Here, aggregation refers to the collection of data on generic threats, custom threats defined by a user, and vulnerabilities (i.e. CPE and CVE data). The aggregation happens outside of the user scope in our backend systems. Dependent on the DFD model and its properties, relevant information is then fed to the threat model during the automated analysis process. After analysis, a user is able to allocate a mitigation status to found security issues, both inside our final tool, and in external applications. Building on this allocation, issues and model iterations can be tracked and thus accompany the development process.

Because this aspect enables the tool to be fully integrated into the development of a software system, we believe the actual implementation of the “3-A-Model” to be the key selling point of our application.

### 3.5.2 Conceptual Data Model

The conceptual data model, as depicted in Figure 6, represents the structural implementation concept of OVVL. This model was derived through the analysis of existing threat modeling tools and is based on our requirements and feature list, as defined in sections 3.4.2 and 3.4.3. As such, the conceptual data model provides an overview over the tool’s core-functionality and also serves as an implementation guideline.

Figure 8: UML diagram of OVVL’s Conceptual Data Model.

**UML Cardinality (Legend)**
- **1** Exactly one
- **0..1** Zero or one
- **0..*** Zero or more
- **1..*** One or more
**DFD Element to Threat Model Relation**

A DFD element can be either an interactor, process, data store or a data flow. A data flow represents the communication flow between elements. It is used to link elements together, so that not only the elements themselves, but also their interaction can be analyzed. These connected elements form a representation of the software- and communication architecture and thus the model for which a threat analysis can be made. To differentiate elements of the same type during the threat analysis, each element defines multiple selectable sub-types, e.g. “Browser Client”- and “Web Server”-Process. They can be distinguished further by setting different properties, like “Isolation Level” and “Accepts User Input”\(^3\). A clear distinction between elements not only of a different, but of the same type, is crucial for a meaningful threat analysis and thus a useful threat model. Visually the elements can be differentiated by their color and corresponding icons, an important aspect required to give a clear overview over the software architecture we are trying to protect.

**CPE to Element Relation**

In order to ensure a thorough threat analysis, it is necessary to allow for the mapping of certain software makes to the elements in the model. By searching for a certain software, e.g. “Firefox 62.0.2”, the user is returned a list of matching CPE’s, which they then can link to the element the search was requested from. By using CPE as an identifier, we make it possible to specify which software an element in the model is based on in a standardized manner. The resulting, more accurate element specification makes it possible to link known software vulnerabilities to the model during the analysis process.

The underlying data is provided by the National Vulnerability Database (NVD) [20], a database maintained by the National Institute of Standards and Technology (NIST) [33]. We host an up-to-date copy in our own database. By using this data, our tool can not only identify current software, but legacy software as well. We also implement a custom search interface, which enables the process of searching CPE’s and linking them to the elements to be significantly faster than NVDs official CPE search tool. This aspect is discussed further in section 5.

\(^3\) Currently these properties and sub-types defined in OVVL are the same as in Microsoft’s TMT. In the future, we plan on defining new specifications.
CVE and Threats

In order to facilitate an extensive analysis, our tool differentiates between threats and vulnerabilities. A threat is a possible risk of someone compromising and/or harming a system, while a vulnerability can be exploited and thus may give rise to a threat [16, p. 1105]. The final tool will distinguish between CVE, STRIDE threats and Custom-Threats, all of which are linked to their respective elements. For CVE-References being linked to the elements, a CPE must first be set by the user.

STRIDE threats are threats found for a specific system, based on a fine-tuned STRIDE-Model as explained in section 2.2⁴. They are applied to the elements dynamically during the analysis, depending on the element type, its connection and communication with other elements and its properties. Since these threats, unlike the CVE-References, cannot be ranked automatically, our tool will enable the user to prioritize the threats manually. While STRIDE threats are not meant to demonstrate detailed threats a system faces, we consider it a good approach for finding possible ways a system could be compromised.

Since each information system has different dependencies and possible threats, only having predefined threats (STRIDE) and vulnerabilities (CVE) would not cover every risk a system faces. By allowing for custom threats to be defined, new threats can be added to the underlying analysis logic and are applied to each new system and analysis a user creates.

Our system looks for all type of threats and vulnerabilities, depending on the available data. When combined, these threat types and vulnerabilities provide a wide overview over the risks a system faces.

Mitigation and Issues

For our tool to be helpful not only for giving a threat-overview over a system, but also during the development process, it is necessary for a user to be able to track and mitigate threats. This use case is covered by the ability to prioritize found threats and setting their mitigation status depending on whether the threat has been resolved or not. While it is not possible for

⁴ It is important to note that the threat definitions currently included in our tool are the same as in Microsoft’s TMT. When continuing development past the MVP, we plan on setting our own threat definitions.
our tool to analyze the development status of a system, we want to make it possible to link the mitigation status of threats to project tracking software like Jira [34] or FogBugz [35].

We plan on this being done by first having a mitigation list in our tool, from which a user then can create issue-tickets in these project tracking tools. While the creation of tickets is currently not implemented, the clear data structure within our current development status allows this feature to be added without much effort.

3.5.3 Design

When looking at Microsoft’s TMT, we found its design and usability to be somewhat limited. While serving its intended purpose, the general layout and behavior seemed dated and unintuitive to us. Based on this observation, we consider a user friendly and appealing design to be one of the main factors in making OVVL a competitive product.

Building a design guideline for OVVL was split into two parts. First, we analyzed Microsoft’s TMT UX behavior (discussed in section 2.4.1) and derived UX requirements for our own tool based on the results. These results then got incorporated in a general design guideline, which was specified by the engineer within our team, who was responsible for the UX and general design. The guideline includes screen mockups, UX behavior specification and font, color and element spacing definitions.

![Figure 9: Color definition of the interactor element in each of its possible states.](image-url)
In summary, the resulting guideline defines OVVL’s design the following way:

1. Separation of concern through the splitting of the view into a design and an analysis view.
2. At any point during the threat modeling process, only information relevant for that point is displayed.
3. A dark gray theme throughout the application, with only elements being colored.
4. A drag and drop system to place elements.
5. Highlighting of elements currently interacted with by the user.

The full design guideline can be found in Appendix A.

