JYX JYVÄSKYLÄN YLIOPISTO UNIVERSITY OF JYVÄSKYLÄ

This is a self-archived version of an original article. This version may differ from the original in pagination and typographic details.

Author(s): Honkaranta, Anne; Leppänen, Tiina; Costin, Andrei

Title: Towards Practical Cybersecurity Mapping of STRIDE and CWE : a Multi-perspective Approach

Year: 2021

Version: Accepted version (Final draft)

Copyright: © 2021, IEEE

Rights: In Copyright

Rights url: http://rightsstatements.org/page/InC/1.0/?language=en

Please cite the original version:

Honkaranta, A., Leppänen, T., & Costin, A. (2021). Towards Practical Cybersecurity Mapping of STRIDE and CWE : a Multi-perspective Approach. In S. Balandin, Y. Koucheryavy, & T. Tyutina (Eds.), FRUCT '29 : Proceedings of the 29th Conference of Open Innovations Association FRUCT (pp. 150-159). FRUCT Oy. Proceedings of Conference of Open Innovations Association FRUCT. https://doi.org/10.23919/FRUCT52173.2021.9435453

Towards Practical Cybersecurity Mapping of STRIDE and CWE – a Multi-perspective Approach

Anne Honkaranta, Tiina Leppänen, Andrei Costin
University of Jyväskylä
Jyväskylä, Finland
anne.honkaranta@gmail.com, leppatii@gmail.com, ancostin@jyu.fi

***Abstract*—Cybersecurity practitioners seek to prevent software vulnerabilities during the whole life-cycle of systems. Threat modeling which is done on the system design phase is an efficient way for securing systems; preventing system flaws is easier and more efficient than patching the security of the system later on. Therefore, many Secure Software Development methods include threat modeling as an integral part of the methodology. STRIDE is a popular threat modeling method used by many practitioners. Threat modelers using the STRIDE method work with abstract threat categories, and would benefit learning about the information about current system weaknesses and vulnerabilities. The information is available on the weakness and vulnerability databases (such as the CWE and the CVE). To our knowledge, there exists no mapping between the STRIDE threats and the actual weaknesses and vulnerabilities listed on the databases, thus hindering the effectiveness of the threat modeling and the DevSecOps and Secure Software Development Life Cycle methods as a whole.**

This work attempts to bridge the gap by exploring possible mappings between the STRIDE threats and the CWE weaknesses, with the goal of improving the cybersecurity processes from end to end. The paper explores three different approaches for mapping the STRIDE to the CWE weakness database, and discusses the findings. The paper concludes that the mapping between the STRIDE and the CWE "Technical Impact" and "Scope" elements of the CWE entries is the most prominent for the mapping. Paper also shows that other mappings were challenged by the different conceptual backgrounds between the threats and the weaknesses. The paper also discusses the challenges caused by the inherent vagueness of the items within the frameworks and the CWE and CVE databases, causing that the mappings to these databases remain largely as a manual tasks, which should be carried out by the domain experts.

I. INTRODUCTION

Currently almost all systems are networked together, and people are increasingly dependent on software and its availability on everyday life. It may be harder to exclude oneself from the networked infrastructure than to be part of it. Appliances from toothbrush and fridge to cars and manufacturing systems are all online by default. As the complexity of software systems grows, new vulnerabilities emerge. Increased networking and system complexity stress out the requirement for securing the systems [1], [2], [3]. System security should be managed with a proactive way instead of focusing on putting out fires [1]. According to [3], on H2/2018 there was an increase on the malware by 151 %, and it was estimated that cyber-crime caused damages reach $6 trillion yearly cost by 2021.

Means for tackling the software security are many. Threat modeling "is the practice of identifying and prioritizing potential threats and security mitigation to protect something of value, such as confidential data or intellectual property" [4]. Threat modeling should be started on the early days of the system design, as it is one of the most efficient ways for ensuring the software security [5]. All software should go through sufficient security testing. Yet security testing in practise may be carried out on a light-weighted manner or neglected for shortening the systems time-to-market, or not being considered feasible enough to justify the expenses of the testing [1]. Secure design is not enough, because some vulnerabilities emerge after a long time of use, potentially triggered by an advancement of other software and technology. To patch up, most of the software vendors provide advisories for maintaining the security of their systems.

This paper is organized as follows. Section II introduces the key concepts related to software security and the two compact most-known vulnerability frameworks: OWASP [6] and Seven Pernicious Kingdoms [7]. OWASP is evaluated with relation to the presented concepts, as an example. Section also presents the weakness type database known as Common Weakness Enumeration (CWE) [8], and the Common Vulnerability Enumeration (CVE) vulnerability database [9], both maintained by the Mitre Corporation. Section III describes the STRIDE threat framework and provides an example of using it in a threat modeling task. Section III also describes the three alternative ways for mapping STRIDE threats with the Mitre CWE database items: mapping by using existing OWASP 10 mapping as a mediator, mapping to the CWE Top 25 weaknesses, and mapping to the CWE database by using two elements of CWE database schema (Technical Impact and Scope). Section V discusses and summarizes the findings.

II. RELATED CONCEPTS, VULNERABILITY FRAMEWORKS AND DATABASES

This chapter presents the concepts related to software security, and two compact frameworks of software or code vulnerabilities. Both frameworks are handy for anyone wishing to use brief listings of vulnerabilities. We also present the other

end of the spectrum, i.e. the huge databases containing detailed information about the weaknesses and the vulnerabilities identified on the real-life.

A. RELATED CONCEPTS

While most people are familiar to the concepts of *confidentiality, integrity, availability, vulnerability* and *weakness*, these concepts provide the pillars for the software security.

Confidentiality, Integrity and *Availability* (CIA, or AIC) [10]. *Availability* means that the information is accessible and available to authorized people when needed. Availability assumes that also appropriate security is provided for the information. Hence, availability is more than just letting the information flow; it is about providing the information to the rightful actors, and by appropriate means. It also means that the information should be able to be recovered, if something unexpected happens to the storage. *Integrity* means that information must be whole, original, and reliable. If there are multiple versions of the information, one must be able to identify the original piece of information, as well as the most recent one. *Confidentiality* assumes that the information is kept secret from those who do not have authorized access to it. It also means that information should be kept secret at rest, i.e., when stored, and also at transit, i.e., when information is transferred to a reader, other storage medium, or other format than the original [10].

