Threat Modeling Tools

A Taxonomy

Zhenpeng Shi | Boston University
Kalman Graffi | Honda Research Institute Europe
David Starobinski | Boston University
Nikolay Matyunin | Honda Research Institute Europe

Threat modeling tools allow for identifying weaknesses in a system's design. Yet, understanding conceptual differences between the tools is not trivial. We propose a taxonomy of threat modeling tools, using it to compare several popular ones.

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 of Stevens et al.[1] presents a case study of threat modeling conducted at New York City Cyber Command, a large-scale, 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 article, 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 as 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 the "Evaluation of Threat Modeling Tools" section:

1. *Model form*: What form of system model is adopted by the tool (e.g., diagram versus text based)?
2. *Model reusability*: Is the tool able to reuse parts of existing models when building new ones?
3. *Threat library*: Which 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 a command-line interface (CLI)? Is the tool open or closed source?

Digital Object Identifier 10.1109/MSEC.2021.3125229
Date of current version: 2 December 2021

7. *Development stage*: Is the tool still in development or in a relatively mature state?
8. *Software development life cycle (SDLC) integration*: Does the tool offer integration with other development tools regularly used in the SDLC (e.g., issue tracking systems)?

We envision that our proposed taxonomy of threat modeling tools will help academic researchers and security practitioners 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 the qualitative evaluation of threat modeling tools. The criteria cover different features and capabilities of the tools.
2. We use the criteria proposed in this article 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., the Microsoft Threat Modeling Tool (TMT), Open Web Application Security Project (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.

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:

1. *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 resource that helps with this step includes the OWASP Top Ten (https://owasp.org/www-project-top-ten/), which represents the most critical 10 types of security risks to web applications. Such resources provide developers with a quick and effective way to reason about fundamental security requirements that should be satisfied.
2. *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.
3. *Identify potential threats based on the system model*: The 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 into each category.[2] The prominent knowledge bases include the Common Vulnerabilities and Exposures (CVE) (https://cve.mitre.org/), Common Weakness Enumeration (CWE) (https://cwe.mitre.org/), and Common Attack Pattern Enumeration and Classification (CAPEC) (https://capec.mitre.org/), 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.
4. *Evaluate the identified threats based on multiple aspects, such as accessibility to attackers, attack complexity, privileges required, and so on*: To quantify the impact of threats, scoring systems are employed. Moreover, as threats are typically inter-connected, attack trees and attack graphs can be used to capture the relationships among the different threats. To assist with evaluation, the Common Vulnerability Scoring System (CVSS) (https://www.first.org/cvss/) and Common Weakness Scoring System (CWSS) (https://cwe.mitre.org/cwss/cwss_v1.0.1.html) are designed to provide numerical scores of known vulnerabilities or weaknesses. The CVSS and CWSS divide the evaluation metrics into multiple groups and compute a score based on the metric groups. The score computed by the 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.

5. *Mitigate 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 the CWE and CAPEC.
6. *Validate that the threats are mitigated after applying mitigation techniques*: The OWASP Application Security Verification Standard (https://owasp.org/www-project-application-security-verification-standard/) 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 introduce 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] As security requirements usually change over time, we argue that it is necessary to perform the "define" step each time to appropriately prioritize threats.

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. A more detailed evaluation and comparison are presented in the "Evaluation of Threat Modeling Tools" section.

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 with 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 Ain't Markup Language (YAML) format]. The threat modeling tool then parses the system model and generates a threat report. Some tools further provide an integration of the threats 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, noncommercial versus commercial, diagram based versus text based. The commercial tools are usually tailored to the needs of large companies (e.g., scalability and collaboration) but come at higher costs, while all the functionalities of noncommercial tools are available for free.

The noncommercial tools introduced in this section are entirely free to use, without limitations on functionalities. All of them are open source, except for the Microsoft TMT.

The commercial tools emphasize usability, agile development, and collaboration. Almost all of the 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, although their effectiveness remains to be compared with other tools.

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 TMT. The Microsoft TMT is a mature tool that has evolved over several years. For starters, it provides a default library (here called the *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. The elements in the diagram, threat types, and threat properties are all customizable. The threat properties include description, priority, and countermeasures. Users can also add community-contributed templates or build their own ones from scratch.