3.6 Implementation

Even though the scope of the project, as discussed in section 3.1, is split into two sections, “MVP” and “Full Product”, it is important to note that these parts, while separate, are not isolated from each other. Instead, they are connected by a core implementation structure spanning the whole project. This allows the full product to be built upon the foundation laid out by the MVP, and not having to start developing it from scratch. The core implementation structure in our case means:

- We use the same technology stack for the MVP and for the full product.
- Features are implemented according to their priority within our project, as discussed in section 3.4.3.
- Features are implemented following the modular approach laid out by the conceptual data model in section 3.5.2, so new features can be integrated into the tool without changing its core structure.
- In order to include them in the actual product, features added during MVP development were implemented rudimentary, though still in a proper and functional manner.
- We focus on clean code, to make it easier to keep an overview over the project’s logic while its complexity increases.
3.6.1 Technology Stack

Three factors were kept in mind while choosing the technology stack of OVVL: Scalability, support availability, and complexity. The chosen technologies as a whole must be scalable, so our tool can be used by a large number of users. No technology should constitute a bottleneck, limiting the performance of other parts of the application. Support availability is an important factor, since issues arising during development need to be resolved in a timely manner. Combined with the scalability factor, we concluded, that each technology must be established enough to have a large user base, but still modern enough to guarantee a reasonable performance. Each technology also must be complex enough to cover the wide range of development use cases the implementation of our tool requires. This includes the communication between front- and backend, state management, and general interconnectivity between the individual parts of our application. Also, the technologies must be complex enough to guarantee scalability, but still be straightforward enough to allow for fast paced development.

3.6.1.1 Frontend

Angular

Angular is an open-source web application framework [36], with its development being led by the Google Angular team [37]. It enables the development of client-sided web applications based on HTML, CSS, and the programming language TypeScript [38]. While standard CSS is the default styling language in Angular applications [39], we use SCSS to structure styling rules more clearly. TypeScript is a superset of JavaScript, utilizing the same syntax and semantics as JavaScript [40]. TypeScript facilitates optional type checking for a clearer code structure, which is crucial in large-scale applications like OVVL. Still, most external JavaScript libraries can be integrated into an Angular application as well.

The view of an Angular application is rendered with components. In some cases, these components are displayed within other components (Figure 8). A single component within our project, for example the toolbar, consists of an HTML-file displaying information to the user, a SCSS-file setting styling rules for the HTML-file, and a TypeScript-file binding data to the HTML-file, supplying it with information.

Additional services handle the communication between the application and the backend and feed processed data to these components. To manage the synchronization of data between
components, we additionally integrate the Redux state management engine [41]. This is discussed further in section 3.6.5.

![Diagram of Angular application components](image)

**Figure 10: Abstracted component structure of our Angular application.**

Angular’s component-based approach offers the possibility to dynamically render certain aspects of an application depending on the underlying data. By processing most of the data in background services, the view stays mostly unaffected should intense calculations take place, e.g. during the analysis process in OVVL. This aspect is crucial for the scalability of our application. Additionally, the ability to integrate external JavaScript frameworks into the application and the type safety offered by TypeScript, made Angular the right choice for our project.

### 3.6.1.2 Backend

**Spring Boot**

We utilize a Spring boot backend in our project, which handles most of the data processing. This includes a large part of the threat modeling and threat analysis. By keeping most of the calculation load away from the frontend, we increase the performance of the Angular application and thus improve the user experience.

Spring is a Java-based, open-source framework providing the infrastructure needed to create large-scale applications [42]. While Spring already offers an extensive foundation for the creation of a structured backend application, its extensive manual configuration options can
make it difficult to set up. Spring boot solves this issue by building on top of the Spring framework [43] and following the “convention over configuration” design paradigm [44]. In this case, Spring boot decreases the configuration time and boilerplate code immensely, by auto-configuring the core application and its dependencies. Additionally, a Spring boot application can be used as standalone – by default, a Spring boot application is deployed on its embedded Tomcat container, instead of a separate webserver.

**Swagger**

Swagger is a specification tool which can be integrated into a Spring boot application [45]. It is used to document and to generate REST web services by describing API endpoints [45]. Put more generally, Swagger defines the endpoints of a Spring boot backend, which can be addressed by the frontend application for communication purposes. The structure of the transferred data, HTTP method to be used, and URL address can all be manually defined in Swagger. Transferred data is then processed by the respective endpoint methods within the Spring boot backend. After processing, a response – which type and content is also defined in Swagger – is sent back to the frontend.

In our case, our API is defined by several URL endpoints. By naming these endpoints similar to the task that needs processing, we not only ensure a clear documentation, but also decrease the amount of communication errors between front- and backend.

Swagger’s automatic and specific endpoint generation and clear documentation structure make it a helpful tool within our project. It significantly improves the quality of our backend as well as the development speed.

**MongoDB**

MongoDB is an open-source document-oriented database [46]. Unlike a relational database, MongoDB stores data object as separate documents [47]. MongoDB focuses on high-performance, availability and scalability [47], which makes it ideal for our project.

In OVVL, we differentiate between five types of data which need to be stored: Threats, vulnerabilities (CVE), software makes (CPE), threat models, and user data. While the storage of the other types could be managed in a relation database, the big data set consisting of CVE- and CPE-References can be handled faster by MongoDB. This increase in performance
compared to existing solutions, like the official NVD [48], is of significant value. This aspect is discussed further in section 5.

3.6.2 Deployment Architecture

The architecture of our deployed system is shown in Figure 10. The communication between all entities in the system is HTTP based [49], utilizing the JSON file format [50].

In the MVP, the data the Angular application sends to the backend mostly consists of DFD data (e.g. to start analysis), or simple input data which needs additional processing (e.g. CPE selection). To access processed data, the Angular application sends a request to the respective endpoints in the backend. Returned data consists, amongst other things, of found threats or CVE-References, additional specification data like CPE, authorization data, or model data when loading a saved DFD.

Storing data is handled by the Spring boot backend. Data which needs to be persisted, like DFDs during the analysis process, gets saved in a MongoDB. Similarly, data needed by the backend for processing (e.g. CVE-References and threats), or data requested by the user (e.g. CPEs), is loaded from the MongoDB to the Spring boot backend.

3.6.3 Class Structure and Data Transfer

The class structure, though fundamentally the same, differs in its implementation between front- and backend. This is done to minimize the amount of data, which is passed during the communication between the two sub-systems. At any point during the communication process, only relevant data is included in the JSON object – for example, when analyzing a threat model, coordinates are not considered relevant and do not have to be included in the data sent to the backend. To showcase this, the differences and changes in the class structure during each step in the threat analysis process are described in the following paragraphs.
When starting the analysis in the frontend, each object representing a DFD element is converted to a “Data Transfer Object” (DTO), which only contains the data relevant for the analysis.

![Diagram](image)

*Figure 12: Abstracted conversion of a "Process" to a "ProcessDTO". The datatypes of the displayed properties either follow the TypeScript notation or consist of custom interfaces.*

The converted objects are then collected in a single “ThreatModelDTO”. Thus, this object contains all data required for the analysis of the DFD.

