"Open Weakness and Vulnerability Modeler" (OVVL): An Updated Approach to Threat Modeling

Andreas Schaad and Tobias Reski
Department of Media and Information, University of Applied Sciences Offenburg, Badstraße 24, 77652 Offenburg, Germany

Keywords: Threat Analysis, Architecture, Security, Risk Assessment.

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 throughout development iterations, threat modeling can help to establish a "Secure 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 paper the current state of threat modeling is investigated. This investigation drove the derivation of requirements for the development of a new threat modelling framework and tool, called OVVL. OVVL utilizes concepts of established threat modeling methodologies, as well as functionality not available in existing solutions.

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 distributed systems with security concerns already in mind from the beginning onwards (Security by Design) 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 overall development lifecycle. While many paper-based resources detailing threat modeling processes are available, tools offering automated threat modeling support are few, rarely free, as well as technically lacking in several areas. As a result, integrating threat modeling into the development process appears to project managers and 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 our goal and our efforts so far are documented in this paper. We analyzed existing tools and made suggestions for possible improvements. These observed improvements were implemented in form of a new threat modeling tool we call OVVL – the "Open Weakness and Vulnerability Modeler":

- An open-source web application framework for threat modelling, based on a user centric minimalistic design and color scheme;
- Based on an analysis of existing threat modelling approaches and tools;
- Delivering a tight and efficient integration of public data sources such as the NIST NVD;
- Including a notion of comparing analyzed architecture versions;
- Suggesting more fine-grained mitigations on basis of details a user has about architecture components and software makes;
- Offering a modular architecture to integrate related methodologies such as attack trees or misuse cases;
- Providing initial support for GDPR compliance checks;
- Defining APIs to consume data in the next stages of the secure development process;

Schaad, A. and Reski, T.
"Open Weakness and Vulnerability Modeler" (OVVL): An Updated Approach to Threat Modeling.
DOI: 10.5220/0007919004170424
In *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications (ICETE 2019)*, pages 417-424
ISBN: 978-989-758-378-0

2 BACKGROUND

"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.

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 (Pandit, 2018). By rating their severity and impact, these threats can then aid in the risk assessment process. One possible way to build a threat model (Pandit, 2018) is by applying the following steps (Secodis GmbH, 2018):

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 (Ma and Schmittner, 2016). Sections within the DFD, in which data processing changes its trust level, are visualized in the model as trust boundaries (Ma and Schmittner, 2016; Stavroulakis and Stamp, 2010).
3. Identifying and modeling of all possible threats (e.g. by applying STRIDE as discussed in section 2.2), even if they cannot (yet) be exploited (Myagmar, Lee and Yurcik, 2005).
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 (Hoglund and McGraw, 2004), applying threat modeling before or during the design phase can help finding security issues early in the development process. This ensures that resources 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" (Myagmar and Lee, 2005).

2.2 Stride

STRIDE is a threat modeling approach developed by Microsoft (Kohnfelder and Garg, 2008) and is based on the assumption, that threats software architectures are susceptible to can be clustered (Shostack, 2008). The STRIDE acronym stands for spoofing, tampering, repudiation, information disclosure, denial of service and elevation of privilege and as such defines the threats a system might face (Kohnfelder and Garg, 2008).

Table 1: STRIDE threats applied to DFD elements.

Element	**Spoofing**	**Tampering**	**Repudiation**	**Information Disclosure**	**Denial of Service**	**Elevation of Privilege**
Interactors	X		X			
Processes	X	X	X	X	X	X
Data Stores		X		X	X	
Data Flows		X		X	X	

It must be noted that the 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 (Shostack, 2008); e.g. a data store might not always be susceptible to denial of service attacks, but to repudiation when implementing a log service.

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. Data regarding software makes (CPE) and their corresponding vulnerabilities (CVE) is provided by the NIST Security Database (NVD).

CPE.
CPE stands for "Common Platform Enumeration" and is a "structured naming scheme for information technology systems, software and packages" (National Vulnerability Database, 2018d). 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 (National Vulnerability Database, 2018b):

$$cpe:2.3:o:microsoft:windows\_10:1607:*:*:*:*:*:x64:*$$

For each CPE, known software vulnerabilities can be found in the form of CVE-References ("Common Vulnerabilities and Exposures" (MITRE Corporation, 2018)).

