SAFECode
Software Assurance Forum for Excellence in Code
Driving Security and Integrity

Tactical Threat Modeling

Table of Contents

1. Foreword

The Software Assurance Forum for Excellence in Code (SAFECode) is a non-profit organization exclusively dedicated to increasing trust in information and communications technology products and services through the advancement of effective software assurance methods. SAFECode is a global, industry-led effort to identify and promote best practices for developing and delivering more secure and reliable software, hardware and services.

This document, in addition to the online training provided by SAFECode (https://training.safecode.org/), will provide guidance about the process of threat modeling as well as the "**generic"** framework in which a successful threat-modeling effort can be conducted. We will suggest basic approaches and more extensive sources for developing your own workflow. Moreover, we will address issues less explored in the literature, such as team composition, scaling the effort, threat modeling in Agile environments, and others.

2. Why Do Threat Modeling?

Threat modeling is a core activity and a fundamental practice in the process of building trusted technology; it has been identified as one of the best "return on investment" activities with respect to identifying and addressing design flaws before their implementation into code. It aims to identify the attacks a system must resist and the defenses that will bring the system to a desired defensive state. These attacks expose and exploit potential weaknesses that will affect the system being modeled in negative ways.

"System" in this context is defined very broadly to include any type of computer system, including small pieces of software functionality, discrete software applications, complex integrations involving multiple hosts, multiple applications and runtime execution environments, infrastructures and even distinct legal entities.

The express aim of threat modeling is to identify and eliminate *design* issues: to identify security weaknesses or arrive at a set of security needs that must be built. These are sometimes referred to as "requirements." We are using the term "requirements" in this document to mean "security issues that need to be addressed." In other words, "requirements" refers to any required item that must be implemented and does not necessarily refer to formally generated requirements.

Once identified, the security requirements when implemented will bring a system or set of systems to the intended security posture. Identifying likely threats and the probable consequences of successful attack is the method of investigation to identify an appropriate set of defenses. It is an industry best practice to validate the defenses that were derived from the threat model.

While some threat-modeling methods focus on identifying threats and security issues, other methods also perform an assessment of the resulting risks by rating the consequences (impacts) and the likelihood of threats. Such methods are also called Threat and Risk Analysis or Assessment (see, for example, ISO 27005 [23], NIST SP 800-30 [24]). Such a rating can be used to prioritize defenses.

3. When To Do Threat Modeling

Ideally, threat modeling is applied as soon as an architecture has been established. There is a timing element to threat modeling that we highly recommend understanding. No matter how late in the development process threat modeling is performed, it is always critical to understand weaknesses in a design's defenses. The cost of addressing issues will generally increase when we uncover design misses and missed security requirements later, or worse, at the end of the development process. It is much more useful to begin the process of identifying potential attacks and their treatments while identifying other system requirements.

A threat model should begin when the major structures, the major components or functions of an architecture, are known. Before this point, much time might be wasted redoing work as structure changes. Beginning too long afterwards might mean that significant structural changes or additional structures required for security will be uncovered only after the timeframes and resources allocated for development have been exhausted. Broad requirements and constraints help to define an architecture, and these necessarily become more specific as things become better defined. Security must be among these and present from the start, becoming "built in" rather than "bolted on."

Thus, threat modeling can be used as part of requirements engineering to derive security requirements, based on a first architecture overview, or threat modeling can be used as a design analysis technique, being applied to the software design before coding starts. Threat-modeling techniques might focus on one of these use cases.

Though teams are encouraged to perform threat modeling early in their structural definition process, if that cannot be achieved, threat modeling is still a useful exercise regardless of how close the system is to deployment or how long the system has been in use. The next development cycles may be used to mitigate risks that a system currently carries. Even when changes to a system are not expected, organizational decision-makers may wish to understand any significant risks a system may add to the organization. Note that there is a distinction between end of development and end of support, and even when active support has ceased, a proper threat model will bring clarity about the possible flaws in the system, which will inform a decision on further investment in the product.