![Diagram](image)

*Figure 13: DTO collection in a "ThreatModelDTO" object.*

Next, the “ThreatModelDTO” is mapped to a JSON object and sent to the respective endpoint in the backend, where it is converted to a Java object. The JSON mapping is done automatically by Angular and requires no further logic implementation.

![Diagram](image)

*Figure 14: Transfer of the "ThreatModelDTO" to the backend. The properties of the corresponding "ThreatModel" Java object can be accessed by auto-generated getter and setter methods.*
In the backend, the “ThreatModel” object is analyzed. Found threats are collected in an array of threat objects. These threats objects include the STRIDE data and an array of strings representing the IDs of the affected DFD elements.

This “Threat” array is then sent back to the frontend via HTTP, where it is iterated over. Since each “Threat” includes an array consisting of the IDs of the affected elements, it can get mapped to each original DFD element. After this is done, the analysis process is finished. The threat and vulnerability analysis are discussed further in the next section.

3.6.4. Threat and Vulnerability Analysis

All analysis taking place during the threat modeling process is done in the backend. As briefly discussed in section 3.6.3, during the analysis the backend is passed a “ThreatModel” object including all the DFD elements placed in the frontend. At this point, these elements only contain data relevant to the analysis, such as their properties, type, and id. Since elements have to be connected by a data flow in order to possibly pose a threat, the data flows are iterated over. Elements connected by the data flow in a single iteration are then analyzed for their type and properties. Combined with the properties of the data flow, the resulting dataset is compared to the threats currently defined in our system. Here, threats can be split into two categories:

1. Threats always applicable to elements of a certain type.
2. Threats only applicable to elements with certain properties.

Applying threats are collected in a new array of “Threat” objects, which also include the IDs of the affected elements. When returned to the frontend, these objects can then be mapped back to the individual DFD elements.

The vulnerability analysis is started simultaneously to the threat analysis. While it works in a similar manner as the threat analysis, here the CPE references are not linked to the DFD elements and sent to the backend individually. CPEs are then compared to the CVE-References currently saved in our database, with applying references being collected and returned to the frontend. Because our CVE-Reference database includes over 150,000

Figure 15: “Threat” class diagram.
datasets, returned references are displayed immediately to minimize loading times. For the same reason, threat analysis is done separately from the vulnerability analysis.

### 3.6.5 State Management

When looking at an Angular application as described in section 3.6.1.1, its structure can be compared to the “Model-View-ViewModel” (MVVM) design pattern. It describes the separation between the user interface and the business logic. Applying this pattern, the HTML-file represents the “View”, the TypeScript-file within the component represents the “ViewModel”, and the services represent the “Model” (Figure 16).

![Figure 16: Basic “Model-View-ViewModel” design pattern applied to abstract Angular application.](image)

For complex systems such as ours however, this approach can be disadvantageous when managing data flow between dependent components. One example of a component dependency is a user clicking on the dataflow button in our application, with the goal to connect elements on the drawing board. In this case, not only must the corresponding toolbar button component switch to its highlighted “clicked state”, but the drawing board hast to make sure that placed elements can be highlighted on click. Additionally, elements already placed on the drawing board have to switch to a “non-moveable state”, in order to ensure the user does not accidentally move parts of the model. Programmatically managing the necessary data flow between the affected components can be an error-prone task [51], especially when many such dependencies need to be handled. To mitigate the risk of unmanaged dependencies, we integrate the Redux state management engine [41] in our Angular application. Simply put, instead of managing the data flow with the use of services, Redux “introduces a central data store” [52] containing the current state of the application at all times. The state can be considered a depiction of all data currently used, thus the “single source of truth” [53] for the application. Each component subscribes to its relevant
sub-parts of the state and gets the updated data automatically when that section of the state gets updated. When data within a single component changes, e.g. through user input, this component updates the corresponding part of the state – and thus updates all dependent components. Asynchronous data processing is handled in a similar manner; when data is processed within services, or data is acquired through communication with the backend, the state gets updated when the process is finished, again automatically updating all dependent components (Figure 17).

![Diagram](image_url)

*Figure 17: Basic communication within an Angular application implementing Redux.*

When this concept is applied to the example described earlier, clicking a button on the toolbar updates the part of the state depicting the “clicked state”. Both the drawing board and each element subscribe to that state, and automatically update accordingly. By following this approach, the data flow is managed by Redux, and each component dependency does not have to implement its own communication path. This greatly reduces the boilerplate code needed and reduces the error rate of component dependencies.

### 3.6.6 GitHub Version Control

In order to utilize strict version control, we maintain two GitHub [54] repositories – one for the Angular application, one for the Spring boot backend. GitHub hosts the project files, by (manually) synchronizing local changes to the code basis with the hosted repositories. As
such, the online repositories represent the current development state of our project. Following benefits come with the utilization of GitHub as version control system:

- **Documentation.** In the online repositories, the project and its dependencies can be described.
- **Backup.** Because project files are available locally as well as stored online, the risk of losing important work is mitigated.
- **Tracking of changes.** Especially when multiple people are collaborating on a single code base, keeping track of project iterations and changes within them is of great help.
- **Integration.** GitHub repositories can be integrated into many cloud based hosting platforms, such as Heroku [55]. This is discussed further in section 3.6.7.

*Figure 18: OVVL web application GitHub [53] repository.*
3.6.7 Hosting and Continuous Integration

Both our Angular and Spring boot application are hosted on the cloud platform Heroku [55]. We chose Heroku instead of a platform like “Amazon Web Services” (AWS) [56], because Heroku not only provides the required hosting infrastructure, but also manages its configuration, customization and maintenance [57]. Since we are a small team with limited resources, not having to deploy and manage our own hosting system allows us to better apply these resources in other parts of the project. While the cost of hosting full-scale applications is higher on Heroku than on AWS [58], not having to manage the infrastructure mitigates these costs. Additionally, Heroku offers the possibility of free hosting with limited computing resources, and scaling of the infrastructure at any point. This makes it possible for us to react to possible changes in the computational requirements for our system.

By linking our applications to their respective GitHub repositories, Heroku also allows for Continuous Integration (CI). CI describes the development practice where changes in the software are automatically pulled and deployed on the hosting platform, provided the new code basis passes the automated build process [59]. This allows our application’s iterations to directly be accessible online after changes have been made. To make sure only a working application gets deployed, we adhere to the Git workflow [60].

3.7 MVP Overview

One major goal of this thesis was the conception and development of a threat modeling tool MVP, offering the core features needed for threat modeling. It also serves as a showcase of additional features which further improve the threat modeling process. While still a prototype, the way the MVP was developed offers a base that can be built upon, instead of a reference for a future tool developed from scratch. In this section, a brief overview over the core features of the MVP is given.