The CIA triad provides as basis for estimating the severity of the vulnerability. It is used by the Common Vulnerability Scoring System (CVSS) – a framework for measuring the impact of software vulnerabilities [11]. Severity consists of the exploitability of the vulnerability and the impact of its exploitation, which is estimated by using the CIA triad. The value of the final CVSS score is between 0.0-10.0. The main advantages of common scoring system are standardized and application-neutral scores, contextual scoring and transparent scoring framework. The CVSS score does not provide strategies for mitigation [11].

Raggad [12] claims that the CIA triad is not enough, and that it needs to be appended by authentication and non-repudiation to form a so-called security star, which is depicted in Fig. 1.

In the heart of the security star lies *Risk*. Risk is an essential factor striving businesses to focus on security. Business managers are focused on managing risks, not security. Not storing your information securely enough may risk the company brand, or it may cause high fees to the company. For example, if one rudely neglects to keep EU citizen's personal information safe according to the GDPR regulation guidelines, one may end up having up to 20 M€ or maximum of 4 % of annual revenue as an administrative fine [13].

There is an interplay between threat and risk. Harris and Maymi [10] (pp. 6) define threat and risk as follows: 1) "A threat is any potential danger that is associated with the exploitation of a vulnerability"; 2) "A risk is the likelihood of a threat source exploiting a vulnerability and the corresponding business impact".

Fig. 1: The security star (image by Raggad [12])

Hence, *vulnerability* is the weakness that one has utilized to jeopardize the software. And, there is a risk that information confidentiality, integrity or availability is lost, as a whole or partially. Same outcome, i.e., the risk, may be triggered by multiple different threats.

National Institute of Standards and Technology (NIST) [14] defines vulnerability as:

"A weakness in the computational logic (e.g., code) found in software and hardware components that, when exploited, results in a negative impact to confidentiality, integrity, or availability". Hence a vulnerability is the result of exploiting a weakness. In the real world, the concepts become mixed, perhaps due to viewpoint or scope change, or because the difference is not seen as remarkable. If someone wishes to process information by automated means, the concepts must be used in a semantically consistent way.

B. OWASP TOP 10 VULNERABILITY FRAMEWORK

The Open Web Application Security Project® (OWASP) is a nonprofit foundation that works to improve the security of software [6].

OWASP Top 10 Application Security List contains a list of the 10 most common security risks for Web Applications [6]. OWASP is familiar to many and it is commonly used also as a checklist for penetration testers. The OWASP Top 10 is often referred as the list of the Top 10 Web vulnerabilities. From the perspective of the concepts discussed above, it may also be considered as a mix of vulnerabilities and risks.

OWASP Top 10 is interesting for this paper in two ways: 1) It is an example of a compact framework of vulnerabilities. Compact frameworks are needed and they may be used as checklists for security testers. 2) Even if the list is called as

”the 10 most common security risks for Web Applications” it is commonly referred as a list of Web Application common vulnerabilities. Hence, the intertwined relation between risks and vulnerabilities is present on the listing, illustrating the problem with compact vulnerability categories; some of the items are actually risks by their nature, some may be classified as vulnerabilities. The problem is inherent; if one needs to provide a compact list containing the most critical and most commonly identified application vulnerabilities, the classes are deemed to be differing by their grain size and the level of abstraction.

Following list provides three examples of the OWASP Top 10 threats [6] and their characterizations with regard to the concepts of risk and vulnerability.

1) Injection: Examples of injection flaws are SQL, NoSQL and LDAP injection [6]. When untrusted or malicious data is sent to application as a part of command or query the activity is called as injection. Injection is a type of vulnerability, and the risk is that the information is exposed/disseminated or malicious code is run on the system.
2) Broken Authentication: Occurs when an application does not implement correct authentication procedure or session management. The application may reveal passwords, keys, or tokens to the attacker, or to provide the attacker with false identity and privileges on the system. Consequently, attacker may disseminate (loss of confidentiality), falsify (loss of integrity), or destroy information (loss of availability), and run code with false /elevated privileges, leading to potential system unavailability (loss of availability) as an extreme outcome.
3) Sensitive Data Exposure: A risk, which may be caused by weak protection of sensitive information. Vulnerability may come in a form of not encrypting the data at rest or at transit (i.e., storage or traffic is not encrypted to protect the data), and it seems obvious that confidentiality is at risk here.

C. THE “SEVEN PERNICIOUS KINGDOMS” FRAMEWORK

A taxonomy of common coding errors called as the Seven Pernicious Kingdoms is provided by [7]. The framework operationalizes the concepts of phylum (a kind/category of a coding error) and a kingdom (a group of phyla with some shared features between them) from biology.

The authors of the Seven Pernicious Kingdoms emphasize that the taxonomy of coding errors/vulnerabilities is needed for spreading out the understanding of software vulnerabilities within the coding community, and for enforcing security of the code used on novel applications [7]. The authors point out that almost half of all software vulnerabilities may be tracked back to the source code level. They also bring out the need for a simple, compact list of main software vulnerabilities instead of an overwhelming encyclopedia of software bugs. Therefore, the authors have built a taxonomy of software vulnerabilities consisting of 7+1 classes. Why 7+1? Because it has been proven by the psychologists that a human memory is capable to manage information in chunks consisting of 7+/-2 items.

The taxonomy of software vulnerabilities by Tsipenyuk et al. [7] is as follows:

1. Input Validation and Representation; 2. API Abuse; 3. Security Features; 4. Time and State; 5. Errors; 6. Code Quality; 7. Encapsulation + Environment.

D. THE CVE DATABASE

Software vulnerabilities were reported in differing ways by the software and hardware vendors until the MITRE Corporation created a common method and a database for vulnerabilities in 1999 [15]. Since then the Common Vulnerabilities and Exposures (CVE) Initiative has been building the common dictionary and structure for describing software vulnerabilities [9]. The key objective of the CVE is that the name and the description of each vulnerability is defined only once, and in a standardized way in the dictionary. The definitions of vulnerabilities are not provided in detail for a variety of reasons, one of them being to prevent attackers from taking advantage of the descriptions [16].

