Threat Modeling Tools: A Taxonomy

Zhenpeng Shi, Kalman Graffi, David Starobinski, Nikolay Matyunin

2021

Preprint:

This is an accepted article published in IEEE Security and Privacy . The final authenticated version is available online at: https://doi.org/10.1109/MSEC.2021.3125229

Threat Modeling Tools: A Taxonomy

Zhenpeng Shi, Kalman Graffi, David Starobinski, Nikolay Matyunin

***Abstract*—**
Threat modeling tools allow to identify weaknesses in a system design. Yet, understanding conceptual differences between the tools is not trivial. We propose a taxonomy of threat modeling tools, and use it to compare several popular ones. Alongside, we illustrate differences between the tools with a usage example.

1. Introduction

Threat modeling is a structured process for identifying and understanding potential threats, as well as developing and prioritizing mitigations, so that valuable assets in the system can be protected. Performing threat modeling can not only help to identify and mitigate security issues at an early stage, but also guide investment on system security. For instance, the work in [1] presents a case study of threat modeling conducted at New York City Cyber Command, a large-scale and high-risk enterprise environment. The results of the case study suggest that, when properly conducted, threat modeling is effective at the enterprise level and results in positive feedback from the involved participants.

Many threat modeling tools have been developed over the years to help with the threat modeling process. Although the tools share the same overarching goal, they vary significantly in terms of functionalities and workflow. Different tools might be effective for different threat modeling steps. Thus, the most suitable tool depends on the specific use case.

In this paper, we propose a set of criteria for evaluating threat modeling tools. We conceptually classify current threat modeling tools, and discuss their properties and design implications. We then apply the criteria to provide a taxonomy of popular threat modeling tools, and qualitatively evaluate and compare the capabilities offered by the tools. Note that comparing the performance of the tools is difficult, since most tools require manual configurations and design. Hence, we do not compare the effectiveness of the tools, and leave quantitative evaluation to future work.

Our proposed evaluation and comparison of threat modeling tools are based on the following criteria, which are discussed in detail in Section 5.

1) Model form. What form of system model is adopted by the tool (e.g., diagram vs. text-based)?
2) Model reusability. Is the tool able to reuse parts of existing models when building new ones?
3) Threat library. What libraries are used for threat identification? What are the sizes of those libraries?
4) Threat evaluation. How does the tool evaluate the identified threats?
5) Mitigation suggestion. Does the tool suggest effective countermeasures to the identified threats?
6) Availability. How does the tool run, as a desktop/web application or through command-line interface? Is the tool open-source or closed-source?
7) Development stage. Is the tool still in development or in a relatively mature state?
8) Software development life cycle (SDLC)

Published by the IEEE Computer Society

integration. Does the tool offer integration with other development tools regularly used in SDLC (e.g., issue tracking systems)?

We envision that our proposed taxonomy of threat modeling tools will help academic researchers and security practitioners to find the tools most suited for their needs. In summary, our work has the following main contributions:

1) We propose a set of criteria for qualitative evaluation of threat modeling tools. The criteria cover different features and capabilities of the tools.
2) We use the criteria proposed in this paper to compare current threat modeling tools. The comparison results are summarized in a table to help understand the strengths and weaknesses of the different tools.
3) We perform threat modeling for an example use case to investigate in more detail three popular threat modeling tools (i.e., Microsoft Threat Modeling Tool, OWASP pytm, and IriusRisk community edition). We illustrate the workflow of the threat modeling process with each tool, and point out interesting findings.
4) We share insights from the evaluation and comparison of the threat modeling tools.

2. Background

In this section, we provide background on threat modeling, including detailing its main steps. Further, we describe popular methods and knowledge bases that help implement each step in practice.

Threat modeling is a structured process for identifying potential threats and developing mitigations, with the purpose of protecting valuable assets in a system. We define threat modeling as a process consisting of the following six steps:

(i) *Define* security requirements. Security requirements usually include a set of standards to meet and functionalities that are related to the security of the system. The requirements help determine what data assets in the system are the most important, and to what extent those data assets need protection from threats. A reference to help with this step includes the OWASP Top Ten[1], which represents the most critical 10 types of security risks to web applications. Such references provide developers with a quick and effective way to reason about fundamental security requirements that should be satisfied.

(ii) *Model* the system. The system model is a graph that represents system artifacts as nodes, which are grouped in trust zones, and data flows between the nodes as edges. This model can either be written in a text-based form following a certain syntax, or visualized in a diagram form, for example, a data flow diagram (DFD). A DFD is a common way to model system entities, events and boundaries of the system. It consists of four types of elements: processes, data flows, data stores, and external entities.

(iii) *Identify* potential threats based on the system model. Typical ways to implement this step include the STRIDE method and using threat knowledge bases. STRIDE is a widely adopted and mature method for threat identification. The word "STRIDE" represents six categories of threats: spoofing identity, tampering with data, repudiation, information disclosure, denial of service, and elevation of privilege. STRIDE analysis is performed from the defender's perspective based on DFDs. Additionally, STRIDE provides checklists to help identify threats that fall in each category [2].