OWASP Threat Dragon. The OWASP Threat Dragon (TD) shares a workflow similar to the TMT's. It is designed as a relatively lightweight tool. Hence, its threat library (here called the *threat rule engine*) is not as customizable as the TMT. TD suggests only 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 [Linkability, Identifiability, Nonrepudiation, Detectability, Disclosure of information, Unawareness, Non-compliance (LINDDUN) and confidentiality, integrity, and availability (CIA] besides STRIDE.

OVVL. The Open Weakness and Vulnerability Modeler (OVVL) works as a web application. The OVVL distinguishes itself from other diagram-based tools by identifying threats at both the design and operation levels.[7] At the design level, the OVVL identifies STRIDE threats stored in its self-defined library (akin to the TMT). At the operation level, the 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 the configuration. Its threat library covers not only common knowledge bases like the CVE and CWE but also self-defined threats from typical enterprise use cases, such as Amazon Web Services (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 that model the system from a defender's perspective, PFDs aim to provide a visual decomposition of the system from an attacker's point of view.

SecuriCAD (Commercial). SecuriCAD features artificial intelligence-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.

Text-Based Tools

To build a model with text-based tools, users are required to express their model in a structured language. In fact, we are not claiming that a text-based approach is always better (it depends on the use cases). Our goal is to highlight the difference between the two approaches, that is, text-based approach does not require drawing the 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 predefined 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, including 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 predefined evaluation information from the threat library, it calculates two metrics based on the system model: relative attacker attractiveness and data loss probability.

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 article.

CAIRIS. Computer-Aided Integration of Requirements and Information Security (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 similar top-down representation to attack trees, and perform risk analysis and response.