Each vulnerability is encoded in the following way: CVE prefix, year, and sequence number digits (for example, CVE-2019-1010200). The trademark and copyright of CVE is managed by MITRE Corporation to legally protect use of it and specially to secure a free and open standard [17].

Fig. 2 shows the CVE as ”a link hub” to the related data sources. Additional information about solution, impact level or technical details can be found on other sources like the National Vulnerability Database (NVD) and the Common Vulnerability Scoring System (CVSS) [18].

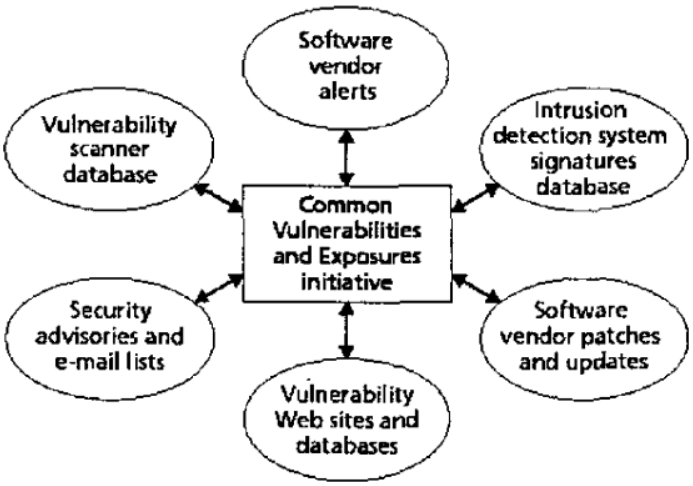

Fig. 2: The CVE is a dictionary which refers to the external databases [15]

CVE is referred as the central of vulnerability and exposure databases. Currently there are 149841 vulnerabilities on the database [9]. The large number of the vulnerabilities identified is an advantage for users, but it also brings out challenges. As new and more complex technologies open new possibilities for attackers, more vulnerabilities are identified, causing the requirements on timeliness and accuracy of the CVE as a master dictionary to be elevated [19].

E. THE CWE DATABASE

Weaknesses expose the systems causing them to be vulnerable to attacks. The Common Weakness Enumeration (CWE) is a classification and a common language for identifying and describing these weaknesses. The CWE list is maintained by the MITRE Corporation and it is available in the Mitre website [14].

The idea of the CWE list is to provide a detailed information of common software and hardware weakness types. Information about the weaknesses has been especially relevant for software developers for avoiding the use of the known types of the security flaws. Recently the CWE categories of hardware have become increasingly important because all sorts of devices include information and network related technology. Categories of hardware weaknesses were added to the CWE in 2020 [8].

The CWE is also one of the repositories that the CVE database refers to. The CWE identifier is linked to each vulnerability thus allowing the vulnerabilities to be navigated by using the CWE ID. According to [20] mapping between the CVE and the CWE items is done manually by domain experts. Manual work is slow and laborous, which effects on the availability of the CVEs for proactive threat mitigation.

The CWE database is presented as a hierarchy of classes where each CWE entity refers to one vulnerability type. There are no limitations on how the weaknesses may be referenced or classified. For example, the CVEs can be mapped to different levels of the classification. This is necessary because of the varying specificity levels of the CVEs [21].

The CWEs are grouped into three hierarchical classifications: Software development, Hardware design and Research concepts. The top level of the Software development and the Hardware design are defined as *classes* which are based on *categories*. *Category* is the root element and a container for the weaknesses that share the similar characteristics. Each category contains *class*, *base* and/or *variant* weaknesses. Classes are like categories: independent of any implementation. The base level weaknesses are more specific, but not as detailed as the variant weaknesses. The research concepts class contains more levels of abstraction than the other classification schemes. Top level elements of the research concepts class are *pillars* which describe *mistakes* but do not specify their impact nor the the exact point affected. The *Pillars* can contain *class*, *base*, or *variant* weakness [22].

III. THE STRIDE THREAT FRAMEWORK AND AN EXAMPLE OF A STRIDE MODEL

Threats may be modeled, analyzed and detected in a numerous ways. For example, the Diamond Model for Intrusion Analysis [23] is based on a long practical experience on intrusion analysis. The Diamond Model provides four views for Intrusion analysis: adversary, infrastructure, capability, and victim. The Diamond Model embeds certain features from the "Kill Chain" analysis model [23]. The Kill Chain model [24] was developed for analyzing Advanced Persistent Threats (APT's).In this method, the phases of the intrusion are analyzed, and mitigation for each of the detected phase of the intrusion are planned. The authors selected the STRIDE method for the analysis for the following reasons: 1) The method is well-known amongst practitioners, because it is a part of the Microsoft's Secure System Development Life Cycle model, 2) The method uses a simple, yet comprehensive classification of threats, and 3) The STRIDE method may be embedded to the actual system design phase, in which the system is designed, and 4) The method was identified as the most mature on a comparison carried out by Shevchenko [3].

This section discusses the STRIDE threat framework and provides an example of the STRIDE threats as identified on a exemplar data flow diagram.

A. THE STRIDE THREAT FRAMEWORK

STRIDE is an acronym that stands for the possible threats towards a system. The STRIDE framework is also used as an integral part of the Microsoft Security Development Lifecycle (SDL) method [25]. The acronym "STRIDE" defines the six threat categories for a system: Spoofing, Tampering, Repudiation, Information disclosure, Denial of service and Elevation of privilege [26]. While the CIA (Confidentiality, Integrity, Availability) triad [10] defines the three pillars for secure systems, the STRIDE presents six threat types relevant to the CIA.

Table I presents the STRIDE [27] threats along with the authors' layman explanation for the acronyms. It also presents a mapping to the CIA triad, thereby mapping the threat classes with the actual risks.

Threat modeling based on the STRIDE threat framework has been studied from several perspectives. For example, [28] pointed out that STRIDE is a popular threat modeling method, but empirical studies about its application are lacking. Therefore, the authors organized a broad empirical study in which 57 students applied the STRIDE threat modeling for a given task. [29] found out that STRIDE is a lightweight, yet efficient framework for modeling threats in systems compromising of critical infrastructures and industrial control systems, which they characterized as complex Cyber-Physical Systems (CPS).