4. Updating the Threat Model

It may not always be clear to a team working on a given release whether or not the threat model needs updating. A partial list of suggested revision triggers is as follows:

1. Changes affecting the processing, handling or classification of data by your software: for example, changes to sensitive content, parsing and handling input (user and/or automated), formatting data streams, changes pertaining to cryptographic algorithms, keys and key management, etc.
2. Addition of a new sub-component, database or data repository, even if the change appears to be minor and not directly related to security
3. Any additions or changes in security controls and functionality:

- Authentication
    - Adding or changing an authentication method, or mechanism
- Authorization
    - Shifting the trust relationships between any components or actors in the system (change of user levels, change of data access permissions, etc.)
    - Adding or changing an authorization method, or mechanism
- Logging, monitoring and alerting
    - Adding or changing application monitoring, business analytics and insight, auditing and compliance requirements or forensics
- Cryptography
    - Adding or changing cryptographic functionality: hashing algorithms, salt, encryption/decryption algorithms, SSL/TLS configuration, key management, etc.

4. Introducing or changing communication channels between subcomponents, the back end, etc.: a new data flow might need to be authenticated, authorized and protected in transit.

As a rule of thumb, changing or adding data that is externally produced or internally consumed – for example, data stores, log files, webpage contents, descriptive error messages, temp files, etc. – should be considered likely triggers for reconsideration of the threat model.

Because it is impossible to come up with a comprehensive checklist for everything that invalidates a threat model, please use the list above as a reference and make any necessary adjustments to fit your software development needs.

Please note that sometimes revisiting the threat model might produce no actions other than confirming that the threat model is still up to date.

5. How To Do Threat Modeling

The process of threat modeling usually involves some of these distinct but closely related sub-activities:

- An initial, possibly incomplete, description of the structure, use cases, misuse and abuse cases, and resources the system is subjected to or constrained by. This is often represented as a diagram (e.g., a data-flow diagram, DFD [7]) that describes the system and maps (some of) the potential attack points from outside the system. It is supported by annotations about the internals of the system, data transformations and storage, and particulars such as deployment modes or asset descriptions. This can be done at varying levels of formality, from specification documents to drawings on the back of an envelope – but the description must accurately depict the system being modeled.
- The identification of a set of possible threats that would be relevant to the system being analyzed, how they would present themselves in various possible scenarios and what could be done to mitigate them.

There are some popular ways of expressing this description and identification:

- Architecture representations (usually logical or component view, but not limited to these views) [6]
- Data flow diagrams (DFDs) [7]
- Attack sets as attack trees [1]
- Mind maps
- Libraries of possible threats [2]
- Lists of security objectives, attack vectors and potential mitigations
- Dynamic media, such as the use of white-boards to foster stakeholder interaction
- Sequence diagrams [21]

The specific format is less important than its utility to the modelers. As long as it is sufficiently expressive, containing all important details without being overwhelmed with unimportant information, and the threats are relevant and well-defined, the modelers will usually converge to a "language" that works. In general, a representation will contain more than one of the items listed above; sometimes most of them. Still, in most threat models one or more of the included elements will be incomplete. Threat models are living documents subject to revision as more information becomes available.

There are many possible ways of performing threat modeling, and the consensus is that there is not one single perfect way. A valid process is one that is repeatable, manageable and, above all, able to identify potential threats. This document aims to provide guidance on some of the theory and lessons learned in the field that should be of immediate applicability, without dictating a specific methodology. Use the material below as guidelines to identify or develop a methodology that will work for your particular case.

As previously noted, threat modeling identifies the threats a system must resist and the defenses that will bring the system to a desired defensive state. The desired defensive state is described in the form of a set of security requirements (also referred to as security controls) for the system that needs to be implemented.

This set of security requirements leads to security testing requirements that define the security testing scope of the system. If the security testing requirements are not defined, security testing will be done blindly, increasing the cost of the effort and reducing the comprehensiveness of the test.

Traditionally, the objectives of a system are defined in two categories of requirements:

- Functional requirements: these define a specific behavior or function of a system.
- Non-functional requirements: also referred to as quality requirements, these specify criteria that can be used to judge the operation of a system, rather than specific behaviors.