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

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 noncommercial diagram-based tool (the TMT), and a noncommercial text-based tool (OWASP pytm). We model the example application with the selected tools, compare the results, and discuss our experience with 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 Figure 1. The application is a simple web application where a user logs in to 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 Structured Query Language (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 the TMT and IriusRisk community edition. When modeling the application with the TMT, we use the default template. Both the TMT and IriusRisk use a diagram-based model. When drawing the diagram with the tools, users can select from predefined system components. Figures 2 and 3 show the interface of drawing system diagrams using the 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 following Python code is used for defining the SQL database in Figure 1:

```
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 next compare these reports and notice the following differences (a more complete evaluation of the three tools is provided in the "Evaluation of Threat Modeling Tools" section; here we list only the 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*: DFDs consist of processes, data stores, data flow, and external entities. The

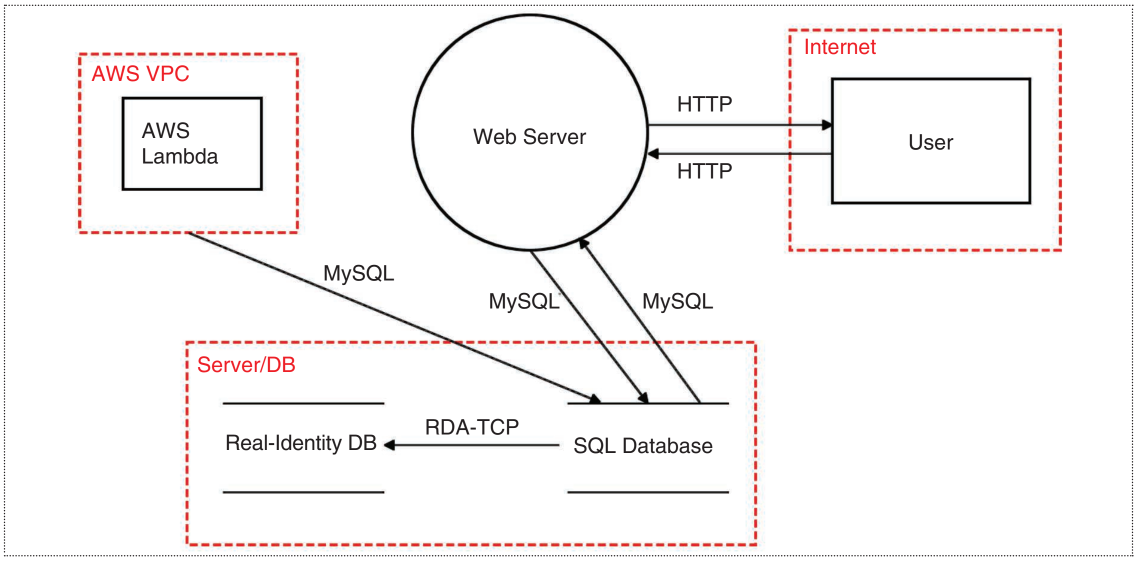


Figure 1. The example application modeled with threat modeling tools. SQL: Structured Query Language; DB: database; VPC: Virtual Private Cloud; RDA: Remote Database Access; TCP: Transmission Control Protocol.

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.

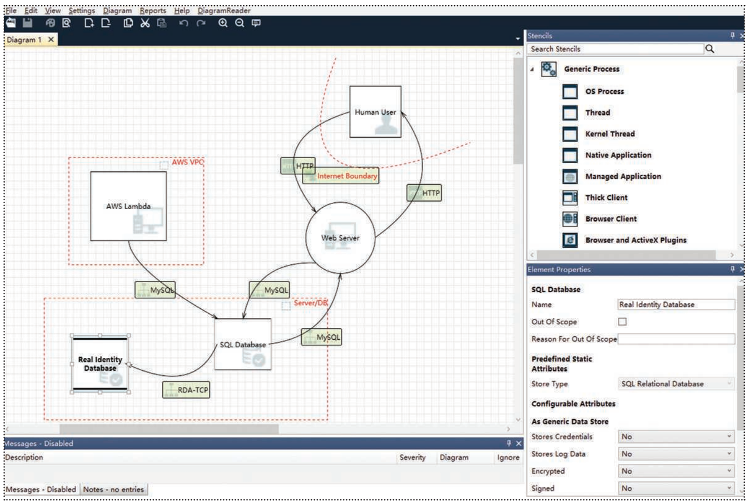
3. *Threat information provided to users*: With its default template, the TMT provides the least amount of information about the identified threats: no mitigations are suggested, and the priority of threats is left for users to set. OWASP pytm offers basic threat evaluation and mitigation based on its threat library. IriusRisk provides further information (e.g., the cost of countermeasures).
4. *Tracking threat status*: OWASP pytm lacks the functionality of tracking threats. The TMT allows users to select the status 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, the TMT and OWASP pytm fall short of certain capabilities. We also noted subtle differences in the methods 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 predefined types of system components. The questionnaires and abundant choice of predefined components provided by IriusRisk also make it easier to build the system model.

Evaluation of Threat Modeling Tools

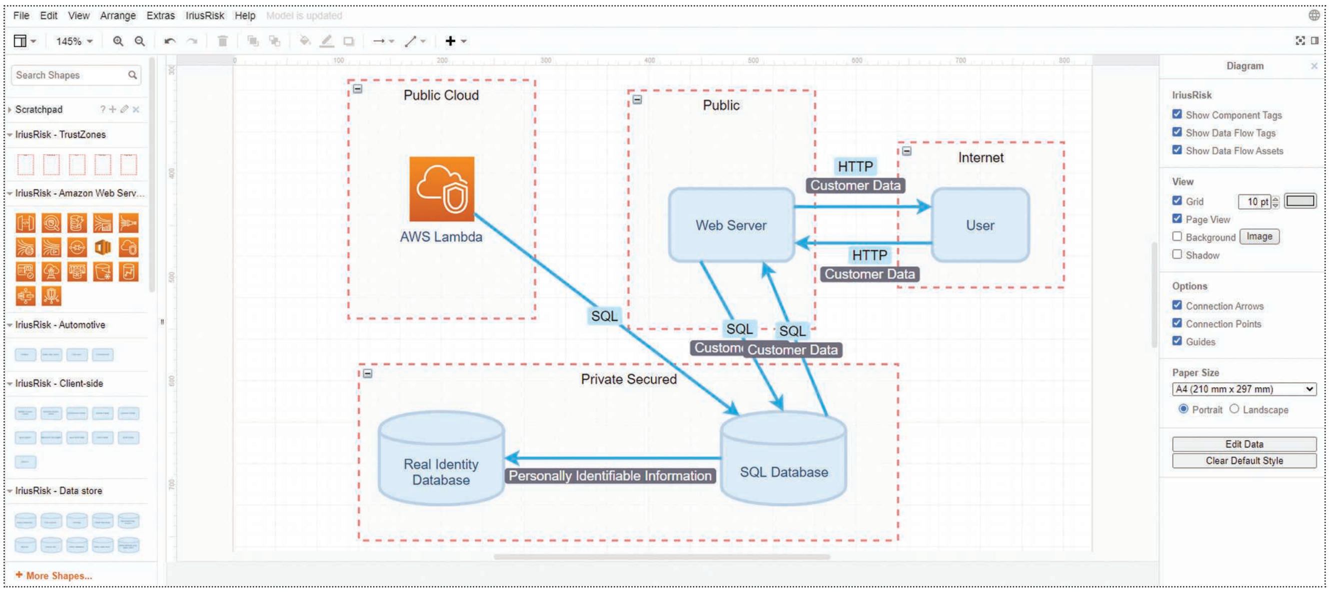
In this section, we compare six threat modeling tools under a certain set of criteria. We mainly focus on freely available tools because we could independently verify their capabilities. The selected tools are the TMT, OWASP TD, OVVL, OWASP pytm, Threagile, and IriusRisk. Among these tools, the TMT, TD, and OVVL are noncommercial diagram-based tools; OWASP pytm and Threagile are noncommercial text-based tools; and IriusRisk is a commercial diagram-based tool. We evaluate the tools based on the criteria introduced earlier and summarize the results in a table.

Note that the TMT offers one default template (threat library) and two additional templates for threat modeling of Azure and medical devices, but we only



Element Properties
SQL Database
Name: Real Identity Database
Out Of Scope: ☐
Reason For Out Of Scope:
Predefined Static Attributes
Store Type: SQL Relational Database
Configurable Attributes
As Generic Data Store
Stores Credentials: No
Stores Log Data: No
Encrypted: No
Signed: No

Figure 2. The system diagrams drawn using the Microsoft TMT.



IriusRisk
☑ Show Component Tags
☑ Show Data Flow Tags
☑ Show Data Flow Assets
View
☑ Grid 10 pt
☑ Page View
☐ Background Image
☐ Shadow
Options
☑ Connection Arrows
☑ Connection Points
☑ Guides
Paper Size
A4 (210 mm x 297 mm)
● Portrait ○ Landscape
Edit Data
Clear Default Style

Figure 3. The system diagrams drawn using the IriusRisk community edition.

evaluated the default one as 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 of Bygdás et al.[8] compares the TMT and TD tools. Our work differs in that we evaluate more tools, including commercial and text-based tools. Further, the evaluation of Bygdás et al.[8] is primarily based on user experience with the tools, while our evaluation is based on objective criteria regarding the tools' capabilities.