3.7.1 Implementation of the Conceptual Data Model

The implementation guideline set by the conceptual data model discussed in section 3.5.2 sets the foundation for all the features implemented in the MVP. In its current state, the MVP implements the relations between elements, CPE and CVE-References, the threat model itself, and STRIDE threats. Continuing forward, next custom threats will get implemented, and threats and CVE-References linked to a mitigation status. This mitigation
status then will be linked to an “Issue”, which represents the interface to external project tracking software.

3.7.2 Feature Stack

The following features of the feature list discussed in section 3.4.3 are currently implemented in the MVP:

**Fully Implemented**

1. Architecture Modeling.
2. Automatic Threat Analysis.
3. DFD Element Subtypes.
4. DFD Element Properties.
5. Linking CPE to DFD Elements.
6. CVE Vulnerability Analysis.
7. Vulnerability Ranking.
Partly Implemented

1. Saving and Loading Threat Models.
2. Model Hierarchy.

The foundation for saving and loading threat models is already implemented, with threat models being saved during the analysis. Since user accounts are not yet implemented, these models cannot be linked to an individual workspace and thus not loaded again.

Because the ability to zoom in and out of the DFD in OVVL is implemented, model hierarchy is partly implemented – though currently, element sizes are not set by the zoom level, making an actual hierarchy not possible yet.

3.7.3 MVP Feature Showcase

DFD Creation

When creating a threat model, a user is first presented by a blank working space. This is the area where the DFD is drawn on.

To place DFD elements, a user is able to drag and drop the three main elements – interactors, processes and data stores – from the left toolbar onto the drawing board. The number of elements is not limited to allow for the visualization of complex systems. After selecting the data flow option in the toolbar, placed elements can be connected through data flows. Once placed, elements can be moved around freely.
To allow for the creation of complex DFDs, a user is able to zoom in and out relative to their mouse position within the drawing board.

**DFD Element Specification**

Placed elements can be specified further to better represent their role within the actual system. This specification is different for each of the three DFD elements and includes the element type and element specific properties. While these properties do not have to be set to perform a threat analysis, they do influence the analysis process.
When updating the specification of DFD elements, their name and type get updated in the view.

**CPE Selection**

Elements can be specified further by choosing their software make in form of a CPE. When choosing a CPE, a user is supplied with additional URL-References linked to the specific CPE.
**Threat Analysis**

A user is able to start the analysis process at any point during the DFD creation. The results of the analysis are displayed in a separate view. In this view, the individual elements cannot be moved anymore. The design- and analysis-view can be switched between at any time.

![Figure 26: Found STRIDE threats within an arbitrary system in OVVL.](image)

When selecting a found threat, the corresponding elements are highlighted, and the user is presented with additional information.

![Figure 27: Highlighting of DFD elements and displaying of additional information after a STRIDE threat has been selected in OVVL.](image)

**Vulnerability Analysis and Ranking**

In the analysis view, found CVE-References – corresponding to the CPEs selected – are displayed as well. The list is ranked by their CVSS impact score, with the border color visualizing their impact; With a CVSS impact score of less than 4, the list entry has a green border, between 4 and 7 a yellow border, and higher than 7 a red border.
When selecting a CVE-Reference, the corresponding element is highlighted, and the user is presented with additional information. This includes a short summary of the vulnerability, its publishing date, its impact score, and a URL linking to more information.

Figure 28: Found CVE-References ranked by their CVSS impact score in OVVL.

Figure 29: Highlighting of a DFD element and displaying of additional information after a CVE-Reference has been selected in OVVL.
4. Case Studies

OVVL can be used to analyze any system which can be visualized by a DFD. To better evaluate OVVL’s workflow and how it can aid in the development process of software systems, two case studies were conducted. In these case studies, both an online ordering system and an IoT system are decomposed into their respective assets and analyzed.

4.1 System X

System X is an arbitrary, full stack online ordering system, with the goal of “providing a generic software infrastructure retailers can use to realize their specific online shopping use cases” [61]. In its specification, the following top level use-case scenario is given [61, p. 2]:

- Customers will choose an item in an online catalogue.
- The item is added to a purchase.
- Customers will submit a purchase and supply payment details
- Once payment data is verified by System X, the warehouse will be instructed to ship the item.
- Customers can maintain their personal details and view their order history.

To meet the use-case requirements even under heavy load, System X’s is designed as a distributed system. Its specification describes the following technical deployment [61, p. 8]:

*J2EE web application with load balancing and clustering which shows specific server instances involved.*

*Incoming HTTP requests are first processed by Apache web server. Static content such as HTML pages, images, CSS, and JavaScript is served by the web server. Requests to JSP pages are load balanced and forwarded to 2x2 Apache Tomcat servers using both vertical and horizontal clustering.*

*All 4 instances of Apache Tomcat servers save/receive data to/from a single instance of Oracle 11g DBMS, which could become a performance bottleneck if web application is data-intensive.*

Additionally, the specification provides a visual representation of System X’s deployment architecture. Here, the individual layers are showcased, including their respective hosting platforms and software versions.
Given this technical specification, a DFD can be constructed in OVVL. In order to get a general idea of the threats a software system like System X might face, the system does not have to be modeled on its lowest level. Still, the DFD must include the systems core assets.

System X’s DFD modeled in OVVL can be found in Appendix B.
System X’s specification offers additional information, which can be included in the respective DFD elements to make the analysis more precise. In this case, this data includes element properties and CPE.

![Figure 32: Set properties and selected CPE in the Apache Web Server specified in System X's documentation.](image)

After the DFD is completed, the analysis is started. Given the current concept state of System X, several threats are applicable. While some of these can be disregarded for not being of major concern – e.g. internal communication with a load balancer – several need to be addressed when implementing the actual system; System X’s main point of weakness are the interface between the customer and the web application, as well as the database.

![Figure 33: Highlighted spoofing threat in System X.](image)
Additionally, several vulnerabilities are found for each component specified by a CPE. These vulnerabilities all have a medium to high impact score, which makes their mitigation a major concern. Especially the Oracle database is a weak link in the system, with it being affected by two vulnerabilities with an average impact score of 8.3. All in all, 6 vulnerabilities are found – two affecting the Oracle database, two affecting the Apache HTTP Server and two affecting each Apache Tomcat Server.

![Figure 34: Oracle Database affected by two vulnerabilities.](image)

**System X Discussion**

After reviewing the found threats and vulnerabilities, several suggestions for the improvement of System X can be made:

- The communication between the web application and the web server should be encrypted utilizing HTTPS, to make sure user credentials can not be intercepted.
- Proper authentication and authorization protocols must be implemented, to make sure only authorized users can access sensitive data.
- While the vertical and horizontal clustering allows for adequate load balancing, the single database constitutes a weak link. Fall-back databases should be integrated in order to lower the threat of denial of service attacks.
• User input sent by the web application needs to be sanitized to prevent the realization information disclosure and tampering threats through SQL injection.