Prominent knowledge bases include the Common Vulnerabilities and Exposures (CVE)[2], Common Weakness Enumeration (CWE)[3], and Common Attack Pattern Enumeration and Classification (CAPEC)[4], which can further help identify threats. The CVE lists publicly disclosed cybersecurity vulnerabilities, while the CWE lists software and hardware weakness types. A weakness can lead to a specific instance of a vulnerability within a product or system. The CAPEC enumerates common attack patterns that help understand how weaknesses can be exploited by adversaries.

(iv) *Evaluate* the identified threats based on multiple aspects, such as accessibility to attackers, attack complexity, privileges required, etc. In order to quantify the impact of threats, scoring systems are employed. Moreover, since the threats are typically inter-connected, attack trees

[1]https://owasp.org/www-project-top-ten/

[2]https://cve.mitre.org/
[3]https://cwe.mitre.org/
[4]https://capec.mitre.org/

and attack graphs can be used to capture the relationships between the different threats.

To assist with evaluation, the Common Vulnerability Scoring System (CVSS)[5] and Common Weakness Scoring System (CWSS)[6] are designed to provide numerical scores of known vulnerabilities or weaknesses. CVSS and CWSS divide the evaluation metrics into multiple groups, and compute a score based on the metric groups. The score computed by CVSS or CWSS indicates the severity of a vulnerability or weakness, respectively.

Attack trees are conceptual diagrams that represent attacks in a tree form, where the root node represents the attack goal, and leaves represent ways to achieve the goal [3]. Attack graphs are directed graphs, where each vertex denotes a state (e.g., indicating that a certain vulnerability has been exploited) and each edge represents a state transition [4]. Attack trees and attack graphs are widely used for network security evaluation.

(v) *Mitigate* the potential threats according to the security requirements. The most suitable mitigations vary with the specific threat and the system where it resides. Nonetheless, some general suggestions for threat mitigation can be found in knowledge bases such as CWE and CAPEC.

(vi) *Validate* that the threats are mitigated after applying mitigation techniques. The OWASP Application Security Verification Standard (ASVS)[7] can help with this step by providing a basis for testing web application security controls.

Note that our definition of threat modeling steps is a variation of the five typical steps introduced by Microsoft on threat modeling [5]. Here, we introduced the additional "evaluate" step, to ensure that scarce security resources are allocated in a timely manner to the most important vulnerabilities.

The six steps defined in this section are to be iteratively applied during the SDLC. In some other definitions of threat modeling steps, the "define" step is omitted [6]. Since security requirements usually change over time, we argue that it is necessary to perform the "define" step each time to appropriately prioritize threats.

[5]https://www.first.org/cvss/

[6]https://cwe.mitre.org/cwss/cwss_v1.0.1.html

[7]https://owasp.org/www-project-application-security-verification-standard/

3. Overview of Threat Modeling Tools

Many threat modeling tools have been developed over the years. In this section, we give a taxonomy of current threat modeling tools. More detailed evaluation and comparison are presented in Section 5.

At the core of a threat modeling tool lies its library. The threat library lists a set of system artifacts to select from, threats associated to each artifact and mitigations associated to the threats. A system developer can design a system using the available artifacts either through a visual diagram drawing plane, or through text-based representation (e.g., in YAML format). The threat modeling tool then parses the system model and generates a threat report. Some tools further provide an integration of the weaknesses and their mitigations in a ticketing system. This allows tracking of the mitigations in the SDLC.

The tools can be categorized in several ways, for instance, non-commercial versus commercial, diagram-based versus text-based. Commercial tools are usually tailored for the needs of large companies (e.g., scalability and collaboration) but come at higher costs, while all the functionalities of non-commercial tools are available for free.

The non-commercial tools introduced in this section are entirely free to use without limitations on functionalities. All of them are open-source, except for the Microsoft Threat Modeling Tool.

Commercial tools emphasize usability, agile development, and collaboration. Almost all commercial tools are diagram-based. In the "model" step, commercial tools usually take measures to simplify the process of drawing the system diagram, for example, by providing questionnaires. Moreover, they typically maintain large threat libraries. Overall, commercial tools offer a rather complete solution for threat modeling, though their effectiveness remains to be compared with other tools.

3.1. Diagram-based tools

Threat modeling with diagram-based tools is intuitive: the tools provide users with an easy-to-use interface to draw system diagrams, and then apply rules from the library to detect potential threats. A report is then generated as a summary of threat modeling results.

Microsoft Threat Modeling Tool. The Microsoft Threat Modeling Tool (TMT) is a mature tool that has evolved over several years. For starters, it provides a default library (here called template). Once the model is built based on the template, the tool identifies threats by checking threat conditions in the template. Threats in the default template are categorized by STRIDE. The template can be customized to suit specific use cases. Elements in the diagram, threat types, and threat properties are all customizable. Threat properties include description, priority, and countermeasures. Users can also add community-contributed templates or build their own ones from scratch.

OWASP Threat Dragon. OWASP Threat Dragon (TD) shares a similar workflow to TMT. It is designed as a relatively light-weight tool. Hence, its threat library (here called threat rule engine) is not as customizable as TMT. TD only suggests generic potential threats, while details of the threats are left for users to fill in. Its advantages include workflow integration with Github, and support for more threat categories (LINDDUN and CIA) besides STRIDE.

OVVL. The Open Weakness and Vulnerability Modeler (OVVL) works as a web application. OVVL distinguishes itself from other diagram-based tools by identifying threats at both the design level and operation level [7]. At the design level, OVVL identifies STRIDE threats stored in its self-defined library (akin to TMT). At the operation level, OVVL can query the CVE knowledge base to find more specific vulnerabilities. This query is based on a description of system artifacts that follows a standard naming scheme.