TABLE I: MAPPING OF STRIDE FRAMEWORK ELEMENTS TO THE CIA TRIAD

Threat Name	**Explanation/Example**	**Relation to CIA: what is risked**
Spoofing	Malicious user (or agent) pretends to be someone else, (s)he uses other user's credentials to access the system [27].	Confidentiality, Integrity
Tampering	The content within the targeted system is altered by the malicious external party.	Integrity
Repudiation	Content or system has been misused or tampered, but we cannot prove it due to absence of proof, such as audit trail.	Integrity
Information disclosure	The information is exposed to parties which do not/should not have access to it [27]. Information leak and data breach are common examples of information disclosure.	Confidentiality
Denial of service	System/information is not available to a legitimate user.	Availability
Elevation of privilege	Malicious or rightful user gets more privileges on the content than is entitled to.	Integrity, Confidentiality

B. AN EXAMPLE OF THE STRIDE THREAT MODEL

To illustrate the use of the STRIDE framework, this subsection provides an example of using the STRIDE on the threat modeling phase.

Microsoft [30] (pp. 1) defines the threat modeling as a process that contains five steps:

"1. Defining security requirements, 2. Creating an application diagram, 3. Identifying threats, 4. Mitigating threats, and 5. Validating that threats have been mitigated".

According to the Web Applications Threat Modeling Guideline [25] each of the five steps contain several tasks to carry out. For example, step 1 "Defining security requirements" considers defining security requirements related to the CIA and the business branch in which the software is used. In step 2, numerous diagrams are created, including the data flow diagrams and use case diagrams.

The threat modeling tool [31] provided by Microsoft contains templates for the threat modeler and some templates that can be used for modeling threats for the Azure cloud platform.

Fig. 3 provides an example of a data flow diagram model within the Microsoft threat modeling tool. The diagram is used for studying the STRIDE threats of a system. The data flow diagram is shown on the upper-hand window of the picture. The lower-hand picture shows the STRIDE threat list provided by the tool. The threat modeler first draws the data flow diagram. Once the modeler chooses a component on the diagram, the tool shows related threat categories (system weaknesses) as well as their outcomes, i.e., risks as well as their descriptions on the lower-hand pane. The picture lists all potential threats on the diagram because user has not selected any component of the data flow diagram. Once the threats are detected, the modeler can start planning the mitigation for them, by designing appropriate security controls to be put into appropriate places.

IV. STRIDE-CWE MAPPINGS

This section presents three differing trials that were carried out in order to map the STRIDE threats with the CWE database weaknesses.

The CWE database contains description of 916 weaknesses at this moment [22]. The weaknesses in the CWE database are not so detailed and technical as in the CVE database. Therefore mapping trial from the STRIDE threats was carried out against the CWE database.

A. MAPPING STRIDE AND OWASP TOP10

While the CWE database can be browsed, searched and navigated in many ways, the current version (4.3) of CWE list [22] also offers mappings to following external frameworks: OWASP Top Ten (from year 2017 [32]), Seven Pernicious Kingdoms (7PK [33]), Software Fault Pattern Clusters, SEI CERT Coding Standards, Architectural Concepts, CISQ Quality and Data Protection Measures. These mappings are downloadable in several file formats (HTML, CSV, XML). Each CWE entry referenced from the external mapping has also references to other related mappings. The mappings are defined in the "Taxonomy Mappings" section of the CWE entry. The *Mapped taxonomy name* is an attribute which identifies the framework that was mapped with the CWE (for instance, OWASP Top Ten 2007) and *Mapped node name* contains the entry point used for the mapping (for instance, Injection Flaws). The *Mapping Fit* element contains description of how close the CWE item mapped is to the responding entry in the framework. Possible values are Exact, CWE More Abstract, CWE More Specific, Imprecise and Perspective. These attributes specify more information about the external mapping than just a title. Especially the mapping fit values provide analysed estimation how external entry is equivalent to the CWE entry in case.

External mappings provided by the CWE database may be used for multiple purposes [22]. Top level definition page of each external taxonomy mapping contains an "Audience" section. It contains short descriptions what kind of benefits the mapping offers for different stakeholders. The key benefits of the external mappings for differing audiences may be summarized as:

- Software developers: Tool to ensure that code quality issues are considered throughout the design process, useful view with familiar concepts, weaknesses can be detected using source code analysis tools, a starting point to code more securely and prevent the weaknesses, help for tool acquisition.
- Product customers: A way to ask software development teams to follow minimum expectations for secure code,