• Chosen software versions must be patched in order to prevent the exploitation of the vulnerabilities affecting the system. If no patch is available, steps suggested by the advisories and solutions linked in the CVE-Reference should be taken to mitigate these issues.

4.2 IoT System

In this section, an arbitrary “Internet of Things” (IoT) System is analyzed. The system consists of several IoT devices measuring crowd density in a busy area throughout the day. Data collected by these devices is sent through a gateway to an IoT hub, which is connected to a central backend processing system. This processing system stores the acquired data in a database. A machine learning component analyses this data and feeds data back to the central processing unit. The processed data is then made available through different API endpoints, which are used by a web application. Users with admin access are able to log into the web application to access the central processing system for debugging purposes.

The development of this fictitious system is done with two main goals in mind:

• Making crowd data available to users to help them decide the best time to enter a busy area.

• Training a machine learning algorithm to make predictions about crowd behavior, in order to help managing traffic lights and public transportation in the future.

Since this system is in the early planning stages, no technological deployment has been decided yet – though the two lead developers have a possible software stack in mind. To visualize their concept, they decide to design the architecture in OVVL.
After creating a first architecture concept and running the analysis, several threats are found. Because the core of the backend is a closed system, the main area of concern is the communication between the devices and the IoT hub, as well as the communication between the user and central processing component. Here, OVVL warns of the possibility that the gateway may be impersonated and used to send incorrect or malicious data to the backend system. One threat in the frontend is a user being able to elevate their privilege by acquiring unauthorized login data and entering it in the web application.

**IoT System Discussion**

While the unauthorized acquisition login data is a threat in any system where a login process takes place, the development team decides to change their approach and remove the login feature from their concept, making endpoints not able to receive data – instead, they integrate an admin panel within the closed parts of the system. As OVVL suggests, they also note down to implement an extensive authentication protocol between the IoT devices and the backend system. Additionally, OVVL suggests implementing an extensive logging system in order to minimize the risk of repudiation.

No part of the software stack initially chosen by the development team – proprietary IoT devices, a Spring Boot backend, MongoDB database and an Angular web application – is
linked to any CVE-References. Thus, the chosen software can be used in their current version without any additional patches.

### 4.3 Discussion of the Case Studies in Respect to the Initial Requirements

The conducted case studies showcase how OVVL responds to the initial requirements, which we derived from the current threat modeling state, as discussed in section 3.4.1. To recapitulate, these requirements were:

1. A threat modeling tool should allow for the visualization of complex communication systems through DFDs in an intuitive, visually appealing way.
2. It has to be possible to analyze constructed DFD for both threats and vulnerabilities.
3. DFD elements and threats must be customizable.
4. Storing threat models both online and offline has to be possible.
5. Found issues, mitigation status and model iterations must be traceable.
6. Multiple users must be able to work on a single threat model.

As the case studies show, OVVL’s MVP realizes some of these requirements already. Even very complex systems such as System X (Section 4.1) can be visualized in a timely manner. By providing the feature of zooming in and out of a DFD, a threat model is not limited by the size of the system to be constructed. Additionally, the possibility of customizing DFD elements by further specifying their properties as well as their type and CPE, increases the possible system complexity further. The analysis of software vulnerabilities is a very helpful feature, allowing further threat model specification. Also, while always subjective, we feel confident in saying that – in its current design iteration – both the creation of DFDs as well as displaying threats is more appealing and intuitive in OVVL than it is in Microsoft’s TMT.

While the tool in its current form can be used to gain a general overview over a software system’s security properties, it is limited in some areas. During the utilization of OVVL in the case studies it became clear, that, while already a useful feature, defining complex systems by zooming in and out of the DFD has to be improved further by allowing for the specification of sub-components. This would allow for a more accurate system representation, thus improving a threat model further. Additionally, we noted that the threat data currently available needs to be fine-tuned. In its current form, the analysis sometimes applies threats to their respective DFD segments which are very far-fetched, or
too broad in their definition. In addition to further fine-tuning of our threat definitions, realizing the requirement of custom threat definition will mitigate this issue.

When it comes to its purpose of accompanying actual software projects, some crucial requirements are still missing. In order to fully integrate OVVL into the development lifecycle, storing threat models both locally and online, as well as allowing the collaboration of multiple users, must be possible. Additionally, the requirement of tracing model iterations, mitigation status and found issues must be met.

By highlighting the importance of the set requirements, the conducted case studies aid in the development process going forward. Currently, the MVP serves its purpose of showcasing what the full product will offer, by making it possible to gain a general overview over the security aspects of a software system. It constitutes a solid foundation on which future work can build upon. The continued development is discussed further in section 5.

4.4 Modeling the Case Studies in Microsoft’s TMT

To further analyze how OVVL’s implementation responds to the requirements, we also modeled both case studies in Microsoft’s TMT. During the modeling process, we made the following observations:

- Microsoft’s approach of connecting elements by first adding a data flow, then manually connecting each end to the respective elements, is significantly slower compared to OVVL’s automatic approach of connecting elements by clicking them.
- Constructing a DFD of complex systems such as System X (Appendix C) can be challenging, because the workspace in Microsoft’s TMT is limited. Zooming out does not increase the space.

Thus, it took less time to construct a DFD in OVVL when compared to Microsoft’s TMT. Due to its complexity, this time difference was especially apparent in the modeling of System X.

Because OVVL implements the same analysis methodology as Microsoft’s TMT, both tools report the same number of threats (Figure 36). Thus, both tools offer the same extent of threat detection. The additional vulnerability analysis offered by OVVL is not available in Microsoft’s TMT.
Figure 36: Threat analysis of System X in OVVL (left) and Microsoft's TMT (right) reports 62 threats in both tools. Both DFD's set the same properties.
5. Discussion

Integrating threat modeling into the development lifecycle can be of great help in developing secure software system. Unfortunately, its integration is often limited by a lack of extensive tool support. The following discussion will reflect on this issue in context of the main goals set out by this thesis: Analysis of the current threat modeling state and the development of the new threat modeling tool OVVL.

Threat Modeling

Threat modeling is the concept of deconstructing a software architecture into its basic components and analyzing the communication flow of these components for threats relating to IT security. A threat model can be built for any system which can be broken down into a DFD, analyzing it in regard to its IT security properties. The fact, that 50% of security flaws within software systems arise during their designing phase [12, p. 53], makes threat modeling especially useful in the early stages of development. But it is not limited just to the designing phase of a project – developing threat model iterations parallel to the development lifecycle reveals the mitigation status of initial design flaws and possible security issues arising from earlier mitigations.