IriusRisk (commercial). IriusRisk embeds the Draw.io diagram editor in its web application to enhance user experience when drawing the DFDs of a system. When initializing the diagram and adding new elements, questionnaires are provided to help users with configuration. Its threat library covers not only common knowledge bases like CVE and CWE, but also self-defined threats from typical enterprise use cases, such as AWS deployments. IriusRisk provides rather comprehensive threat evaluation, and offers priority and cost estimation of the suggested countermeasures.

ThreatModeler (commercial). The functionalities offered by ThreatModeler are similar to those of IriusRisk. However, in the "model" step, it applies so-called process flow diagrams (PFDs). Compared to the commonly used DFDs which model the system from a defender's perspective, PFDs aim to provide a visual decomposition of the system from an attacker's perspective.

SecuriCAD (commercial). SecuriCAD features AI-based predictive attack simulations. It includes a potential attacker in the diagram-based system model, and uses it to simulate possible ways ("attack paths") to compromise a system. By measuring the time-to-compromise, the critical attack paths to high-value assets can be found, and risk can be evaluated.

3.2. Text-based tools

To build a model with text-based tools, users are required to express their model in a structured language, instead of drawing the system diagram directly.

OWASP pytm. OWASP pytm describes the system model in Python language. It requires users to create a Python object for each of the system components based on its pre-defined Python classes. The tool then generates a system diagram and identifies threats from the text-based model. Its threat library contains comprehensive information about each potential threat. To mention a few, the "condition" attribute for threat identification, the "likelihood" and "severity" attributes for simple evaluation, and the "mitigation" attribute suggesting countermeasures.

Threagile. Threagile is designed for agile threat modeling. The models in Threagile are represented in YAML format. This format is more concise and abstract than the Python representation used in OWASP pytm. Threagile evaluates threats in a more rigorous way: instead of providing pre-defined evaluation information from the threat library, it calculates two metrics based on the system model: relative attacker attractiveness and data loss probability.

3.3. Other tools

A few other tools have been designed with slightly different purposes or expected outcomes. As a result, they do not strictly follow the six threat modeling steps defined in this paper.

CAIRIS. CAIRIS is a platform for specifying and modeling secure and usable systems. It takes

information about different aspects of the system, including assets, roles, and tasks. Users can then use CAIRIS to manually model the system as DFDs, or as a top-down representation similar to attack trees, and perform risk analysis and response.

Threatspec. Threatspec performs threat modeling directly on the code. It requires that developers and security engineers write threat modeling annotations as comments inside their source code. It then generates DFDs and reports from the commented source code.

SD Elements (commercial). SD Elements does not require users to draw a system diagram. Instead, it gathers information about the system via a survey. It claims to identify threats at a faster rate than traditional threat modeling processes.

Kenna (commercial). Kenna emphasizes data-driven threat modeling. It uses data science techniques, such as natural language processing and predictive modeling, to assess, prioritize, and predict risks. It introduces its own risk score engine for quantitative threat evaluation.

Tutamen Threat Model Automator (commercial). The Tutamen Threat Model Automater leverages existing project design diagrams instead of letting users create separate DFDs. Users enter metadata as values into each of the diagram elements to describe the threat model.

4. Tools Usage Example

In this section, we investigate three representative threat modeling tools through an example application. The selected tools include the community edition of a commercial tool (IriusRisk), a non-commercial diagram-based tool (TMT) and a non-commercial text-based tool (OWASP pytm). We model the example application with the selected tools, compare the results, and discuss our experience in using the tools. The main purpose of this section is to show how the tools are actually used for threat modeling, and how the selected tools differ.

The example application is one provided by OWASP pytm. The DFD of the application is shown in Fig. 1. The application is a simple web application, where a user logs into the application and posts comments on the app. The app server uses the real identity database to verify the user's identity, stores the comments into the SQL database, and shows the comments back to the user when requested. In the meantime, a separate AWS Lambda service periodically cleans the database.

We next model this application with TMT and the IriusRisk community edition. When modeling the application with TMT, we use the default template. Both TMT and IriusRisk use a diagram-based model. When drawing the diagram with the tools, users can select from pre-defined system components. Fig. 2 and Fig. 3 show the interface of drawing system diagrams using TMT and IriusRisk, respectively.

When building the model in OWASP pytm, a user needs to create a Python object for each system component, then set the component attributes (or leave them as default values). For example, the Python code used for defining the SQL database in Figure 1 is as follows:

```
db = Datastore("SQL Database")
db.OS = "CentOS"
db.isHardened = False
db.inBoundary = server_db
db.isSQL = True
db.inScope = True
db.maxClassification = Classification.
    RESTRICTED
```

All three tools generate reports of threat modeling results. We compare next these reports and notice the following differences (a more complete evaluation of the three tools is provided in Section 5, and here we only list additional findings from our usage example):

1) *Duplicate threats.* Each tool lists the same threat, if it occurs for different components. For example, OWASP pytm identifies the "privilege abuse" threat for both the web server and the real identity database. This is because the tools iterate through each component and check threat conditions in its library. In other words, each component is examined independently when identifying threats.
2) *Threat identification.* Data flow diagrams consist of processes, data stores, data flow, and external entities. TMT finds threats by checking the data flows, while IriusRisk mainly reasons about the other three types of components. In comparison, OWASP pytm examines all components in the diagram.