Evaluation Criteria Justification

In this section, 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 kinds, such as commercial[10,11] and free/open source types.[12,13] The common concerns include functionality, usability, sustainability, and maintainability. Shevchenko et al. evaluate and compare common threat modeling methods,[14] which provide insights into the 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 select only those criteria that can be evaluated objectively, such as whether a tool is available as a web application or through a CLI. 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: 1) model form, model reusability, threat library, and threat evaluation and 2) availability, development stage, and SDLC integration. The first group examines the functionality of the tools for threat modeling, and 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. The 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 the SDLC.[9,12,13]

Model Form. Although a diagram-based model is often more intuitive to build and understand, a text-based one has several advantages, including 1) performing version control of the model with common tools such as Git, 2) enabling a formal and rigorous analysis, and 3) generating different forms of diagrams from the text-based one, for instance, DFDs and sequence diagrams.

Model Reusability. Lightweight tools might work well for simple and small systems, but they 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 the importation or exportation of a 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 for 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 a static evaluation shows predefined information in the threat library and a dynamic evaluation involves a metric calculation based on the system model.

Mitigation Suggestion. The suggestions for threat mitigation are valuable information when users decide to handle a threat. Common threat knowledge bases such as the 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 subcriteria: 1) how can users run the tool, 2) on which platform the tool is available, and 3) whether the tool is open source. Subcriteria 1) and 2) are related to the usability of the tool, while 3) influences sustainability and maintainability.[9] Threat modeling tools can run as desktop or web applications, or through a CLI. Tools that run as desktop or web applications are usually more intuitive to use, while tools providing only a CLI might be more efficient for experienced users and consume fewer resources. Some tools work only on specific platforms, which may cause problems for 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] The 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 their functionalities. In comparison, some newly released tools are still in development, with more feature updates planned for 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 the SDLC, such as issue tracking systems, can reduce manual work and make a tool easier to use.

Tool Evaluations

Model Form. In this section, we compare the selected tools based on our proposed criteria. Many tools, including the TMT, TD, OVVL, and IriusRisk, model the system in a diagram form, which makes it easier for nonexperts 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 the 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 also lack built-in support for reusing system modules. Nonetheless, as system components are all defined in text forms, it is easier to reuse the 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 suggests only generic threats following the STRIDE/LINDDUN/CIA category, without indicating any specific threat. The default threat library of the TMT includes approximately 50 predefined common threats categorized by STRIDE. We have not found the sources of those threats so the predefined threats were presumably proposed by the developers of the TMT. The OVVL includes roughly 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 approximately 40 threats from the CWE, and OWASP pytm covers roughly 100 threats from the CAPEC, CWE, and CVE. Threagile also indicates the STRIDE category of each threat. Among the selected tools, IriusRisk has the largest threat library (more than 200), which contains not only threats in the CVE, CWE, and CAPEC, but also many self-defined threats, which are related to security standards and use cases, such as the potential threats related to AWS.