Non-functional requirements cover a range of areas, including but not limited to:

- Security and privacy
- Accessibility
- Responsiveness
- Scalability

Each security requirement is generally composed of three parts:

- **The problem:** the potential weakness being addressed. Some practitioners map findings to the Common Weakness Enumeration (CWE) database [17] when appropriate.
- **The control:** this is the task that needs to be done or the operation that needs to be performed. Generally, it is written in a technology-agnostic language and focuses on the acceptance criteria without specifying how they can be achieved.
- **Implementation guidance** (optional): this explains how the control can be achieved. While this is an optional component, it is strongly recommended that you include specific guidance on how the requirement can be achieved in the appropriate technology stack.

By the nature of the development process, threat modeling is also a good opportunity to identify constraints that should be applied to the implementation, such as choice of controls to mitigate identified threats, and to raise possible red flags to be considered at implementation and testing. These too have a place in the threat model findings and results.

6. Failing at Threat Modeling

When learning a new process like threat modeling, it is important to see how it can fail, as much as how it can succeed. It is not just about how flaws and threats might have been missed, but also about failures in the threat-modeling process itself. For instance, is it a failure to think threat modeling is not necessary because the product undergoes penetration testing and code reviews? Is it a failure to think there is no reason to do threat modeling because the system is already deployed and no breaches have been detected?

In a talk [11] given in early 2016, SAFECode members Jim DelGrosso of Cigital and Brook Schoenfield of Intel addressed six myths of threat modeling that might well be considered failures:

- "We already do pen-tests with tools AND people … we don't need to do threat modeling."
- "The system is already built and deployed … there's no reason to do threat modeling."
- "We did a threat model when the system was built … we don't need to do it again."
- "Threat modeling is too complicated."
- "We don't have software security experts, so we can't do threat modeling."
- "I'm doing threat modeling at all the right times ... there's no reason to do pen tests or code reviews or <whatever> anymore."

These might be classified as failures of mindset – they present a barrier of entry to threat modeling by trying to stop it from happening or justifying it away. On the other hand, we have practical failures caused by failing to adhere to a proper methodology:

- Failing to control scope of the analysis. It is rare that the team has as much time as needed or desired, so there needs to be control over what will actually be analyzed.
- Focusing on areas that are understood really well. For example, it has been observed that cryptography gurus devoted considerable time digging into the nuances of crypto, even though the use of crypto may be unnecessary.
- Not defining "success." When building out a threat-modeling program, you must define what "success" looks like. Is it the identification of a defect that could not be found through your penetration testing and code review efforts? Is it identifying that some flaw is occurring under some particular set of conditions? Is it building the set of security controls that will drive design and implementation? Perhaps something else?

Threat modeling is a human activity best practiced with a range of expertise. Major pitfalls can occur when stakeholders with key knowledge are not included in the process, or when the team fails to agree on an up-to-date view of the system, its "moving parts" and communication mechanisms. Many examples of threats that failed to be identified at threat-modeling time are in the categories of "things that fell between the cracks during design" and failures to communicate, as some of the examples later in this paper will show. Failures of communication are responsible for "Eureka moments:" one designer explains how a certain part of the system works, and is interrupted by an implementer who clarifies that that is not exactly how it is working; the team collectively experiences a sudden understanding that previous assumptions were perhaps erroneous. This demonstrates the importance of including key stakeholders and making sure the formats adopted allow for clear and precise communication of details.

Attack surfaces that have not been completely addressed are another common pitfall encountered during threat modeling, which can derive from a number of communication failures. For example, expectations about interfaces or interactions with external systems may not have been explicitly expressed and fully understood while threat modeling. Sometimes the inclusion of functionality provided by a third-party component has not been accounted for – or even known during a threat-modeling session. Each of these misses, when discovered, will require revisiting the threat model.

Finally, there are distinct perspectives for what the threat model is intended to do: it will identify specific threats, document the topology, identify systemic failures in the control environment, and it may locate policy violations that are not specific security threats. Using a threat model to focus exclusively on threat identification and not on the full set of valuable intelligence, it can offer would be a common but costly mistake.

7. Building a Great Team – People

When conducting a threat model it is critically important to include the product owner and technical subject matter experts (development, network design, operations), to the degree they are available. There is also value in having quality engineering/assurance people involved, and sometimes even the people responsible for documentation and release engineering.