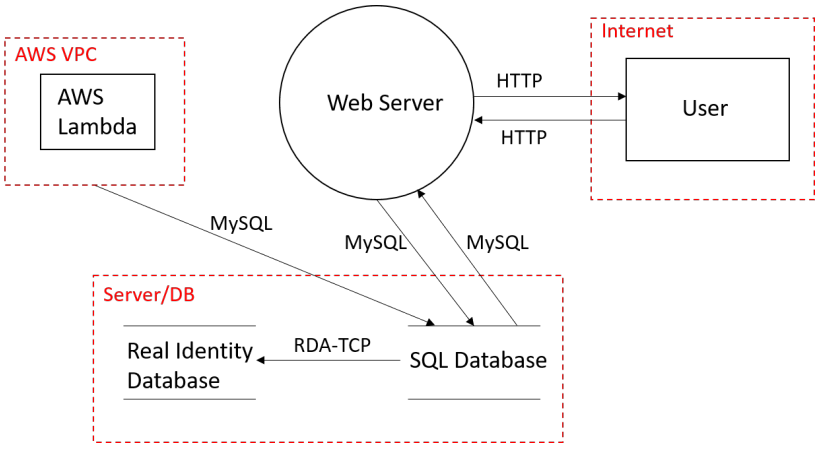

Figure 1. The example application modeled with threat modeling tools.

3) *Threat information provided to users.* With its default template, TMT provides the least information about the identified threats: no mitigations are suggested, and priority of threats are left for users to set. OWASP pytm offers basic threat evaluation and mitigation based on its threat library. IriusRisk provides further information (e.g., costs of countermeasures).
4) *Tracking threat status.* OWASP pytm lacks the functionality of tracking threats. TMT allows users to select the state of each threat from "not started", "needs investigation", "not applicable", and "mitigated". IriusRisk has a relatively more complete tracking system, and can also utilize external issue trackers such as Jira.

We find that IriusRisk provides a more complete support for the threat modeling process. In comparison, TMT and OWASP pytm fall short of certain capabilities. We also noted subtle differences in the ways used by the tools to identify threats. Regarding user experience, we find that the two diagram-based tools are more intuitive to use, and have more pre-defined types of system components. The questionnaires and abundant choice of pre-defined components provided by IriusRisk also make it easier to build the system model.

5. Evaluation of Threat Modeling Tools

In this section, we compare six threat modeling tools under a certain set of criteria. We mainly focus on freely available tools, since we could independently verify their capabilities. The selected tools are: Microsoft Threat Modeling Tool (TMT), OWASP Threat Dragon (TD), OVVL, OWASP pytm, Threagile, and IriusRisk. Among these tools, TMT, TD, and OVVL are non-commercial diagram-based tools, OWASP pytm and Threagile are non-commercial text-based tools, and IriusRisk is a commercial diagram-based tool. We evaluate the tools based on the criteria introduced in Section 1, and summarize the results in a table.

Note that TMT offers one default template (threat library), and two additional templates for threat modeling of Azure and medical devices, but we only evaluated the default one, since it is suitable for general use. Regarding IriusRisk, we choose the community edition for our comparison. This version can be used for free but has limited functionalities.

We stress that quantitative comparison of the performance of the threat modeling tools is difficult, because the tools require manual configuration and support customization (e.g., customizing rules for threat identification). Thus, users can greatly influence the results provided by each tool. As a result, we compare the tools qualitatively using the *default library and threat rules* provided by each tool. In other words, we only evaluate the capabilities offered by the tools, instead of their effectiveness. Still, the comparison is based on objective metrics.

We note that the recent work in [8] compares the TMT and TD tools. Our work differs in that