Current State of Threat Modeling

The current state of threat modeling does not seem to reflect the scope it should utilize, leaving many demands unmet [32, p. 13]. While several enterprise applications offering threat modeling support exist, their pricing options constitute a barrier of entry especially for smaller development teams. This is due to the fact that arising issues within software systems are often erroneously not put in causal dependency to their architectural design [62, p. 5], and thus enterprise threat modeling software might not be seen worth their price tag. Thus, it might make more sense for smaller software projects to utilize one of the freely available tools.
Microsoft TMT

Here, Microsoft’s “Threat Modeling Tool” is the contender offering the widest feature palette. While a great tool, it is lacking in some areas. One major drawback is a missing focus on modern design and layout. This might confuse a user, thus not only making it hard to create threat models in a timely manner, but also to present an architectural analysis to other team members. When viewed from a functional standpoint, the lack of custom threats, mitigation status and version tracking might aggravate to justify using it as a tool accompanying the development lifecycle. Last but not least, it being dependent on the windows platform is an additional drawback.

OVVL

Based on this, two assumptions can be made: First, the demand for free threat modeling tools is not satisfied. Second, a new tool must offer functionalities similar to existing ones and improve on certain areas. Following this train of thought, in the context of this thesis a concept for a new threat modeling tool – called OVVL – was designed. It includes a general design and UX concept, as well as a feature backlog. While in the future a tool implementing the full extent of the concept will be built, the MVP of OVVL developed during the course of this thesis offers a first impression on how this tool will improve the threat modeling state and rival existing tools. The MVP couples a modern design approach with the core features required for threat modeling: DFD creation, DFD element properties and threat analysis. It is developed in form of a web application, thus offering platform independence.

CPE and CVE-References Feature Discussion

Compared to existing solutions, the MVP also goes a step further in offering the specification of software makes and analyzing them for their respective vulnerabilities. While looking up software makes and corresponding vulnerabilities already was possible externally through the search engine offered by the NVD [20], our approach offers a decrease of loading times of about 83% for CPE queries and about 71% for CVE queries (Table 3)\(^6\).

\(^6\) A detailed breakdown of the loading times is shown in Appendix B. It must be noted that the loading times shown in Table 3 include the content download time. It differs by about 200 milliseconds between requests of OVVL and the NVD. This is due to the fact that in OVVL queries a JSON object is returned, while queries in the NVD return a HTML document.
The fact that the NVD only returns 20 results at a time, with further requests being sent out for additional results, could be considered an even bigger decrease in loading times in OVVL.

Table 3: Comparison of the loading times of both the “Windows 10 64-bit” CPE and the derived CVE search query between OVVL and the NVD [20]. Values are displayed in milliseconds (ms) and rounded to the nearest integer.

<table>
<thead>
<tr>
<th></th>
<th>OVVL</th>
<th>NVD</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPE Query</td>
<td>279 ms</td>
<td>1620 ms</td>
</tr>
<tr>
<td>CVE Query</td>
<td>448 ms</td>
<td>1570 ms</td>
</tr>
</tbody>
</table>

Additionally, the tool combines the lookup of vulnerabilities with the linking of software makes to multiple DFD elements in a single workspace context. This significantly reduces the number of steps which need to be taken in order to analyze a software system for vulnerabilities.

Possible Improvements to OVVL

While the MVP can already be considered an improvement to the current threat modeling state, some enhancements need to be made. Still missing is the concept of model iterations and mitigation status of threats, as well as user management and the definition of custom threats. These improvements are somewhat dependent on each other and will be implemented in the months following this thesis.

Utilization of OVVL in Lectures

OVVL is planned to be utilized as part of future teaching activities at Bachelor and Master level at the University of Applied Sciences Offenburg. These teaching activities include areas of IT security and software development. In these areas, OVVL can assist in pointing out the importance of threat modeling and “Security by Design”, as well as in teaching about software architectures and distributed systems in general.

Licensing

In the future, we plan on making OVVL open-source under the Apache License 2.0 [63]. This means, amongst other things, that the tool can be freely used commercially, modified and distributed. An open-source approach allows for additional community development – thus
further improving the tool. While we have not decided on a financial concept, several possibilities are imaginable:

- Continuing development sponsored through third-party funds, possibly including crowdfunding options.
- Offering the basic version of the application for free, while keeping certain features under a proprietary license.
- Offering support and/or consulting work, aiding development projects in integrating OVVL into development lifecycles.
6. Conclusion

By utilizing threat modeling during the design phase of a system, as well as during its development lifecycle, IT security flaws can be mitigated before they arise in a production system. As a result, the demand for tools offering threat modeling support is ramping up, but not addressed adequately by existing solutions. Especially factors such as a limited functionality, platform dependence, or a dated design can make it challenging to justify integrating existing tools into the development process. By analyzing the lacking areas of existing tools in detail, but also taking into accounts their positive aspects, several requirements for a new tool could be derived. An MVP of this new tool, which we call OVVL, was developed during the course of this thesis. Even though in its current form only the features we prioritized the highest are implemented, it still offers the key features needed for threat modeling. By offering a web application, which combines an extensive threat analysis with an additional vulnerability analysis and an intuitive design, we showcase how the lacking areas of the threat modeling state can be filled. The MVP lays the foundation for future development, which we hope will be enhanced further through community involvement, by making OVVL open source. We think that this open source approach, coupled with the wide array of features OVVL will be offering, will make this tool a meaningful contender in the world of threat modeling. With the increasing importance of developing secure software systems, integrating the approach of “Security by Design” as a core concept into the development lifecycle will greatly benefit software projects of any size. By being simple in its structure, yet powerful in its functionality, OVVL will aid in utilizing this mindset. As such, we are hopeful that OVVL will improve the current state of threat modeling.
7. References


Appendix A: Design Guideline OVVL

Threat Model "OVVL"

OWL

Corporate Design
Inhalt

1. Logo
   1.1 Sicherheitsabstände 3
   1.2 Logoverwendung 4

2. Farbdefinition
   2.1 Orange 5
   2.2 Grün 5
   2.3 Blau 6
   2.4 Rot 6
   2.5 Graustufen 7

3. Formdefinition
   3.1 Hauptelemente 8
   3.2 Side- & Topbars 9
   3.3 PopUp Fenster 9
   3.4 Rundungen 10
   3.5 Schlagschatten 10

4. Schriftverwendung 11
1. Logo

1.1 Sicherheitsabstände

Das Logo braucht einen definierten Sicherheitsabstand, damit es in richtigem Maße zum Ausdruck kommt. Diesen Abstand müssen alle umliegenden Elemente einhalten. Der seitliche Abstand definiert sich über die Breite des Logos und entspricht genau einem Drittel der Gesamtbreite. Der Abstand oben und unten definiert sich über die Höhe des Logos und entspricht einem Viertel der Gesamthöhe.