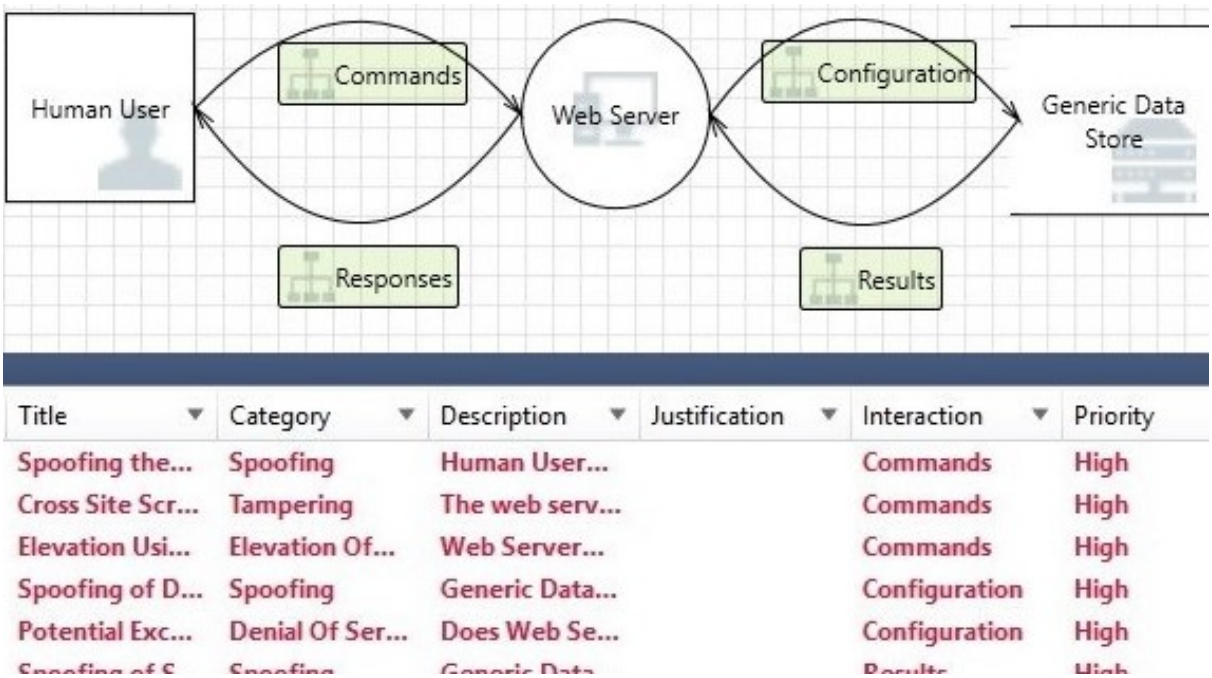

Fig. 3: A screenshot of the STRIDE threat modeling tool [30] in action

a view to requirements that must be met when software developers claim to follow standard in question.

- Product vendors: A view to help understanding code quality issues.
- Assessment tool vendors: A starting point to understand what a software with good code quality is consisted of and possible quality issues.
- Educators: Can be used as training material, multiple ways to create views for different subjects.

An external mapping has three output formats: booklet.html (view), csv.zip or xml.zip. The booklet.html [32] is shown as a tree-like relationships where the top-level items of the selected framework are depicted as categories, which are associated with CWE entries. An example of OWASP Top 10 [6] mapping is shown in Fig. 4.

As depicted in Fig. 4, the framework category name may be mapped to a vulnerability at any level of the CWE category: Category (depicted by letter "C" on a dark red rectangle-shaped background), Pillar, Class (depicted by letter "C" on top of green balloon icon), Variant (depicted by letter "V" on top of lilac balloon shape), Base (depicted by letter "B" on top of blue balloon shape) or Composite (is not shown on this picture). The schema used for external mapping does not seem to limit the CWE entry types to any specific level of the CWE vulnerability category.

The existing mappings [32] can be used as a mediator between the STRIDE and the CWE database. The first mapping trial was carried out by mapping the STRIDE with the OWASP Top 10. The result is shown in Table II.

As shown in the Table II, OWASP vulnerabilities 2, 3, 5, and 10 can be mapped one-to-one with STRIDE threats. A STRIDE modeler may thus find the related weaknesses to Spoofing, Information disclosure, Elevation of privilege, and Repudiation by using the existing OWASP mapping view. By using these mappings, the modeler can also drill down from the direct mappings by using the "booklet.html" tree view, and selecting the related sub-weaknesses, as shown in Fig. 4. Because each of these weaknesses have in turn a description defining the related weaknesses, as well as upper and lower-level weaknesses, a modeler may find lots of related weaknesses and their descriptions.

Fig. 5 represents an example where the STRIDE modeler selected the OWASP vulnerability number 3 "Sensitive Data Exposure" because (s)he wants to study the weaknesses related to the STRIDE threat "Elevation of privilege" by using the mapping. From the tree-like view exposed, (s)he has further selected a related weakness categorized as "Exposure of Private Personal Information to an Unauthorized Actor".

Because the OWASP mapping only provided partial results, two other mapping trials were carried out.

B. MAPPING STRIDE AND CWE TOP 25

"CWE Top 25 Most Dangerous Software Weaknesses (2020)" [34] is a listing prepared by Mitre. It lists the most common weaknesses that were identified on the previous two years. The list is not solely based on the Mitre CWE database; it utilizes the CVE database by Mitre and the National Vulnerability Database, as well as the CVSS vulnerability scoring system as a basis for listing [34].

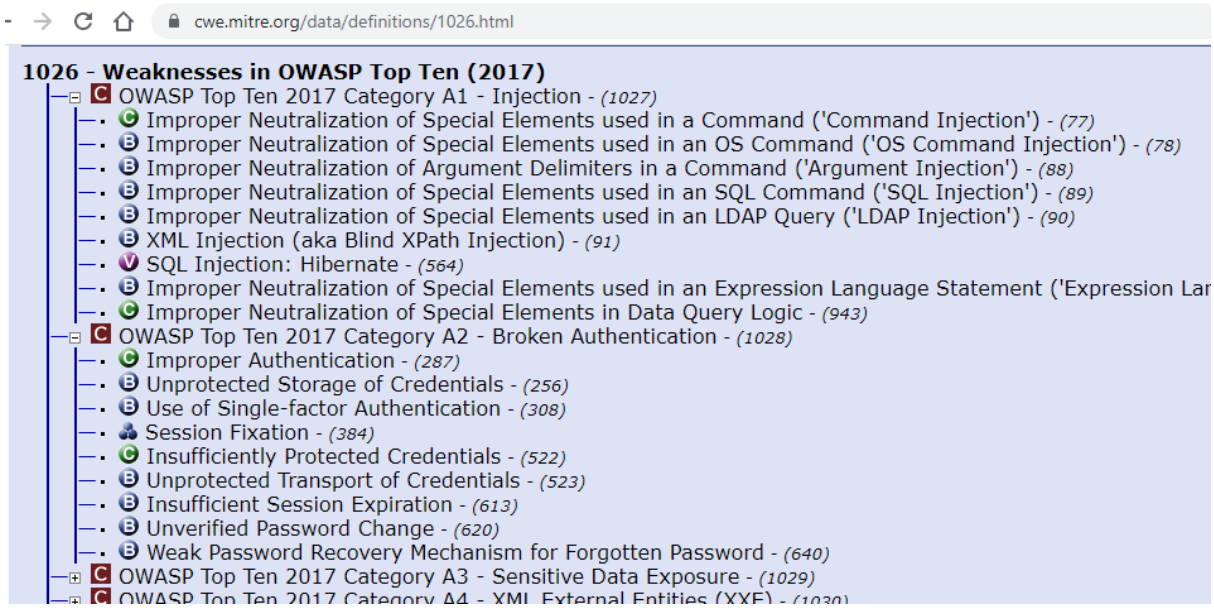

Fig. 4: A snapshot of "booklet.hml" view on OWASP – CWE vulnerability database mapping [32]