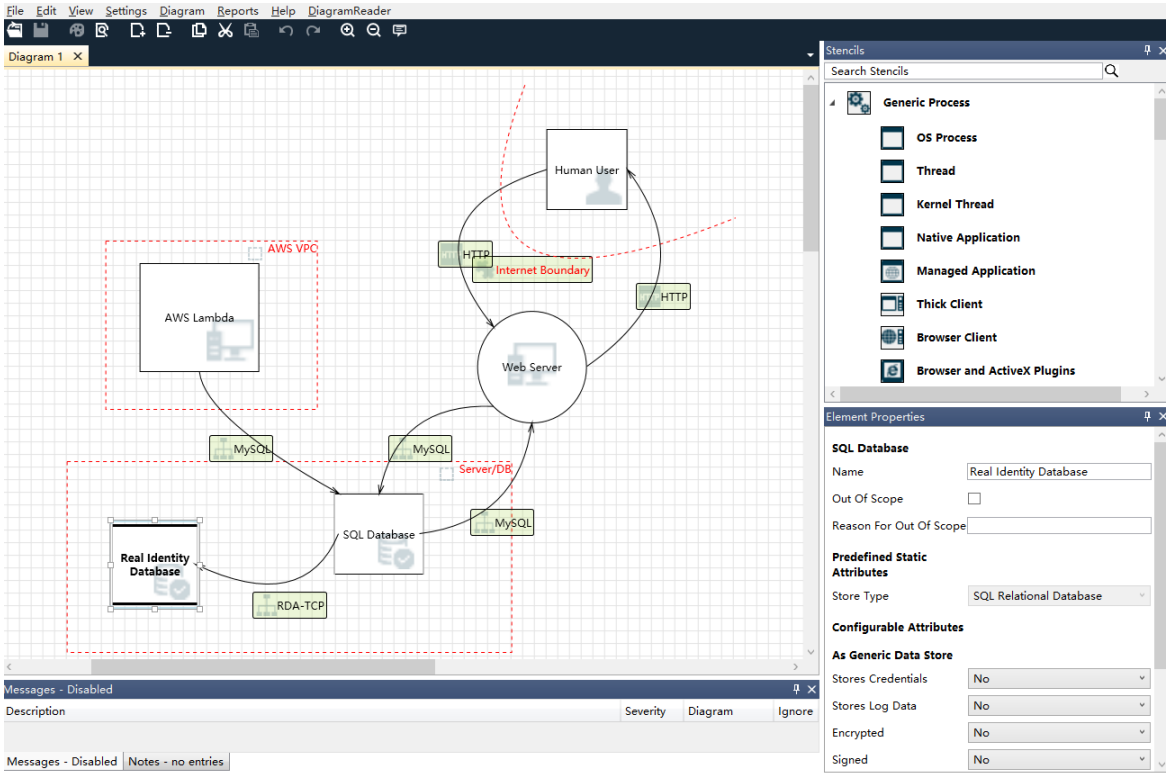

Figure 2. System diagrams drawn with the Microsoft Threat Modeling Tool.

we evaluate more tools, including commercial tools and text-based tools. Further, the evaluation in [8] is primarily based on user experience with the tools, while our evaluation is based on objective criteria regarding the tools' capabilities.

5.1. Evaluation criteria justification

In this subsection, we introduce our evaluation criteria, and explain their rationale.

There exists significant literature on evaluation criteria for software. Some work proposes criteria for general software [9], while others focus on specific types of software, such as commercial software [10], [11] and free/open-source software [12], [13]. Common concerns include functionality, usability, sustainability, and maintainability. In [14], common threat modeling methods are also evaluated and compared, which provides insights into requirements for threat modeling tools.

Note that some software evaluation criteria are subjective. For example, users might think web applications have better usability, or prefer command-line tools. Here, we only select criteria that can be evaluated objectively, such as whether a tool is available as a web application or through a command-line interface. Our evaluation can serve as a reference, while the final choice of a tool depends on the needs and preferences of users.

Our proposed criteria fall into two groups: (i) model form, model reusability, threat library and threat evaluation; (ii) availability, development stage and SDLC integration. The first group examines the functionality of the tools for threat modeling; the second evaluates other important aspects of the tools as a software. Evaluating functionality helps characterize the capability of a tool to assist with the threat modeling process. Other important aspects of a tool include its form (e.g., open-source or commercial), whether it is regularly maintained, and how well the tool fits into SDLC [9], [12], [13].

Model form. While a diagram-based model is often more intuitive to build and understand, a text-based model has several advantages, including: (i) performing version control of the model with common tools such as Git; (ii) enabling for-

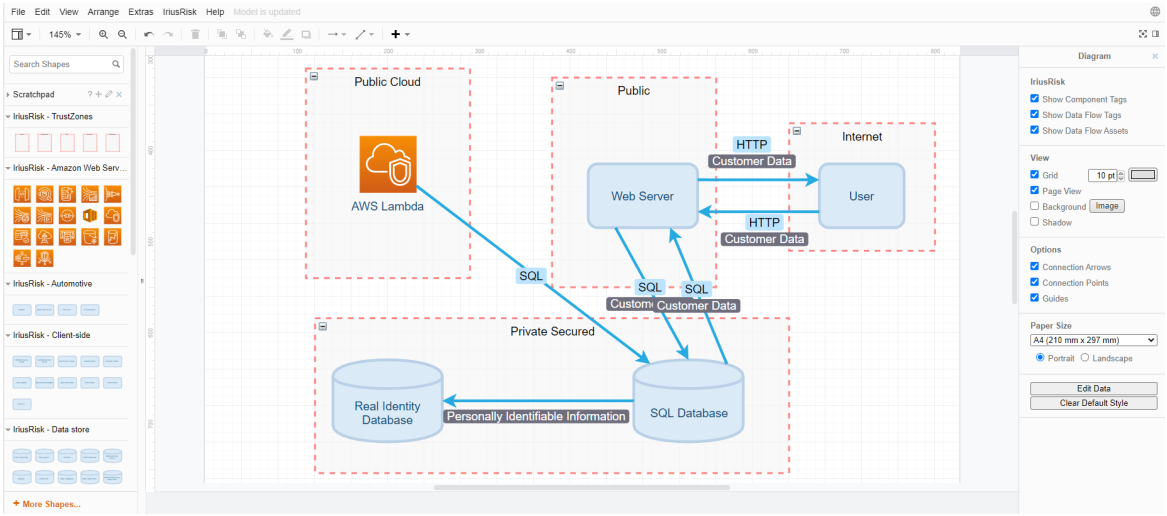

Figure 3. System diagrams drawn with the IriusRisk community edition.

mal and rigorous analysis; (iii) generating different forms of diagrams from the text-based model, for instance, DFDs and sequence diagrams.