Die einzige Ausnahme hierbei bildet das Logo in Verbindung mit dem dazugehörigen Schriftzug. Für diese Version gibt es zwei vorgefertigte Dateien, welche verwendet werden müssen:
1.2 Logoverwendung

Das Logo sollte immer gleich verwendet werden, um dessen Wiedererkennungswert zu steigern. Hier sind Verwendungen vorgestellt, welche auf keinen Fall umgesetzt werden dürfen:

- Logo zuschneiden
- Logo falsch skalieren
- Logo auf braunem Hintergrund verwenden
- Logos ein Schwarz Weiß verwenden
- Logo mit Umrandung verwenden
- Logo drehen oder schräg stellen
2. Farbdefinition

2.1 Orange

Form
RGB: 222/113/63
CMYK: 0/49/72/13
HEX: #DE713F

Inhalt
RGB: 198/96/44
CMYK: 0/52/78/22
HEX: #C6602C

Umrandung
RGB: 172/81/26
CMYK: 0/53/85/33
HEX: #AC511A

Highlight Form
RGB: 255/116/0
CMYK: 0/55/100/0
HEX: #FF7400

Highlight Inhalt
RGB: 218/81/11
CMYK: 0/63/95/15
HEX: #DA510B

2.2 Grün

Form
RGB: 59/124/102
CMYK: 52/0/18/51
HEX: #3B7C66

Inhalt
RGB: 55/105/70
CMYK: 48/0/33/59
HEX: #376946

Umrandung
RGB: 41/85/52
CMYK: 52/0/39/67
HEX: #295534

Highlight Form
RGB: 44/146/65
CMYK: 70/0/55/43
HEX: #2C9241

Highlight Inhalt
RGB: 55/116/50
CMYK: 53/0/57/55
HEX: #377432
2. Farbdefinition

2.3 Blau

<table>
<thead>
<tr>
<th>Form</th>
<th>Inhalt</th>
<th>Umrandung</th>
<th>Highlight Form</th>
<th>Highlight Inhalt</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB: 70/124/157</td>
<td>RGB: 70/100/144</td>
<td>RGB: 59/87/129</td>
<td>RGB: 35/120/179</td>
<td>RGB: 50/87/159</td>
</tr>
<tr>
<td>CMYK: 55/21/0/38</td>
<td>CMYK: 51/31/0/44</td>
<td>CMYK: 54/33/0/49</td>
<td>CMYK: 80/33/0/33</td>
<td>CMYK: 69/45/0/38</td>
</tr>
<tr>
<td>HEX: #467C9D</td>
<td>HEX: #466490</td>
<td>HEX: #3B5781</td>
<td>HEX: #2378B3</td>
<td>HEX: #32579F</td>
</tr>
</tbody>
</table>

2.4 Rot

<table>
<thead>
<tr>
<th>Form</th>
<th>Inhalt</th>
<th>Umrandung</th>
<th>Highlight Form</th>
<th>Highlight Inhalt</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMYK: 0/83/79/14</td>
<td>CMYK: 0/85/83/28</td>
<td>CMYK: 0/89/90/39</td>
<td>CMYK: 0/89/85/0</td>
<td>CMYK: 0/89/88/19</td>
</tr>
<tr>
<td>HEX: #DC252E</td>
<td>HEX: #B71C1F</td>
<td>HEX: #9B110F</td>
<td>HEX: #FF1C25</td>
<td>HEX: #CF1618</td>
</tr>
</tbody>
</table>
2. Farbdefinition

2.5 Graustufen

<table>
<thead>
<tr>
<th>Element</th>
<th>RGB</th>
<th>CMYK</th>
<th>HEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hintergrund</td>
<td>122/122/122</td>
<td>0/0/0/S2</td>
<td>#7A7A7A</td>
</tr>
<tr>
<td>Sidebar/Scrollbar</td>
<td>100/100/100</td>
<td>0/0/0/61</td>
<td>#646464</td>
</tr>
<tr>
<td>Topbar/Data Flow</td>
<td>90/90/90</td>
<td>0/0/0/65</td>
<td>#5A5A5A</td>
</tr>
<tr>
<td>Trust Boundary/PopUp</td>
<td>135/135/135</td>
<td>0/0/0/47</td>
<td>#878787</td>
</tr>
<tr>
<td>Abdunkelung</td>
<td>0/0/0</td>
<td>0/0/0/100</td>
<td>#000000</td>
</tr>
<tr>
<td>Buttons PopUp/Sidebar</td>
<td>80/80/80</td>
<td>0/0/0/69</td>
<td>#505050</td>
</tr>
<tr>
<td>PopUp/Innen</td>
<td>70/70/70</td>
<td>0/0/0/73</td>
<td>#464646</td>
</tr>
<tr>
<td>Schrift/Button Aktiv/</td>
<td>205/205/205</td>
<td>0/0/0/20</td>
<td>#CDCDCD</td>
</tr>
<tr>
<td>Grauer Button herauskloppt</td>
<td>190/190/190</td>
<td>0/0/0/25</td>
<td>#BEBEBE</td>
</tr>
<tr>
<td>Grauer Button Inhalt</td>
<td>160/160/160</td>
<td>0/0/0/37</td>
<td>#A0A0A0</td>
</tr>
</tbody>
</table>
3. Formdefinition

3.1 Hauptelemente

Die Hauptelemente werden kreisförmig dargestellt. Unter den Elementen wird der Name angezeigt, welcher individuell angepasst werden kann. Die Größe eines Kreises definiert sich über die Größe der Arbeitsfläche. Der Durchmesser eines Kreises ist immer 10% von der Höhe der Arbeitsfläche (siehe Abb. 1)

Auch die Größe dieser zwei Hauptelemente definiert sich über die Größe der Arbeitsfläche. Die Höhe der Symbole entspricht 10% der Höhe der Arbeitsfläche.

Der eingezeichnete Data Flow auf der Arbeitsfläche wird mit einer Transparenz von 70% angezeigt.

**Highlighting**: Wenn mit dem Cursor eines der Hauptelemente angewählt werden sollte, ändert sich neben der Farbe auch die Größe. In diesem Fall entspricht die Höhe eines Symbols 12% der Arbeitsfläche.
3.2 Side- & Topbars

Sidebar Nummer 1 (siehe Abb. 1) enthält die Hauptelemente in der "Design View". Dieses Element hat genau 1,5 mal die Breite der Hauptelemente.