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name
ChildOf	C	668	Exposure of Resource to Wrong Sphere
ParentOf	B	201	Insertion of Sensitive Information Into Sent Data
ParentOf	B	203	Observable Discrepancy
ParentOf	B	209	Generation of Error Message Containing Sensitive Information
ParentOf	B	213	Exposure of Sensitive Information Due to Incompatible Policies
ParentOf	B	215	Insertion of Sensitive Information Into Debugging Code
ParentOf	B	359	Exposure of Private Personal Information to an Unauthorized Actor
ParentOf	B	497	Exposure of Sensitive System Information to an Unauthorized Control Sphere
ParentOf	B	538	Insertion of Sensitive Information into Externally-Accessible File or Directory
ParentOf	B	1243	Sensitive Non-Volatile Information Not Protected During Debug
ParentOf	B	1258	Exposure of Sensitive System Information Due to Uncleared Debug Information
ParentOf	B	1273	Device Unlock Credential Sharing
ParentOf	B	1295	Debug Messages Revealing Unnecessary Information
CanFollow	V	498	Cloneable Class Containing Sensitive Information
CanFollow	V	499	Serializable Class Containing Sensitive Data
CanFollow	B	1272	Sensitive Information Uncleared Before Debug/Power State Transition

Fig. 5: Example of navigating the weakness by using the OWASP mapping to find related weaknesses to STRIDE threat "Information Disclosure"

The 2020 CWE Top 25 was built by first obtaining the vulnerability data from NVD database (years 2018-2019). Then analysts designed the complex scoring system which uses many elements, including the vulnerability's CVSS score and the vulnerability's number of occurrences on the weakness category [34].

Mapping from CWE Top 25 to STRIDE was done by scrutinizing each of the Top 25 weaknesses on the CWE Top 25 list with STRIDE elements, one by one. The name of the weakness was not descriptive enough for reasoning about to which element of the STRIDE the item should be mapped to. Each item on the "Top 25 list" was studied by using the weakness enumeration page, which was available by clicking on the hyperlink attached to the weakness name. Technical impact information on the weakness' detail page provided sufficient information for mapping.

Table III shows our mapping between the CWE Top 25 weaknesses (as listed in the CWE Top 25 [34]) and STRIDE.

This mapping shows that all STRIDE threats have related weaknesses in the Top 25 listing. The listing of the Top 25 weaknesses contains weaknesses of differing kinds and scope; for example, NULL pointer reference is clearly a weakness related to coding, and perhaps specific to some of the coding languages commonly in use. It is very specific of its nature. Improper privilege management may be a consequence of poor code, but also an outcome of not protecting the password database. It is clearly broader by its nature, and not so much dependant of a language used for software construction.

The authors of the Top 25 list bring out that during the

TABLE II: MAPPING OF STRIDE FRAMEWORK TO THE OWASP TOP 10 [6]

OWASP vulnerability/risk category	STRIDE Threat Category
1. Injection	Has not direct counterpart, injection can lead most probably to the Elevation of privilege, Information disclosure or Tampering.
2. Broken Authentication	Spoofing (Elevation of privilege)
3. Sensitive Data Exposure	Information disclosure
4. XML External entities	Has not direct counterpart, injection can lead most probably to the Elevation of privilege, Information disclosure or Tampering, perhaps even to Denial of Service.
5. Broken Access Control	Elevation of privilege
6. Security Misconfiguration	Elevation of privilege, and may lead also to Information dissemination, Tampering, and Denial of Service
7. Cross-site scripting (XSS)	Same as (6).
8. Insecure Deserialization	Same as (6).
9. Using Components with Known Vulnerabilities	All STRIDE threats apply.
10. Insufficient Logging and Monitoring	Repudiation

past year the weaknesses related to the Authentication and Authorization have been rising on the listing. Also, a move towards more specific weaknesses has emerged recently [34].

C. MAPPING STRIDE AND CWE'S TECHNICAL IMPACT AND SCOPE

The key to a common language within the CWE database is the common grammar which is defined by the CWE Schema [35]. It represents the CWE data structure which is used for the CWE entries, and it defines several enumerations for describing the attributes for each of the CWE items.

Schema of the CWE provides alternative means for carrying out the mapping between the CWE and the STRIDE. As the mapping from the CWE Top 25 to the STRIDE was carried out, the information in the CWE entry page section titled as "Scope" and "Technical impact" was found beneficial for the mapping. These items were also present at the schema [36]. *Technical impact* defines the anticipated outcome of the weakness and *Scope* is related to the system security requirements. The Scope uses the Security Star as a base, and appends it with *Access Control* element.

Process and method used in the STRIDE schema item mapping was pragmatic. Accurate values of both enumerations were listed from the CWE schema XSD description first [36]. Second, each category of STRIDE was linked with one or more enumerations of the *Technical Impact*. Finally, the relations between the *Scope* and *Technical Impact* were defined by searching the CWE list with "technical impact" as the keyword. Scopes were also checked against the STRIDE categories to finalize the mapping. The result of the analysis is presented in the Table IV.