Model reusability. Light-weight tools might work well for simple and small systems, but fall short of handling models of workflow at large companies. Some tools strive for model reusability, making it easier to scale the model. For example, allowing users to define a nested system component that can be reused, or allowing to import or export part of a diagram that can be used by other models.

Threat library. Threat modeling tools identify threats by following associations of system components to threats in their libraries. As a result, the threat library used by a tool determines the types and numbers of threats that can be found. In general, most of the tools allow extending the basic library, with commercial tools often providing a much larger and sophisticated threat library.

Threat evaluation. Some threats are more severe or urgent than others. By evaluating threats, a tool can help users make decisions on how to judiciously allocate security resources. The evaluation of a threat can be static or dynamic, where static evaluation shows pre-defined information in the threat library, and dynamic evaluation involves metric calculation based on the system model.

Mitigation suggestion. Suggestions for threat mitigation are valuable information when users decide to handle a threat. Common threat knowledge bases such as CWE usually include simple countermeasures to corresponding threats, making it easier for tools to propose mitigations. Some, but not all threat libraries of the considered tools link this mitigation information to threats in their libraries.

Availability. This criterion covers three sub-criteria: (i) how can users run the tool; (ii) on which platform the tool is available; (iii) whether the tool is open-source. Sub-criteria (i) and (ii) are related to the usability of the tool, while (iii) influences sustainability and maintainability [9]. Threat modeling tools can run as desktop or web applications, or through a command-line interface (CLI). Tools that run as desktop or web applications are usually more intuitive to use, while tools providing only command-line interface might be more efficient for experienced users and consume fewer resources. Some tools only work on specific platforms, which may cause problems to users on other platforms. Open-source tools are accessible to everyone for free, and tend to have a community that continually maintains the tools. Moreover, experienced users can extend these tools to suit their own needs. On the other hand, the adoption of commercial tools can be costly.

Development stage. It is important to consider whether a tool has long-term support and evolves over time [12]. Recent updates of a tool can indicate such information. Moreover, the maturity of the tool is reflected by its development stage. Some tools were released several years

ago, and have become relatively mature in terms of functionalities. In comparison, some newly released tools are still in development, with more feature updates planned in the future.

SDLC integration. Threat modeling tools are typically used in the SDLC in conjunction with other development software. During the SDLC, threat modeling needs to be performed repeatedly when new changes are made to the software. As a result, tools are required to be interoperable with other software [9]. SDLC integration is not only an important concern for threat modeling methods [14], but also for the tools themselves. Providing integration with tools in SDLC, such as issue tracking systems, can reduce manual work and make a tool easier to use.

5.2. Evaluation of the tools

In this subsection, we compare the selected tools based on our proposed criteria.

Model form. Most tools, including TMT, TD, OVVL and IriusRisk, model the system in a diagram form, which makes it easier for non-experts to build and understand models. However, a large diagram might be difficult to maintain. As an alternative approach, OWASP pytm and Threagile model the system in a text form following a specific syntax, which is more suitable for formal analysis and deliberate extensions using text-processing scripts, as both the system model and the threat library are available in structured text formats.

Model reusability. With TMT, TD and OVVL, system models are usually built from scratch, and the diagrams for each system are stored in individual files, which makes it difficult to reuse parts of the existing diagrams when building new models. OWASP pytm and Threagile, too, lack built-in support for reusing system modules. Nonetheless, since system components are all defined in text forms, it is easier to reuse modules of an existing model with these two text-based tools. For example, in OWASP pytm, users can import existing system modules from other files as Python objects. IriusRisk allows users to import and export parts of system diagrams as reusable modules and supports nested components, so that users can easily rearrange the diagram and define reusable modules.

Threat library. TD only suggests generic threats following the STRIDE/LINDDUN/CIA category, without indicating any specific threat. The default threat library of TMT includes about 50 pre-defined common threats categorized by STRIDE. We have not found the sources of those threats, so the pre-defined threats were presumably proposed by the developers of TMT. OVVL includes about 30 self-defined threats in its library according to the STRIDE category. It also supports importing the CVE list. Hence, more specific vulnerabilities can be identified based on the description of system components. Threagile covers about 40 threats from CWE, and OWASP pytm covers about 100 threats from CAPEC, CWE, and CVE. Threagile also indicates the STRIDE category of each threat. Among the selected tools, IriusRisk has the largest threat library (over 200), which contains not only threats in CVE, CWE, and CAPEC, but also many self-defined threats that are related to security standards and use cases, such as potential threats related to AWS.

Threat evaluation. With TMT and TD, threats are evaluated manually by letting users set the priority or severity of an identified threat. OVVL evaluates threats in two ways: self-defined STRIDE threats are evaluated by users manually, while threats from the CVE are presented with pre-defined CVSS scores. OWASP pytm contains pre-defined information about the likelihood of attack and threat severity in its library. Threagile not only has a pre-defined evaluation of each threat, but also calculates metrics based on the specific system model. IriusRisk evaluates identified threats and provides information including confidentiality, integrity, availability, and ease of exploitation. The threat evaluation is based on information stored in its threat library and on an optional set of priorities and criticality provided by the user.

Mitigation suggestion. TMT, TD and OVVL do not suggest possible countermeasures for mitigating the threats. OWASP pytm, Threagile, and IriusRisk provide users with simple countermeasures to the identified threats. IriusRisk also shows general guidelines to implement the mitigations and estimated costs of the countermeasures.