The technical subject matter experts must include those people who understand the gross structure of the entire system ("solution architects"), people who are structure experts for defined portions of the system ("architects"), and people with expertise from domains as diverse as networking, hosting, operating systems, deployment vehicles and processes, cloud, quality and quality assurance, integration and software design. Even implementers may have key information that will improve the comprehensiveness and applicability of the threat model.

It has been said that "threat modeling is a team sport." Reducing the size of the team due to schedule and other limitations may sound like a good idea in the beginning, but over time the quality of the resulting threat model tends to decline, as much "tribal knowledge" is missed by the smaller set of individuals involved in the process. With that said, a threat model is a living document, and as such, it is subject to repeated revision. We recommend that you create a draft that is filled in as the process evolves and more knowledge is gained. Filling in a draft threat model is an organic process that builds understanding in a team and can involve more expertise within the team as the process unfolds.

Apart from technical expertise and knowledge of the system, it is useful to have individuals with a variety of approaches and talents in the group. For example, you might include a critical thinker who is able to mimic the exploration process done by an attacker, probing at sensitive points, and a process-oriented thinker who is able to unravel the sequence of events and transformations of data as it works its way through a system being modeled. These two, while offering very distinct views, should be able to complement each other and float up a larger number of possible threats to be evaluated. In practical terms, at least some people in the group should have experience and knowledge in "how to break software."

If the system will serve customers who have specific needs such as a government's Justice Ministry or those affected when privacy issues cross borders, then it might be beneficial to have legal counsel involved during threat modeling.

7.1. Selecting Good Threat Modelers

Effective contributors to a threat-modeling process would have a background that includes software engineering and architecture knowledge, at a university level or equivalent, enterprise architecture and a basic knowledge of software security (e.g., definition of basic threats). It is not necessary to have a computer science degree, as all of these skills can be acquired with experience, but they must be present in the team.

8. Threat Modeling Scope

Defining scope is critical in conducting threat modeling: where do you start and where do you stop?

- How deep do the threat modelers go into the structure of the system?
- What are the prerequisites for beginning a threat model?
- What constitutes completion of the model?

How much of a system to investigate when threat modeling is often dependent upon the size and complexity of the system, as well as factors such as the runtime/execution environment, the typical environment in which the software will be run, any infrastructure or services that the software is expected to consume or use, language(s) in which the software is written, how it is deployed to be used, etc. Hence, there is no clear definition that applies to every system with any particular combination of factors.

As for prerequisites, some practitioners suggest listing every asset coupled to every possible technical attack. Others contend that initial lists need to include every possible vector from which a threat may materialize, as well as all possible points in the system where mitigations and controls can be implemented. However, having enumerated such lists, threat modelers will be faced with the problem of culling these lists of unlikely attack scenarios and assets whose loss would be insignificant or would not interest likely attackers. Prioritization of what controls to actually implement, no matter what approach is used, will always be important in the face of limited resources.

The problem then becomes knowing when to stop – once the prerequisites exist and a methodology has been selected, how do we know the work is complete? A general rule of thumb might be that the threat model is complete when all the attack surfaces whose compromise could affect the organization or the product in some significant manner and which are exposed to attack have been enumerated. Further, the threat model may be considered to be complete when all the likely controls to protect the attack surfaces have been defined.

At whatever level of granularity the system under analysis exists, whether it be a discreet application, a library, a set of integrated systems or an enterprise architecture, a boundary to the analysis has to be agreed upon at the beginning of the threat-modeling exercise, in order to define an exit point at which it can be considered done.

The definition of "done" may include guidance covering when to refresh the threat model. Although some treat it as a living document, subject to change as the system evolves, other views suggest that structural change to the architecture, the existing security controls, or any design element that bears upon the security posture of the system are the events that should trigger a holistic threat model review. If there have been no significant changes, when the only changes being made are non-security features, then a comprehensive threat model will "stand" and not need updating.