TABLE III: MAPPING OF CWE TOP 25 WEAKNESSES AND STRIDE THREATS

CWE ID and Name	STRIDE Threat
CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	Elevation of privilege, Information disclosure, Tampering and Denial of Service.
CWE-787 Out-of-bounds Write. Scope: Integrity Availability	Same as above
CWE-20 Improper Input Validation	Same as above.
CWE-125 Out-of-bounds Read	Information disclosure, Denial of Service
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer	Information disclosure, Denial of Service. Possible also: Tampering.
CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	Information disclosure, Tampering, Spoofing, Elevation of privilege.
CWE-200 Exposure of Sensitive Information to an Unauthorized Actor	Information disclosure
CWE-416 Use After Free	Denial of Service, Tampering, Information disclosure
CWE-352 Cross-Site Request Forgery (CSRF)	Spoofing, Elevation of privilege, Information disclosure, Tampering, Denial of Service.
CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	Denial of Service, Information disclosure, Tampering, Repudiation.
CWE-190 Integer Overflow or Wraparound	Denial of Service, Tampering, Elevation of privilege.
CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	Elevation of privilege, Tampering, Information disclosure, Denial of Service.
CWE-476 NULL Pointer Dereference	Denial of Service, Tampering, Information disclosure
CWE-287 Improper Authentication	Elevation of privilege, Spoofing
CWE-434 Unrestricted Upload of File with Dangerous Type	Information disclosure, Tampering. Can also lead to Elevation of privilege, Denial of Service.
CWE-732 Incorrect Permission Assignment for Critical Resource	Spoofing, Elevation of privilege, Information disclosure, Tampering, Denial of Service.
CWE-94 Improper Control of Generation of Code ('Code Injection')	Elevation of privilege, Spoofing, Repudiation.
CWE-522 Insufficiently Protected Credentials	Spoofing, Elevation of privilege
CWE-611 Improper Restriction of XML External Entity Reference	Elevation of privilege, Spoofing, Denial of Service.
CWE-798 Use of Hard-coded Credentials	Spoofing, Elevation of privilege, Information disclosure.
CWE-502 Deserialization of Untrusted Data	Tampering, Denial of Service.
CWE-269 Improper Privilege Management	Spoofing, Elevation of privilege.
CWE-400 Uncontrolled Resource Consumption	Denial of Service, Elevation of privilege.
CWE-306 Missing Authentication for Critical Function	Spoofing, Elevation of privilege.
CWE-862 Missing Authorization	Elevation of privilege, Information disclosure, Tampering, Repudiation.

This mapping provided the best outcome in two ways. First, the concepts of the STRIDE and the Weakness elements were shown to be quite well comparable with each other, they are

TABLE IV: MAPPING OF STRIDE FRAMEWORK TO "TECHNICAL IMPACT" AND "SCOPE" ELEMENTS OF THE CWE ENTRY

STRIDE	**CWE/Technical Impact**	**CWE/Scope**
Tampering	Modify data	Integrity
Information disclosure (privacy breach or data leak)	Read data	Confidentiality
Denial of service	DoS: unreliable execution	Availability
Denial of service	DoS: resource consumption	Availability
Elevation of privilege	Execute unauthorized code or commands	Confidentiality Integrity Availability Access control
Spoofing	Gain privileges / assume identity	Access Control Authentication
Elevation of privilege	Bypass protection mechanism	Access Control Authentication
Repudiation	Hide activities	Non-Repudiation Accountability

"more same calibre" than the STRIDE-CWE pairs identified on the previous mapping trials. Second, this mapping was also easier to do. As a downside, we found out that there is no view available on the CWE for this mapping. This view would be one topic for further development of the CWE.

As an outcome of the trials it may be concluded that the CWE entry's *Scope* and *Technical* impact elements provided the most appropriate and straightforward outcome. As it is not always possible to compare oranges with apples, finding an exact fit between threats and weaknesses was found as a challenging task. The concepts of threat and weakness come from different scenes; one is quite abstract and on a low level of detail, while the other one is more technical and more or less related to a detailed finding on a real-life.

This mapping rehearsal did not cover the relation between vulnerability database (CVE) and STRIDE. Vulnerabilities are exact findings in specific software packages, and threats are quite abstract entities, thus the mapping is perhaps just as difficult as the mapping trial we carried out between vulnerabilities and weaknesses. The NIST vulnerability database provides links from vulnerabilities to CWE weaknesses' "crossings", using the CWE weaknesses as classification mechanism for the vulnerabilities [14]. These links may provide a starting point for those wishing to make mappings from the weaknesses to the related vulnerabilities.

The CWE authors and maintainers themselves had recognized the need to study the inter-operability of the CWE with other resources related to software security. They carried out a study in which two analysis tools and one secure programming reference was chosen, and a trial mapping from CWE to them was carried out [37]. The authors sum up that exact mappings were found for 45,72% of the items. As a summary, the authors denote that mapping to CWE entities is not as straight-forward as one might think [37]. Even though the modest mapping rehearsal carried out by the authors does not provide enough information for statistical analyses, it however supports the finding made by Loveless [37].

V. Conclusion

This paper explored and implemented three different mappings from the STRIDE threats to the CWE weaknesses. We found out that the CWE weakness type (category) names are not sufficient for a novice user to perform an effective mapping. Detailed information about the weaknesses on the corresponding CWE details page was essential for carrying out the mapping. The firts two mapping attempts; the mapping by using the OWASP as a mediator (Section IV-A) and the second mapping to CWE Top 25 weaknesses (Section IV-B), both provided only partial mapping to STRIDE elements. The third mapping from the STRIDE to the CWE schema elements (Scope and Technical Impact) (Section IV-C), was found to provide the most optimal outcome. However, this mapping calls upon further development from the CWE maintainers, because the view related to this mapping is not yet available on the CWE external mappings list.

The authors wish that the findings presented on this paper will help the threat modeling practitioners to identify practical information about weaknesses and vulnerabilities for threat estimation and mitigation. Additionally, the software developers and security researchers may find our mapping trials and related findings helpful for practical and research purposes.

VI. Acknowledgements

Authors wish to thank the anonymous reviewers for their valuable comments and feedback that helped to improve the quality of the paper.

References

[1] G. Erdogan, P. H. Meland, and D. Mathieson, "Security testing in agile web application development-a case study using the east methodology," in *International Conference on Agile Software Development*. Springer, 2010, pp. 14–27.

[2] Y. Ayachi, E. H. Ettifouri, J. Berrich, and B. Toumi, "Modeling the owasp most critical web attacks," in *International Conference Europe Middle East & North Africa Information Systems and Technologies to Support Learning*. Springer, 2018, pp. 442–450.

[3] N. Shevchenko, "Threat modeling: 12 available methods," *URL: https://insights. sei. cmu. edu/sei blog/2018/12/threat-modeling-12-available-methods. html [accessed: 2020-05-24]*, 2020.

[4] J. Brandon, "What is threat modeling and how does it impact application security?" *SecurityIntelligence*, 2019.

[5] W. Xiong and R. Lagerström, "Threat modeling–a systematic literature review," *Computers & security*, vol. 84, pp. 53–69, 2019.