Availability. TMT is only available as a desktop application on Windows, while other tools are all cross-platform. TMT works locally as a

desktop application, IriusRisk and OVVL work as web applications, while TD can work in both ways. OWASP pytm and Threagile run through a command-line interface, while Threagile can also run as a server with REST API.

IriusRisk is the only commercial tool among the six tools considered here. Although the community edition of IriusRisk is available for free, its functionalities are limited and not suitable for real-world enterprise use. In comparison, TD, OVVL, OWASP pytm, Threagile are all open-source, whereas TMT is free to use but not open-source.

Development stage. TMT was initially released in 2014, and IriusRisk was initially released in 2015. Both TMT and IriusRisk have been continuously updated since then, and currently offer relatively stable functionalities. TD, OVVL, OWASP pytm, and Threagile are still in development. Their initial releases are relatively new, and more feature updates are planned in the future. We note that TD and OWASP pytm have been updated regularly.

SDLC integration. IriusRisk has integration with issue tracking systems like Jira, and testing frameworks like JUnit and Cucumber. TD supports opening models from a GitHub repository, and has scripts for integration with other tools such as Jira. OVVL offers integration with the Zoho BugTracker and interfaces to a few vulnerability scanners (e.g., Nessus). Threagile supports executing user-defined jobs (e.g., generating diagrams and reports) once the threat model file is changed on Github. It also provides editing support in various IDEs and YAML editors.

5.3. Summary

The evaluation results of the selected threat modeling tools in this section are summarized in Table 1. Note that the entire CVE knowledge base can be imported into OVVL as a threat library, which results in a very large library size. However, the resultant size is not directly comparable to that of other tools, since CVE entries are related to specific products or systems, while the pre-defined threats in other tools are more general. Note also that the default threat library of TMT does not suggest mitigations, but users are able to customize this library to include such kind of information.

From the comparison results, we note that commercial tools typically have more complete functionalities, though their commercial editions with all the features unlocked come at a higher cost. While non-commercial tools can be used for simple threat modeling, they might fall short of certain aspects when used by large companies or organizations, e.g., model reusability, threat library size, or SDLC integration.

TD, OVVL, OWASP pytm, and Threagile are all open-source and relatively new. Compared with closed-source tools, they are easier to extend. However, they tend to be in a relatively premature state, with certain functionalities planned but not yet implemented.

Most threat modeling tools leverage common threat knowledge bases, such as CWE. The knowledge bases used by OWASP pytm, Threagile, and IriusRisk not only enlarge the threat libraries, but also provide basic information about threat evaluation and mitigation, which can be valuable to users.

Current tools typically adopt simple solutions for threat evaluation and mitigation. Some tools let users perform these steps manually, and some provide pre-defined information from their threat libraries. However, threats in a real-world system are typically inter-related. Hence, the severity and expected loss of a threat depend on the specificity of a system. Research on threat modeling has proposed rigorous approaches for threat evaluation and mitigation prioritization, e.g., based on attack graphs [4], [15], but current tools have not adopted these approaches yet.

Our evaluation suggests that some tools lack certain capabilities. For example, TMT does not offer SDLC integration. However, missing a few capabilities does not necessarily mean a tool is useless. The most suitable threat modeling tool depends on users' needs, preferences, and budget. For users with a large budget, IriusRisk seems a good option. For a user that prefers a relatively mature and intuitive tool, TMT is appropriate. If integration with other tools in SDLC is important, TD and OVVL might be worth considering. Last, tools that adopt a text-based system model provide unique advantages in terms of reusability and automation (scripting).

Criterion		TMT	OWASP TD	OVVL	OWASP pytm	Threagile	IriusRisk community edition
Availability	Usage	desktop app	desktop / web app	web app	CLI	CLI / server	web app
	Cross-platform	✗	✓	✓	✓	✓	✓
	Open-source	✗	✓	✓	✓	✓	✗
Model form		diagram	diagram	diagram	text-based (Python)	text-based (YAML)	diagram
Model reusability	Nested components	✗	✗	✗	✗	✗	✓
	Module import / export	✗	✗	✗	✗	✗	✓
Threat library	Source	self-defined (STRIDE)	self-defined (STRIDE, LINDDUN, CIA)	self-defined (STRIDE), CVE	CAPEC, CVE, CWE	CWE	CAPEC, CVE, CWE, self-defined
	Size	~ 50	no specific threats	$\sim$ 30 self-defined, and CVE list	~ 100	~ 40	> 200
Threat evaluation		manual or pre-defined	manual	manual or pre-defined	pre-defined	pre-defined, calculation-based	provided, but the mechanism unclear
Mitigation suggestion		library-dependent[1]	✗	✗	✓	✓	✓
Development stage	Initial release	2014 (TMT 2014)	Feb 2020 (v1.0)	Jul 2019 (v1.0)	Dec 2019 (v1.0)	Aug 2020 (v1.0)	2015 (v1.0)
	Recent release	Jul 2020 (v7.3)	Aug 2021 (v1.5)	Jul 2019 (v1.0)	Apr 2021 (v1.2)	Aug 2020 (v1.0)	Sep 2021 (v3.14)
SDLC integration		✗	Github, issue trackers	issue trackers, vulnerability scanners	✗	Github, IDE editing support	issue trackers, testing frameworks

Table 1. Comparison of six threat modeling tools: Microsoft Threat Modeling Tool (TMT), OWASP Threat Dragon (TD), OVVL, OWASP pytm, Threagile, and IriusRisk community edition.