CVE.
CVE is a standardized system for referencing known software vulnerabilities (National Vulnerability Database, 2018a). 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" (FIRST.org Inc., 2018) score. A CVE-Reference might be named in the following way (National Vulnerability Database, 2018b):

$$CVE-2018-8505$$

CVSS.
CVSS "provides a way to capture the principal characteristics of a vulnerability and produces a numerical score reflecting its severity" (FIRST.org Inc., 2018). 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 Current Threat Modeling State

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" define the current threat modeling state the following way (Brown-White, 2017):

- 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.

2.5 Existing Tool Support

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 our work, we performed an extensive analysis of the free existing tools, focusing on their features and on their user experience (UX) design. By doing so, we aimed 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 tools available. Microsoft's TMT and the OWASP ThreatDragon.

Microsoft Threat Modeling Tool (TMT).
Microsoft provides a free tool in the form of a Windows desktop application. Its threat analysis is based on the STRIDE methodology. TMT's main features include (Microsoft Threat Modeling Tool 2016):

1. Extensive documentation.
2. Creating DFDs manually.
3. Setting properties of DFD elements and adding custom properties.
4. Listing potential STRIDE threats following an automated DFD analysis.
5. Creating custom threats.
6. Manually prioritizing threats.
7. Exporting a CSV file of found threats.
8. Creating a threat report in form of a HTML file

As such, TMT covers most use cases related to threat modeling, but it is lacking in several aspects. One main 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. More precisely, even if the architect already knows about which technology he will or has used, they cannot apply this knowledge within the context of TMT.

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 (Stack Overflow, 2018). 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 formally defined UX acceptance criteria, we were able to note that Microsoft's TMT design might seem dated to some users. Especially when using TMT for the first time, an overload of information can be off-putting – here, a "Getting Started Guide" might aid a user in the initial threat modeling process. The threat modeling process itself seems to be, in our opinion, mostly well thought out. TMT facilitates the placing of DFD elements by Drag & Drop, which to us feels like an intuitive approach. The way TMT links found threats to their respective elements visually by highlighting them, aids in understanding a threat model better. When it comes to TMT's layout in general, we noticed a few lacking areas. Here, the way different DFD elements and their sub-components are displayed in one long, by default unfolded list, can make it challenging to find certain elements during the modeling process. This is further reinforced by the boxed-in nature of the working area, where different settings and a magnitude of information seems overwhelming. Instead of keeping a logic separation of the different element types visually using icons, TMT's approach is a focus on text description – again making it harder to get an overview over the current process. Lastly, it can be challenging to create very complex systems, because TMT does not allow the linking of multiple diagrams in one project file.

OWASP Threat Dragon.
As described in its documentation (OWASP, 2018), 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 (OWASP, 2018):

1. Creating DFDs manually.
2. Setting properties of DFD elements.
3. Adding custom STRIDE threats.
4. Manually prioritizing threats.
5. Linking threat models to GitHub Repositories.

Being open source and platform independent, Threat Dragon is a tool that could potentially be integrated in most development lifecycles without much effort, however we are not aware of any such efforts. Its design and implementation aids in giving a clear overview over architectures of varying complexity.

Threat Dragon's focus on community driven development 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 (Goodwin, 2018), its ambitious goals defined in the roadmap still seem far away from completion (OWASP, 2018). In its current state, Threat Dragon is still missing several key features for it to be considered a valid tool for threat modeling.

3 OPEN WEAKNESS AND VULNERABILITY MODELER

We have developed a new open-source framework and tool called "OVVL - Open Weakness and Vulnerability Modeler" to facilitate the integration of threat modeling into the development lifecycle for software teams of any size. Its core functionality is derived from the prior analysis of the current state and existing solutions.

3.1 Core Data Model

The conceptual data model, as depicted in Figure 2, 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. As such, the conceptual data model provides an overview over the tool's core functionality and also serves as an implementation guideline.