[6] OWASPFoundation, "Owasp to 10 web application risks," *URL: https://owasp.org/www-project-top-ten/ [accessed: 2021-02-24]*, 2017.

[7] K. Tsipenyuk, B. Chess, and G. McGraw, "Seven pernicious kingdoms: A taxonomy of software security errors," *IEEE Security & Privacy*, vol. 3, no. 6, pp. 81–84, 2005.

[8] TheMitreCorporation, "Cwe - common weakness enumeration," *URL: https://cwe.mitre.org/ [accessed: 2021-02-24]*, 2021.

[9] ——, "Cve database," *https://https://cve.mitre.org/index.html [accessed: 2021-02-20]*, 2021.

[10] S. Harris and F. Maymi, *CISSP all-in-one exam guide*. McGraw-Hill Education New York, NY, USA, 2016.

[11] P. Mell, K. Scarfone, and S. Romanosky, "Common vulnerability scoring system," *IEEE Security & Privacy*, vol. 4, no. 6, pp. 85–89, 2006.

[12] B. G. Raggad, *Information security management: Concepts and practice*. CRC Press, 2010.

[13] EU, "Regulation eu 2016/679 of the european parliament and of the council of 27 april 2016," *Official Journal of the European Union. Available at: http://ec. europa. eu/justice/data-protection/reform/files/regulation_oj_en. pdf (accessed 20 September 2017)*, 2016.

[14] NIST, "Nvd, national vulnerability database," *https://nvd.nist.gov/vuln [accessed: 2021-02-20]*, 2021.

[15] R. A. Martin, "Integrating your information security vulnerability management capabilities through industry standards (cve&oval)," in *SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme-System Security and Assurance (Cat. No. 03CH37483)*, vol. 2. IEEE, 2003, pp. 1528–1533.

[16] M. Schiappa, G. Chantry, and I. Garibay, "Cyber security in a complex community: A social media analysis on common vulnerabilities and exposures," in *2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS)*. IEEE, 2019, pp. 13–20.

[17] TheMitreCorporation, "Mitre frequently asked questions," *https://cve.mitre.org/about/faqs.html [accessed: 2021-02-20]*, 2021.

[18] T. Bhuddtham and P. Watanapongse, "Time-related vulnerability lookahead extension to the cve," in *2016 13th International Joint Conference on Computer Science and Software Engineering (JCSSE)*. IEEE, 2016, pp. 1–6.

[19] TheMitreCorporation, "The future of vulnerability management (1/2) - hackuity'.riskinsight, 10 feb. 2021," *https://www.riskinsight-wavestone.com/en/2021/02/hackuity-shake-up-the-future-of-vulnerability-management-threat-status-and-current-issues-in-vulnerability-management-1-2/ [accessed: 2021-02-25]*, 2021.

[20] E. Aghaei and E. Al-Shaer, "Threatzoom: neural network for automated vulnerability mitigation," in *Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security*, 2019, pp. 1–3.

[21] A. Tripathi and U. K. Singh, "On prioritization of vulnerability categories based on cvss scores," in *2011 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT)*. IEEE, 2011, pp. 692–697.

[22] TheMitreCorporation, "Cwe list," *https://cwe.mitre.org/data/index.html [accessed: 2021-02-20]*, 2021.

[23] S. Caltagirone, A. Pendergast, and C. Betz, "The diamond model of intrusion analysis," Center For Cyber Intelligence Analysis and Threat Research Hanover Md, Tech. Rep., 2013.

[32] TheMitreCorporation, "Cwe - common weakness enumeration. cwe view: Weaknesses in owasp top ten (2017). view id: 1026," *URL: https:/cwe.mitre.org/data/slices/1026.html [accessed: 2021-02-24]*, 2021.

[24] E. M. Hutchins, M. J. Cloppert, R. M. Amin *et al.*, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," *Leading Issues in Information Warfare & Security Research*, vol. 1, no. 1, p. 80, 2011.

[25] Microsoft, "Threat modeling web applications," *URL: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff648006(v=pandp.10) [accessed: 2021-02-24]*, 2010.

[26] Wikipedia, "Wikipedia, 2021," *URL: https://en.wikipedia.org/wiki/STRIDE*$_{(security)}[accessed : 2021-02-24]$, 2021.

[27] Microsoft, "The stride threat model," *URL: https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)?redirectedfrom=MSDN [accessed: 2021-02-24]*, 2009.

[28] R. Scandariato, K. Wuyts, and W. Joosen, "A descriptive study of microsoft's threat modeling technique," *Requirements Engineering*, vol. 20, no. 2, pp. 163–180, 2015.

[29] R. Khan, K. McLaughlin, D. Laverty, and S. Sezer, "Stride-based threat modeling for cyber-physical systems," in *2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*. IEEE, 2017, pp. 1–6.

[30] Microsoft, "Threat modeling tool," *URL: https://www.microsoft.com/en-us/securityengineering/sdl/threatmodeling [accessed: 2021-02-24]*, p. 1, 2021.

[31] J. Geib, D. Couter, J. Martinez, M. Baldwin, and B. Keiss, "Getting started with the threat modeling tool," *URL: https://docs.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-getting-started [accessed: 2021-02-24]*, 2017.

[33] ——, "Cwe - common weakness enumeration. cwe view: Seven pernicious kingdoms. view id: 700," *URL: https://cwe.mitre.org/data/slices/700.html [accessed: 2021-02-24]*, 2021.

[34] ——, "Cwe top 25 most dangerous sofware weaknesses," *https://cwe.mitre.org/top25/archive/2020/2020*$_{cwe_top25.html}[accessed : 2021-02-20]$, 2021.

[35] ——, "Cwe database. xsd schema documentation. schema version 6.3," *https://cwe.mitre.org/data/xsd/cwe*$_{schema_latest.xsd}[accessed : 2021-02-20]$, 2020.

[36] ——, "Schema documentation - schema version 6.3." *https://cwe.mitre.org/documents/schema/ [accessed: 2021-02-20]*, 2018.

[37] M. Loveless, "Cwe mapping analysis," *https://cwe.mitre.org/documents/mapping*$_{analysis/index.html}[accessed : 2021-02-20]$, 2008.