9. Methodology

A non-exhaustive but representative list of examples of industry-accepted methodologies and best practices for threat modeling would include at least these:

- **Microsoft Threat Modeling Process:** a step-by-step approach to threat modeling that focuses on identifying assets and architecture, decomposing the application, identifying and documenting the threats, and ranking them in order of criticality
- **P.A.S.T.A (Process for Attack Simulation and Threat Analysis):** a seven-step process applicable to most application development methodologies, which is platform agnostic. It not only aligns business objectives with technical requirements, but also takes into account compliance requirements, business impact analysis and a dynamic approach to threat management, enumeration and scoring.
- **Trike:** a methodology used to perform threat modeling that focuses on a requirements model designed to ensure that the level of risk assigned to each asset is classified as acceptable by the system's stakeholders
- **ATASM:** Architecture, Threats, Attack Surfaces, and Mitigations is a threat-modeling approach that highlights the importance of structural understanding of a system for the purpose of threat modeling (architecture). The architecture is broken apart into its logical and functional components (decomposing and factoring) to discover all potential attackable surfaces (inputs and outputs of the system). Decomposition is also used to define those points at which defenses will be built (mitigations are placed at defensible boundaries).
- **Threat Library/List Approach:** using a pre-defined set of common and prevalent threats, a team will try to identify instances of them in the product by tracking the triggers -- for example, cross-site scripting might be present if a product offers a web interface and no input validation and output sanitization. The team is free to evolve the threat library as technologies and frameworks change [2].
- **Lightweight/Rapid Threat Modeling:** there are a range of processes that use lighter-weight variations of other methodologies, and additional approaches that use quick classifications and other ways to achieve a similar result in less time for less critical systems [18].

Besides methodologies for threat modeling, there are others for threat enumeration and discovery, such as Microsoft's STRIDE [12], or using the many "Top X Threats" lists (for example, OWASP's Top 10 [13]; there are others available) as a basis for a threat library, and for the ranking of findings there are options including CVSS (Common Vulnerability Scoring System) [14], Open Group™ Factor Analysis of Information Risk (FAIR), CWSS (Common Weakness Scoring System) [15] and CWRAF (Common Weakness Risk Analysis Framework) [16], each with particular applications and scenarios to which they might best apply. One helpful criteria is to differentiate between vulnerabilities and weaknesses, as explained in section "Terminology." CVSS targets vulnerabilities while FAIR and CWSS target weaknesses. There are other commonly used risk methodologies as well.

STRIDE may require more initial experience with threats than might be available without a security expert, while the Top X lists offer those attacks that have been most prevalent at a given time, built from statistics provided by practitioners. CVSSv3 (and previously, v2) is rapidly becoming the de facto standard when ranking vulnerability scenarios in order to evaluate their criticality and allocate resources for their mitigation.

Each organization will adopt a different set of practices or variations on these processes that suits its needs. Therefore, instead of choosing a specific approach over others, in this paper we address the common challenges that frequently arise with most of these approaches, with recommendations based on the lessons learned from SAFECode member organizations. They can be adapted to most approaches and can have significant impact on the success of your practice.

10. Terminology

Incorrect usage of terminology can cause confusion and leads to lack of buy-in. The first step toward a successful practice is educating the stakeholders on the terminology and the correct distinctions among terms. "Threat," "risk" and "vulnerability" are often used incorrectly and interchangeably to refer to different concepts. See the glossary at the end of this document for formal definitions.

10.1. Weakness vs. Vulnerability

Common Weakness Enumeration (CWE) [17] is a catalog of weaknesses that aims to enable communication of weaknesses between systems or organizations. For example, cross-site scripting is identified as CWE-79: "Improper Neutralization of Input During Web Page Generation." The catalog contains descriptions, consequences, likelihood of exploits, examples, etc.

On the other hand, the Common Vulnerabilities and Exposures (CVE) [22] is a dictionary of publicly known information security vulnerabilities and exposures found in existing, implemented and deployed systems. For example, the CVE entry CVE-2014-0160 is known by its popular name "Heartbleed": a specific input validation weakness as applicable to certain versions of OpenSSL libraries when handling Heartbeat Extension packets.

The distinction between weaknesses and vulnerabilities is that, before the implementation of software starts, there is no vulnerability that can be associated with the project, but there might be weaknesses that can be identified during threat modeling.

11. Handling Complex Systems

Threat-modeling complex systems that contain multiple components with different roles can be challenging. This is encountered, for example, in Internet-of-Things systems that may contain various components such as a separate cloud service, web application inside devices for interactions and configuration, multiple sensors and so on.