[1] Mitigation suggestion in TMT depends on its threat library, which can be customized by users.

6. Conclusion

This work provided a taxonomy of threat modeling tools. We categorized common threat modeling tools, and highlighted their differences, including through an example case. In order to evaluate threat modeling tools in a standardized manner, we introduced several criteria relevant to the threat modeling process. We further compared six representative tools following these criteria, and summarized the results in a table. We expect that this taxonomy will be useful to researchers and practitioners when selecting a threat modeling tool.

An issue identified by our taxonomy is the relative immaturity of open-source threat modeling tools. Extending functionalities of these tools would be highly valuable for the community, as only open-source tools can properly be scrutinized and vetted. We also noted that current tools adopt relatively simple approaches for evaluating threats and prioritizing mitigations, which leaves room for improvement. Another important area left for future work is the development of standardized benchmarks, which could assist in performing quantitative evaluation of different threat modeling tools.

Acknowledgement

This work was supported in part by a Honda Research Institute and BU Hariri Institute Research Incubation Award (2020-06-006).

REFERENCES

1. R. Stevens, D. Votipka, E. M. Redmiles, C. Ahern, P. Sweeney, and M. L. Mazurek, "The battle for new york: a case study of applied digital threat modeling at the enterprise level," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 621–637.
2. R. Scandariato, K. Wuyts, and W. Joosen, "A descriptive study of Microsoft's threat modeling technique," *Re-*

quirements Engineering, vol. 20, no. 2, pp. 163–180, 2015.

3. V. Saini, Q. Duan, and V. Paruchuri, "Threat modeling using attack trees," *Journal of Computing Sciences in Colleges*, vol. 23, no. 4, pp. 124–131, 2008.
4. K. Ingols, M. Chu, R. Lippmann, S. Webster, and S. Boyer, "Modeling modern network attacks and countermeasures using attack graphs," in *2009 Annual Computer Security Applications Conference*. IEEE, 2009, pp. 117–126.
5. "Threat modeling," 2021. [Online]. Available: https://www.microsoft.com/en-us/securityengineering/sdl/threatmodeling
6. A. Shostack, *Threat modeling: Designing for security*. John Wiley & Sons, 2014.
7. A. Schaad, "Project OVVL–threat modeling support for the entire secure development lifecycle," *SICHERHEIT 2020*, 2020.
8. E. Bygdås, L. A. Jaatun, S. B. Antonsen, A. Ringen, and E. Eiring, "Evaluating threat modeling tools: Microsoft TMT versus OWASP Threat Dragon," in *2021 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*. IEEE, 2021, pp. 1–7.
9. M. Jackson, S. Crouch, and R. Baxter, "Software evaluation: criteria-based assessment," *Software Sustainability Institute*, vol. 1, 2011.
10. M. Morisio, C. B. Seaman, V. R. Basili, A. T. Parra, S. E. Kraft, and S. E. Condon, "COTS-based software development: Processes and open issues," *Journal of Systems and Software*, vol. 61, no. 3, pp. 189–199, 2002.
11. D. J. Carney and K. C. Wallnau, "A basis for evaluation of commercial software," *Information and Software Technology*, vol. 40, no. 14, pp. 851–860, 1998.
12. D. Cruz, T. Wieland, and A. Ziegler, "Evaluation criteria for free/open source software products based on project analysis," *Software Process: Improvement and Practice*, vol. 11, no. 2, pp. 107–122, 2006.
13. L. P. Money, S. Praseetha, and D. Mohankumar, "Open source software–quality benefits, evaluation criteria and adoption methodologies," *Journal of Computations & Modelling*, vol. 2, no. 3, pp. 1–16, 2012.
14. N. Shevchenko, B. R. Frye, and C. Woody, "Threat modeling for cyber-physical system-of-systems: Methods evaluation," Carnegie Mellon University Software Engineering Institute Pittsburgh United States, Tech. Rep., 2018.
15. J. Homer, S. Zhang, X. Ou, D. Schmidt, Y. Du, S. R. Rajagopalan, and A. Singhal, "Aggregating vulnerability metrics in enterprise networks using attack graphs," *Journal of Computer Security*, vol. 21, no. 4, pp. 561–597, 2013.

Appendix

Zhenpeng Shi is a Ph.D. candidate in Electrical and Computer Engineering at Boston University. His research interests include network systems, cybersecurity, and game-theoretic modeling. Contact him at zpshi@bu.edu.

Kalman Graffi is Principal Scientist at the Honda Research Institute Europe. His research interests include security and privacy in distributed systems. Contact him at kalman.graffi@honda-ri.de.

David Starobinski is a Professor of Electrical and Computer Engineering, Systems Engineering, and Computer Science at Boston University. His research interests include cybersecurity, wireless networking, and network economics. He is an IEEE Senior Member. Contact him at staro@bu.edu.

Nikolay Matyunin is a Scientist at the Honda Research Institute Europe. His research interests include different aspects of information security, data privacy, and privacy-enhancing technologies. Contact him at nikolay.matyunin@honda-ri.de.