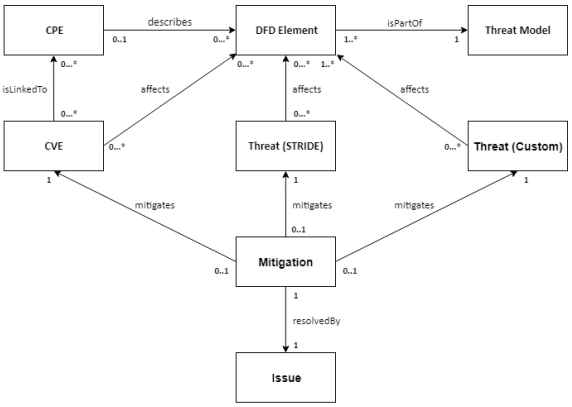

Figure 1: OVVL Conceptual Data Model.

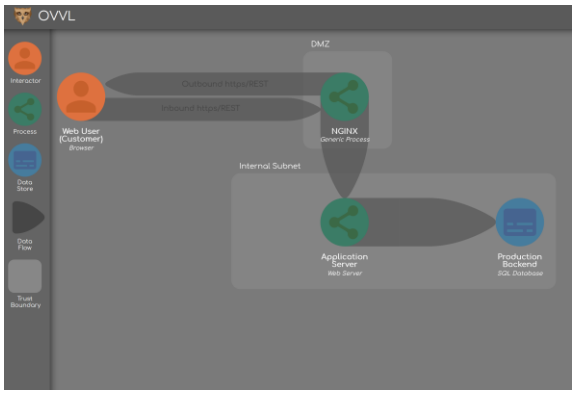

Figure 2: Basic view of a system in OVVL.

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”. 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.

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 provided with a list of matching CPE’s, which she can then 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. This is a feature which clearly distinguishes our approach from the analyzed threat modelling solutions.

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 (Schaad and Borozdin, 2012). OVVL distinguishes between CVE-based, STRIDE-based 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.

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 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 (Atlassian, 2018) or FogBugz (FogBugz, 2018).

3.2 Threat Analysis

During the analysis process, our system iterates over the data flows and thus over each element connection. Properties of the elements are considered, and matching threats are collected and returned to the user. Generally, the threat definitions can be split into threats always applicable to elements of a certain type and threats only applicable to elements with certain properties.
Currently, our threat definitions are the same as in Microsoft’s TMT and are based on a modified

Figure 3: Adding details to a model element.

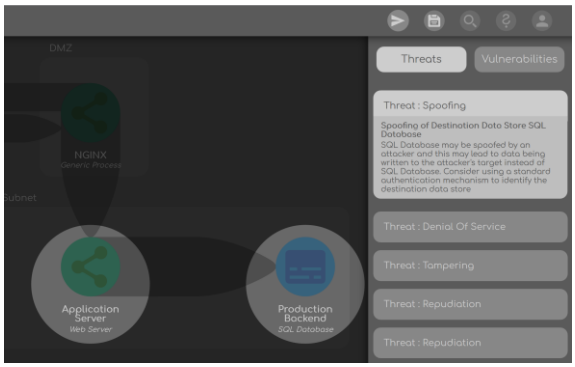

Figure 4: View of a spoofing threat found in OVVL.

STRIDE methodology. Our implementation structure allows for the creation of new definitions, as well as the integration of custom threat definitions set by a user in the future without much effort. The vulnerability analysis works in a similar manner; CPEs are iterated over and compared to the available CVE-References. Because the size of our CVE dataset and the resulting lookup duration, found threats and vulnerabilities are returned to the user separately.

3.3 Data Protection

When performing threat analysis with OVVL, we can use the tool to also make an architect aware of requirements on a system such as stipulated in the GDPR (GDPR, 2018). Here OVVL offers to label data flows as containing personal data as well as mark backend systems that store or process personal data. OVVL is not designed to perform a full privacy impact analysis, however data already gathered in OVVL could be used further in the tool chain.

3.4 Related Modelling

As we initially indicated, our framework and tool also support additional security modelling techniques that could be applied in the early stages of the development process. One such technique is that of attack trees which are hierarchical, graphical diagrams that show how low-level hostile activities interact and combine to achieve an adversary's objectives - usually with negative consequences for the victim of the attack. Another technique is that of misuse case diagrams (Figure 5), which can be thought of as inside-out use cases. They aim at capturing features that should not be implemented in a system and offer another viewpoint of the system to manage security requirements.

The OVVL framework and its conceptual model provide the possibility to create such attack trees and misuse cases on basis of the already defined model elements as part of the core threat modelling functionality.