The recommended approach is to examine the system at multiple resolutions for threat-modeling purposes. For example, at a two-level approach:

- A team would first model the system at a high level, focusing on the interaction between large components.
- Then for each large component, the team will model the component separately, focusing on the component itself and the interactions within it and with other components.

The benefit of this approach, in addition to breaking the activity into manageable pieces, is that it will result in a smaller and more contextually focused set of security requirements for each component, as applicable to that component.

12. Technologies/Tools

Many of the processes that support and enable threat modeling can be automated or at least made more accessible by the judicious use of tools.

While there are both free and commercial tools that may aid the threat-modeling process, as of this writing there is no substitute for human analysis and, importantly, experience in threat modeling. It will take time and patience to deliver a well-crafted threat model. It will take time, perseverance and probably quite a few mistakes to build threat-modeling expertise in an organization. No single tool can offer a "silver bullet" for the process of threat modeling.

Late in 2015, many of the members of SAFECode joined in a birds-of-a-feather discussion on the subject of identifying a tooled workflow to enable both beginners and seasoned practitioners to perform and improve their threat-modeling processes, and they agreed on the present state of the field:

- Current existing solutions fail to address sufficiently the growing needs of security practitioners.
- The demand for and interest in threat modeling is ramping up.
- There are few tools available, constituting a barrier for entry.
- There are few opportunities of mentorship and training available to overcome that barrier.
- Starting a threat-modeling practice can be challenging.
- The whole effort is sometimes seen as purely time-consuming, as results are not always immediately evident.

In order to fulfil the birds-of-a-feather group’s vision of an appropriate tool, a solution would have to deliver value as a security tool solely by virtue of its use – that is, the use of the tool would generate security findings that would be of immediate relevance to the user. It would generate run-of-the-mill requirements -- for example, "you accept input from an uncontrolled source, therefore you need to have proper input validation" –- and still be expandable and extensible so that it can address the unique needs of the product being modeled, the domain where it exists, internal policies and requirements, as well as enabling experienced practitioners to add their own findings and input.

Based on these initial requirements, the group agreed on a wish list that would serve as a basis either for development of a tool or for evaluation of existing offerings:

- A solution should enable modeling of architecture, by diagramming, and linking of separate models to create larger systems.
- It should provide an annotation framework that encourages creation, extension and customization of architectural annotations and analysis, and custom guidance for issues identified during analysis.
- The analysis process should make use of the diagramming and annotations to identify issues and provide solutions, and accept findings created in free form by practitioners.

- The findings need to be trackable in their state (mitigated, N/A, etc.) and ranked according to some of the existing industry standards, like CVSS [14] or Open GroupTM Factor Analysis of Information Risk (FAIR).
- The solution should be agnostic about methodology, supporting whichever one is chosen by the end-users.
- It should provide report generation and version-to-version tracking of models, with inheritance of unmitigated previous findings.
- As additional nice-to-haves, the tool should enable import/export of its models to industry-standard tools to enable presentation and sharing (for example, Visio diagrams, or more robust existing architecture modeling tools).
- And finally, the tool should support the use of distinct overlays, annotations and analysis rule sets to support other forms of modeling analysis, such as resilience, recovery, privacy data flows, etc., allowing for multiple uses of a single model.

Other practical aspects were brought up, such as the tool being platform-agnostic and supporting concurrent multiple users for collaboration, but these were considered minor.

With these in mind, we see that today there is no single tool available that fulfills every single point on the checklist, but there are alternatives that offer partial help. This is by no means a complete list, and we do not stand behind any particular tool as being "best of breed" or similar. There are commercial tools that offer some of the more specialized functionality found in the list, to varying degrees of success and independence from any given methodology, and a quick web search will come up with some of those.

- Microsoft Threat Modeling Tool 2016 [3] -- free to use, offers diagramming, annotations and a rule set that can be modified by the user
- Mozilla SeaSponge [4] -- free to use, browser-based, offers diagramming (with multiple linked diagrams) and annotations. Offers loading a configuration file to enforce standards. No rule set or rule application

Some enterprises have pointed to their work developing an internal tool; others have had experience adapting existing tools to their needs. Some list the use of applications like Visio for diagramming and Excel for bookkeeping of findings, and these are all perfectly valid alternatives. At the end of the day, you should be able to find tooling that will permit your own workflow to evolve, the characteristics of your product to be captured, and findings to be identified and communicated in a clear, concise and ranked way, so that they can be properly mitigated.