Threat Evaluation. With the TMT and TD, threats are evaluated manually by letting users set the priority or severity of an identified threat. The OVVL evaluates threats in two ways: self-defined STRIDE threats are evaluated by users manually, while threats from the CVE are presented with predefined CVSS scores. OWASP pytm contains predefined information about the likelihood of an attack and threat severity in its library. Threagile not only has a predefined 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. The 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 presents general guidelines to implement the mitigations and estimated costs of the countermeasures.

Availability. The TMT is available only as a desktop application on Windows, while other tools are all cross platform. The 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 CLI, while Threagile can also run as a server with a Representational State Transfer application programming interface (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, and Threagile are all open source, whereas the TMT is free to use but not open source.

Development Stage. The TMT was initially released in 2014, and IriusRisk was initially released in 2015. Both the 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 for 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. The 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.

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 the 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 as CVE entries are related to specific products or systems, while the predefined threats in other tools are more general. Note also that the default threat library of the TMT does not suggest mitigations, but users are able to customize this library to include such kinds of information.

From the comparison results, we note that commercial tools typically have more complete functionalities, although their commercial editions with all the features unlocked come at a higher cost. Even though noncommercial 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 the CWE. The knowledge bases used by OWASP pytm, Threagile, and IriusRisk not only enlarge threat libraries but also provide basic information about threat evaluation and mitigation, which can be valuable to users.

The current tools typically adopt simple solutions for threat evaluation and mitigation. Some tools let users perform these steps manually, and some provide predefined information from their threat libraries. However, threats in a real-world system are typically interrelated. Hence, the severity and expected loss of a threat depend on the specificity of a system. The 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 yet adopted these approaches.

Our evaluation suggests that some tools lack certain capabilities; for example, the 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, the TMT is appropriate. If integration with other tools in the SDLC is important, TD and the OVVL might be worth considering. Finally, tools that adopt a text-based system model provide unique advantages in terms of reusability and automation (scripting).

This work provided a taxonomy of threat modeling tools. We categorized common threat modeling tools and highlighted their differences, including through an example case. 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.

Table 1. A comparison of the six threat modeling tools: the Microsoft TMT, OWASP TD, OVVL, OWASP pytm, Threagile, and IriusRisk community edition.

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, and CIA)	Self-defined (STRIDE) and CVE	CAPEC, CVE, and CWE	CWE	CAPEC, CVE, CWE, and self-defined
	Size	~50	No specific threats	~30 self-defined and CVE list	~100	~40	> 200
Threat evaluation		Manual or predefined	manual	Manual or predefined	Predefined	Predefined and calculation based	Provided, but the mechanism is unclear
Mitigation suggestion		Library dependent*	✗	✗	✓	✓	✓
Development stage —	Initial release	2014 (TMT 2014)	February 2020 (v1.0)	July 2019 (v1.0)	December 2019 (v1.0)	August 2020 (v1.0)	2015 (v1.0)
	Recent release	July 2020 (v7.3)	August 2021 (v1.5)	July 2019 (v1.0)	April 2021 (v1.2)	August 2020 (v1.0)	September 2021 (v3.14)
SDLC integration		✗	GitHub and issue trackers	Issue trackers and vulnerability scanners	✗	GitHub and integrated development environment editing support	Issue trackers and testing frameworks

The mitigation suggestion in the TMT depends on its threat library, which can be customized by users.