3.5 OVVL in the Development Tool Chain

OVVL is flexible enough to forward data gathered as part of the threat modelling process to other tools in the software lifecycle toolchain.

One example is that of creating threat modeling related tickets in bug tracking tools such as Jira (Atlassian, 2018) or OpenProject (OpenProject, 2018), which is simple to realize by using the APIs provided by Jira. A developer can then clear all the identified threats as part of his development and configuration work.

Though still part of our currently ongoing work, we also plan on offering the integration with tools such as Nessus (Tenable, 2019) for automated vulnerability scanning. Here, we are very confident that we can achieve a high degree of automation. One use case is Nessus automatically loading threat reports generated by OVVL and perform its scans of the staged or operational system on basis of the provided data.

3.6 Technology Stack

Three factors were kept in mind while choosing the technology stack of OVVL: Scalability, support availability, and complexity. As such, we chose Angular as our frontend framework, because its component-based approach allows for a high level of scalability.

In the backend, where most of our data processing is handled, we utilize Spring Boot. Spring boot decreases the configuration time and boilerplate code

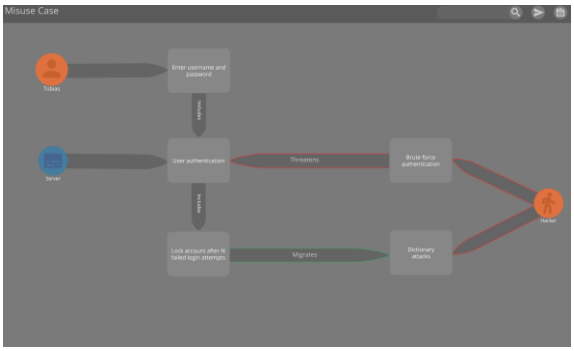

Figure 5: Misuse Case example in OVVL.

immensely, by auto-configuring the core application and its dependencies.

The endpoints provided by our backend are documented and generated by Swagger. Our data is stored in a MongoDB, which allows for easy handling of big datasets, such as the CVE and CPE data.

As hosting architectural data as part of a cloud service may be considered as too security sensitive for certain application domains or projects, we offer to locally host an OVVL instance in form of a virtual machine.

Our code and documentation is available at https://github.com/open-weakness-and-vulnerability-modeler and the OVVL website is https://ovvl.org.

4 EVALUATION

To evaluate the functionality of OVVL, we conducted several case studies. We used the case study already supplied by Microsoft for their TMT tool, as well as two further case studies on typical (cloud-based) 3-tier E-Commerce systems as part of our faculty’s secure software engineering teaching activities.

In its current state, OVVL already facilitates the DFD creation and analysis of software systems of any complexity. This aspect is enhanced by the possibility of defining elements further through the tool’s functionality to set specific properties. Additionally, selecting the software make of DFD elements and analyzing them for their respective vulnerabilities, provides a great insight over potential weaknesses a system faces – at any point during development. When compared to Microsoft’s TMT, OVVL reports the same number of threats regardless of the specified system. We also observed, that the creation of DFD’s is significantly faster in OVVL than in Microsoft’s solution. When it comes to looking up software makes and their corresponding vulnerabilities, our tool offers a significant decrease of loading times compared to the official search engine provided by the NVD itself. Here, OVVL resolves CPE queries about 83% faster and CVE queries about 71% faster.

While in its current form OVVL can be used to gain a general overview over a software system’s security properties, it is still 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 must 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.

5 CONCLUSIONS

Immediate future work will focus on user studies, first focusing on the feedback of our BA and MSc Enterprise Security Students.

As we already observed in (Schaad and Borozdin, 2012), a core problem is that of a too high false positive rate when blindly implementing the STRIDE matrix on a DFD. However, we assume that simple machine learning techniques could help to mitigate this. The training data required for this can be equally extracted from our user studies.

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 existing tools in detail, several requirements for a new tool could be derived. Offering OVVL as a web application, which combines an extensive threat analysis with an additional vulnerability analysis and an intuitive design, we showcased how the lacking areas of the threat modeling state can be filled. We hope OVVL will be enhanced further through community involvement, by making it 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 support this approach. As such, we are hopeful that OVVL will improve the current state of threat modeling.

REFERENCES

Atlassian, *Jira Software.* [online] Available at: https://www.atlassian.com/software/jira [Accessed 29 Dec. 2018].