13. Threat Modeling Within a Development Life Cycle (SDLC)

Threat modeling is an activity best executed while creating the structure of systems (architecture) and especially, to discover what security elements, features, requirements and constraints should be designed (implementation specification). That is, threat modeling delivers its best, most useable results when performed early, before implementation has begun.

It does not matter what form of software development life cycle is used: Agile, Waterfall, Continuous Evolution/Improvement, Branched, what-have-you. The security requirements and constraints must be determined before the product can be built. Secure design naturally follows a threat model that is as complete as it can be, given what is known about an architecture and the design that expresses it at the time of the threat modeling activity or a threat model update. As the learning from iteration is incorporated into a design, the threat model may also have to be iterated, along with design changes.

> *"If the architecture is changing, then the process should start at architecture assessment and threat modeling. The assumption is that the existing architecture has been assessed and threat modeled to ensure that appropriate security requirements have been built into the design. In cases where there never has been an architecture assessment and threat model, the architecture should be treated the same as a greenfield project.*
>
> *"Even if there are no additions or changes to the existing architecture, adding any feature with security implications indicates the necessity for design work. The design makes the architecture buildable. Programmers work from the design. So it's important that any security requirement or feature be designed correctly and completely. Security expertise is critical; the purpose of the design review is to ensure that the appropriate security expertise is applied to the design."*
>
> **– Schoenfield, Brook S.E., "Applying the SDL Framework to the Real World," Core Software Security, Ransome, James, Misra, Anmol, CRC Press, 2014, p. 279**

The following graphic illustrates the placement of threat modeling within the activities that are usually performed while architecting and designing software and systems. Threat modeling is generally not the only security activity (SDL -- secure development life cycle -- activity) that is performed. It may not be the first activity.

When threat modeling is the only design-time security activity, its placement is still roughly the same: early within a development life cycle. Threat modeling is performed when there is sufficient understanding of the basic structure of the system that it can be threat-modeled. In other words, enough stability to system/software structure has to be established that threats can be uncovered for that structure, since each structure and set of integrations will have some uniqueness, some local variation, even in fairly constrained or highly specified environments. Threat modeling often begins later in the architecting process or at the beginning of design. It may be thought of as a security "bridge" between architecture (structure) and design (the specification for what and how to implement).

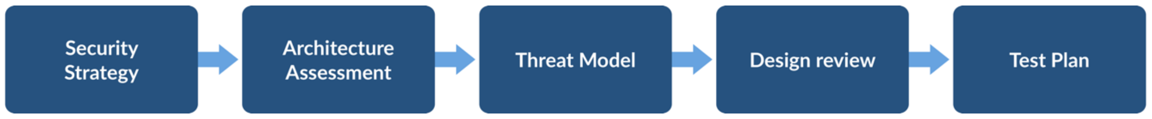

Figure 1. Architecture task flow when a project is new or a redesign – Ibid, p. 278

(Note that the graphic above does not depict an entire secure development life cycle nor a complete software development life cycle, but rather only the portion of a conceptual SDL focused on secure architecture and design.)

14. Threat Modeling Examples

The following examples illustrate principles and methodologies described above. They are by no means complete or exhaustive, and in fact might present some further issues that would serve as interesting points of discussion and exercise if the reader would like to continue the reasoning and methodologies sampled. These analyses are not, to the best of the knowledge of the authors, representative of any specific system, deployed or in development.