Topbar Nummer 2 (siehe Abb. 1) enthält diverse Buttons mit unterschiedlichen Funktionen. Diese Buttons haben genau den halben Durchmesser der Hauptelemente. Die Topbar ist genau 1,5 mal so hoch wie die darin enthaltenen Buttons.

Sidebar Nummer 3 (siehe Abb. 2) enthält die Threat und Elements Übersicht in der "Analysis View". Dieses Element hat genau 4,5 mal die Breite der Hauptelemente.

3.3 PopUp Fenster

Wenn ein PopUp Fenster erscheint verdunkelt sich der Hintergrund mit der Farbe schwarz und einer Transparent von 80%. Die Breite des PopUp Fensters entspricht genau 60% der Gesamtbreite des Bildschirms, die Höhe des PopUp Fensters entspricht genau 65% der Gesamthöhe.
3.4 Rundungen


3.5 Schlagschatten

Die Topbar, die linke Sidebar und die PopUp Fenster haben den gleichen Schlagschatten. Dieser ist definiert mit:

- Abstand: 5 px
- Farbe: Schwarz
- Winkel: 135°
- Größe: 8 px
- Modus: Abdunkeln
- Deckraft: 40%

Der Schlagschatten der rechten Sidebar definiert sich mit:

- Abstand: 5 px
- Farbe: Schwarz
- Winkel: 45°
- Größe: 8 px
- Modus: Abdunkeln
- Deckraft: 40%
4. Schriftverwendung

Als Schriftart für das Threat Model wurde die Schrift "Comfortaa" von Google Fonts ausgewählt, weil sie mit den abgerundeten Buchstabenenden ins Gesamtbild passt.

Es gibt drei Variationen der Schrift: Light, Regular und Bold

- **Light:**
  
  abcdefghijklmnopqrstuvwxyz
  ABCDEFGHIJKLMNOPQRSTUVWXYZ

- **Regular:**
  
  abcdefghijklmnopqrstuvwxyz
  ABCDEFGHIJKLMNOPQRSTUVWXYZ

- **Bold:**
  
  abcdefghijklmnopqrstuvwxyz
  ABCDEFGHIJKLMNOPQRSTUVWXYZ

"Comfortaa Light" wird lediglich bei Fußnoten oder Texten unter Schriftgröße 10 pt verwendet.

"Comfortaa Regular" ist die Hauptschrift und wird für alle normalen Texten und Elementbeschreibungen verwendet.

"Comfortaa Bold" wird für Überschriften in Textdateien verwendet.
Appendix B: System X Modeled in OVVL

Figure 37: System X (Section 4.1) modeled in OVVL.
Appendix C: System X Modeled in Microsoft’s TMT

Figure 38: System X (Section 4.1) modeled in Microsoft’s TMT.
Appendix D: Comparison of CPE and CVE queries loading times in OVVL and the NVD [20]

Figure 39: Loading time of the CPE query "Windows 10 64-bit" in OVVL.

<table>
<thead>
<tr>
<th>Name</th>
<th>X</th>
<th>Headers</th>
<th>Preview</th>
<th>Response</th>
<th>Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>find?product=Windows%2010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Queued at 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Started at 0:23 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resource Scheduling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Queuing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.23 ms</td>
</tr>
<tr>
<td>Connection Start</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Skilled</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.16 ms</td>
</tr>
<tr>
<td>Request/Response</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Request sent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>59 μs</td>
</tr>
<tr>
<td>Waiting (TTFB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>273.14 ms</td>
</tr>
<tr>
<td>Content Download</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.21 ms</td>
</tr>
<tr>
<td><strong>Explanation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>278.80 ms</td>
</tr>
</tbody>
</table>

Figure 40: Loading time of the CVE query of the CPE of "Windows 10 64-bit" in OVVL.

<table>
<thead>
<tr>
<th>Name</th>
<th>X</th>
<th>Headers</th>
<th>Preview</th>
<th>Response</th>
<th>Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>websocket</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>threat/cpe=cpe/comicrosoft...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Queued at 14.78 s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Started at 14.78 s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resource Scheduling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Queuing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.28 ms</td>
</tr>
<tr>
<td>Connection Start</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Skilled</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.26 ms</td>
</tr>
<tr>
<td>Request/Response</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Request sent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>77 μs</td>
</tr>
<tr>
<td>Waiting (TTFB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>443.54 ms</td>
</tr>
<tr>
<td>Content Download</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.68 ms</td>
</tr>
<tr>
<td><strong>Explanation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>447.84 ms</td>
</tr>
</tbody>
</table>
Appendix D: Comparison of CPE and CVE queries loading times in OVVL and the NVD [19]

<table>
<thead>
<tr>
<th>Name</th>
<th>Headers</th>
<th>Preview</th>
<th>Response</th>
<th>Cookies</th>
<th>Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>results?form=Basic&amp;result...</td>
<td>Queued at 213.25 ms</td>
<td>Started at 214.06 ms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Resource Scheduling</td>
<td></td>
<td>Queueing</td>
<td>0.81 ms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connection Start</td>
<td></td>
<td>Stalled</td>
<td>3.33 ms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Request/Response</td>
<td></td>
<td>Request sent</td>
<td>0.11 ms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Waiting (TTFB)</td>
<td></td>
<td>Content Download</td>
<td>230.40 ms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Explanation</td>
<td></td>
<td></td>
<td>1.62 s</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 41: Loading time of the CPE query "Windows 10 64-bit" in the NVD [19].*

<table>
<thead>
<tr>
<th>Name</th>
<th>Headers</th>
<th>Preview</th>
<th>Response</th>
<th>Cookies</th>
<th>Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>results?form=Basic&amp;result...</td>
<td>Queued at 0</td>
<td>Started at 0.21 ms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Resource Scheduling</td>
<td></td>
<td>Queueing</td>
<td>0.21 ms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connection Start</td>
<td></td>
<td>Stalled</td>
<td>2.02 ms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Request/Response</td>
<td></td>
<td>Request sent</td>
<td>58 μs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Waiting (TTFB)</td>
<td></td>
<td>Content Download</td>
<td>231.91 ms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Explanation</td>
<td></td>
<td></td>
<td>1.37 s</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 42: Loading time of the CVE query of the CPE of "Windows 10 64-bit" in the NVD [19].*
Declaration of Independent Work

I, Tobias Reski, confirm that the work for this Bachelor Thesis with the title: "Conception and Development of a Threat Modeling Tool", was solely undertaken by myself and that no help was provided from other sources as those allowed.

All sections of the paper that use quotes or describe an argument or concept developed by another author have been referenced, including all secondary literature used, to show that this material has been adopted to support my thesis.

__________________________  ________________________
Place / Date                Signature