Brown-White, J. *et al.*, 2017. *Tactical Threat Modeling.* [online] Available at: https://www.safecode.org/wp-content/uploads/2017/05/SAFECode_TM_Whitepaper.pdf [Accessed 11 Dec. 2018].

FIRST.org Inc. *Common Vulnerability Scoring System SIG.* [online] Available at: https://www.first.org/cvss/ [Accessed 29 Nov. 2018].

FogBugz. [online] Available at: https://www.fogbugz.com/ [Accessed 29 Dec. 2018].

GDPR, 2018 [online] Available at: https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules_en [Accessed 18 Mar. 2019].

Goodwin, M. O*wasp-threat-dragon-desktop Master Branch.* [online] Available at: https://github.com/mike-goodwin/owasp-threat-dragon-desktop/commits/master [Accessed 06 Dec. 2018].

Hoglund, G., McGraw, G., 2004. *Exploition Software: How to break code*, Addison Wesley.

Kohnfelder, L., Garg, P., n.d. *The threats to our products*, Microsoft Foundation.

Ma, Z., Schmittner, C., 2016. Threat Modeling for Automotive Security Analysis. In *Advanced Science and Technology Letters Vol. 139*, pp. 333–339.

Microsoft Threat Modeling Tool 2016: Getting Started Guide, n.d. Microsoft Corporation.

MITRE Corporation, n.d. *CVE - Common Vulnerabilities and Exposure.* [online] Available at: https://cve.mitre.org/ [Accessed 28 Nov. 2018].

Myagmar S., Adam J. Lee A., Yurcik W., 2005. Threat Modeling as a Basis for Security Requirements. In *IEEE Symposium on Requirements Engineering for Information Security.*

National Vulnerability Database, 2018a. [online] Available at: https://nvd.nist.gov/ [Accessed 29 Nov. 2018].

National Vulnerability Database, 2018b. *CPE Summary.* [online] Available at: https://nvd.nist.gov/products/cpe/detail/334460?keyword=windows+7+64+bit&status=FINAL&orderBy=CPEURI&namingFormat=2.3 [Accessed 29 Nov. 2018].

National Vulnerability Database, 2018c. *CVE-2018-8505 Detail.* [online] Available at: https://nvd.nist.gov/vuln/detail/CVE-2018-8505 [Accessed 14 Nov. 2018].

National Vulnerability Database, 2018d. *Official Common Platform Enumeration (CPE) Dictionary.* [online] Available at: https://nvd.nist.gov/products/cpe [Accessed 14 Nov. 2018].

OpenProject, 2018 Available at: https://www.openproject.org/de/ [Accessed: 17.03.2019]

OWASP. *OWASP Threat Dragon: Roadmap.* [online] Available at: https://www.owasp.org/index.php/OWASP_Threat_Dragon#Roadmap [Accessed 06 Dec. 2018].

OWASP. *Threat Dragon.* [online] Available at: http://docs.threatdragon.org/ [Accessed 06 Dec. 2018].

Pandit, D. *Threat Modeling: The Why, How, When and Which Tools.* [online] Available at: https://devops.com/threat-modeling-the-why-how-when-and-which-tools/ [Accessed 27 Nov. 2018].

Schaad, A., Borozdin, M., 2012. "TAM2: Automated Threat Analysis". In *SAC '12 Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pp. 1103–1108.

Secodis GmbH. Threat Modeling [online] Available at: https://www.secodis.com/bedrohungsanalysen/ [Accessed 27 Nov. 2018].

Shostack, A., 2008. *Experiences Threat Modeling at Microsoft.* [online] Available at: https://adam.shostack.org/modsec08/Shostack-ModSec08-Experiences-Threat-Modeling-At-Microsoft.pdf [Accessed 29 Jan. 2019].

Stack Overflow, 2018. *Developer Survey Results 2018: Platforms.* [online] Available at: https://insights.stackoverflow.com/survey/2018/#technology-platforms [Accessed 05 Dec. 2018].

Stavroulakis, P., Stamp, M., 2010. *Handbook of Information and Communication Security*, Springer.

Tenable, 2019. *Nessus.* [online] Available at: https://www.tenable.com/products/nessus/nessus-professional [Accessed 17 March, 2019].