An issue identified by our taxonomy is the relative immaturity of open source threat modeling tools. Extending the 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 a quantitative evaluation of different threat modeling tools. ■

Acknowledgments

This work was supported in part by a Honda Research Institute and BU Hariri Institute Research Incubation Award (2020-06-006). Zhenpeng Shi is the corresponding author.

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 *Proc. 27th USENIX Security Symp. (USENIX Security 18)*, 2018, pp. 621–637.
2. R. Scandariato, K. Wuyts, and W. Joosen, "A descriptive study of Microsoft's threat modeling technique," *Requirements Eng.*, vol. 20, no. 2, pp. 163–180, 2015, doi: 10.1007/s00766-013-0195-2.
3. V. Saini, Q. Duan, and V. Paruchuri, "Threat modeling using attack trees," *J. Comput. Sci. 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 *Proc. Annu. Comput. Security Appl. Conf.*, 2009, pp. 117–126, doi: 10.1109/ACSAC.2009.21.
5. "Threat modeling," Microsoft, 2021. [Online]. Available: https://www.microsoft.com/en-us/securityengineering/sdl/threatmodeling
6. A. Shostack, *Threat Modeling: Designing for Security*. Hoboken, NJ, USA: Wiley, 2014.
7. A. Schaad, "Project OVVL–threat modeling support for the entire secure development lifecycle," in *Proc. SICHERHEIT 2020*, 2020, pp. 121–124.
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 *Proc. Int. Conf. Cyber Situational Awareness, Data Analytics Assessment (CyberSA)*, 2021, pp. 1–7. doi: 10.1109/CyberSA52016.2021.9478215.
9. M. Jackson, S. Crouch, and R. Baxter, "Software evaluation: Criteria-based assessment," *Softw. Sustain. Inst.*, vol. 1, 2011. [Online]. Available: https://software.ac.uk/sites/default/files/SSI-SoftwareEvaluationCriteria.pdf.
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," *J. Syst. Softw.*, vol. 61, no. 3, pp. 189–199, 2002, doi: 10.1016/S0164-1212(01)00147-9.
11. D. J. Carney and K. C. Wallnau, "A basis for evaluation of commercial software," *Inf. Softw. Technol.*, vol. 40, no. 14, pp. 851–860, 1998, doi: 10.1016/S0950-5849(98)00099-8.
12. D. Cruz, T. Wieland, and A. Ziegler, "Evaluation criteria for free/open source software products based on project analysis," *Softw. Process, Improvement Pract.*, vol. 11, no. 2, pp. 107–122, 2006, doi: 10.1002/spip.257.
13. L. P. Money, S. Praseetha, and D. Mohankumar, "Open source software–quality benefits, evaluation criteria and adoption methodologies," *J. Comput. 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," Software Engineering Inst., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep., 2018. [Online]. Available: https://apps.dtic.mil/sti/pdfs/AD1084209.pdf
15. J. Homer et al., "Aggregating vulnerability metrics in enterprise networks using attack graphs," *J. Comput. Security*, vol. 21, no. 4, pp. 561–597, 2013. doi: 10.3233/JCS-130475.

Zhenpeng Shi is a Ph.D. candidate in electrical and computer engineering at Boston University, Boston, Massachusetts, 02115, USA. His research interests include network systems, cybersecurity, and game-theoretic modeling. Shi received his B.E. in automation from Harbin Institute of Technology, China. Contact him at zpshi@bu.edu.

Kalman Graffi is a principal scientist at the Honda Research Institute Europe, Offenbach am Main, 63073, Germany. His research interests include different aspects of privacy-preserving technologies, secure software engineering and secure distributed systems. Graffi received his Dr.-Ing. in 2010 from the Institute of Electrical Engineering and Information Technology at the Technical University, Darmstadt, Germany. 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, Boston, Massachusetts, 02115, USA. His research interests include cybersecurity, wireless networking, and network economics. Starobinski received his Ph.D. in electrical engineering from the Technion—Israel Institute of Technology. He is a Senior Member of IEEE. Contact him at staro@bu.edu.

Nikolay Matyunin is a scientist at the Honda Research Institute Europe, Offenbach am Main, 63073, Germany, and a Ph.D. candidate in the Computer Science Department at the Technical University of Darmstadt. His research interests include different aspects of information security, data privacy, and privacy-enhancing technologies. Matyunin received his diploma in computer science from Lomonosov Moscow State University, Moscow, Russia. Contact him at nikolay.matyunin@honda-ri.de.