14.1. Web-based User Feedback System

We will use this common use case found in web-based systems to illustrate a generic system:

- User registers in the system for the first time
- User can then log in using the registered username and password
- User will enter feedback comments and then log off system

The following data flow diagram represents the use case described above:

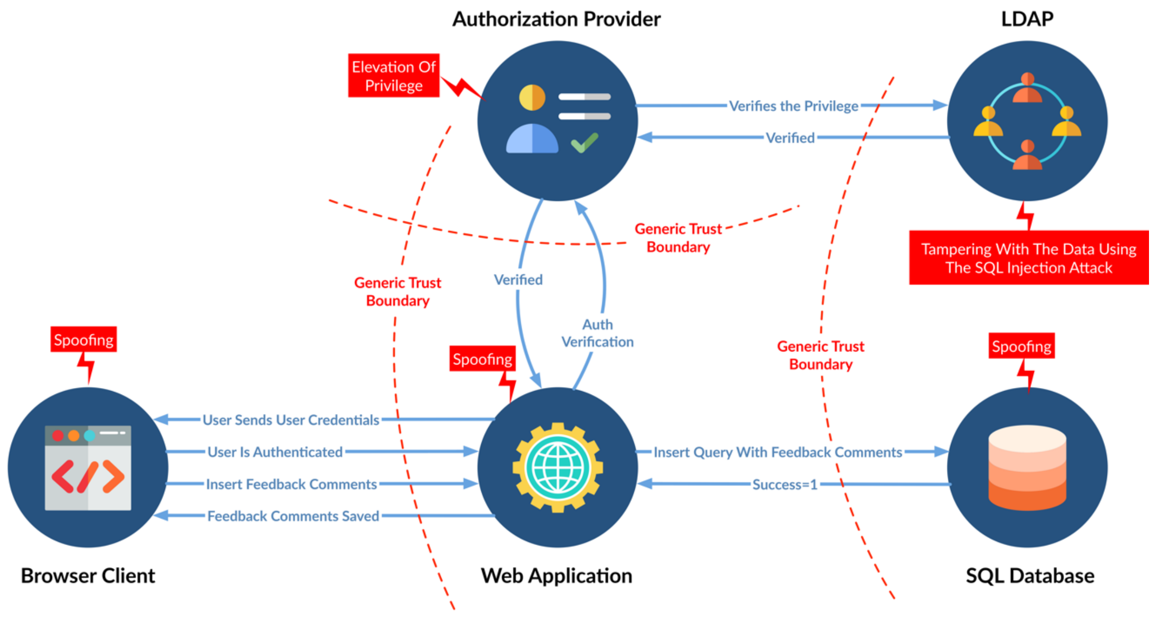

Figure 2. DFD for a simple web application

Using STRIDE, listed here are some of the threats possible against this system, organized by class of threat:

Spoofing	Tampering	Repudiation	Information disclosure	Denial of service	Elevation of privilege
The web application may be spoofed by an attacker and this may lead to unauthorized access to the browser client. (When discussed with the design team, it appears that this is not possible, as a security mechanism to identify the server is present.)	SQL injection is an attack in which malicious code is inserted into strings that are later passed to an instance of SQL Server for parsing and execution.	The SQL Database claims that it did not write data received from an entity on the other side of the trust boundary. Consider using logging or auditing to record the source, time and a summary of the received data.	Data flowing across saved feedback comments may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. Consider encrypting the data flow.	The browser client crashes, halts, stops or runs slowly; in all cases violating an availability metric.	The browser client may be able to impersonate the context of the web application in order to gain additional privilege.
The SQL Database may be spoofed by an attacker, and this may lead to data being written to the attacker's target instead of the SQL Database. (When discussed with the design team, it appears that this is not possible, as SQL connection is made using the database username and password.)	LDAP injection is possible, as the user authentication is verified using an LDAP query.	The web application claims that it did not receive data from a process on the other side of the trust boundary. Consider using logging or auditing to record the source, time and a summary of the received data.	Improper data protection of LDAP can allow an attacker to read information not intended for disclosure. Review authorization settings.	An external agent prevents access to a data store on the other side of the trust boundary.	An attacker may pass data into the browser client in order to change the flow of program execution within the browser client to the attacker's choosing.
	Data flowing across saved feedback comments may be tampered with by an attacker. Failure to verify that input is as expected is a root cause of a very large number of exploitable issues. Consider all paths and the way they handle data. Verify that all input is verified for correctness using an approved-list input validation approach.	LDAP claims that it did not write data received from an entity on the other side of the trust boundary. Consider using logging or auditing to record the source, time and a summary of the received data.			

The diagram below is a representation of a sample attack tree, which illustrates the potential vulnerabilities that could result from exploiting a specific weakness within the application that is being threat-modeled:

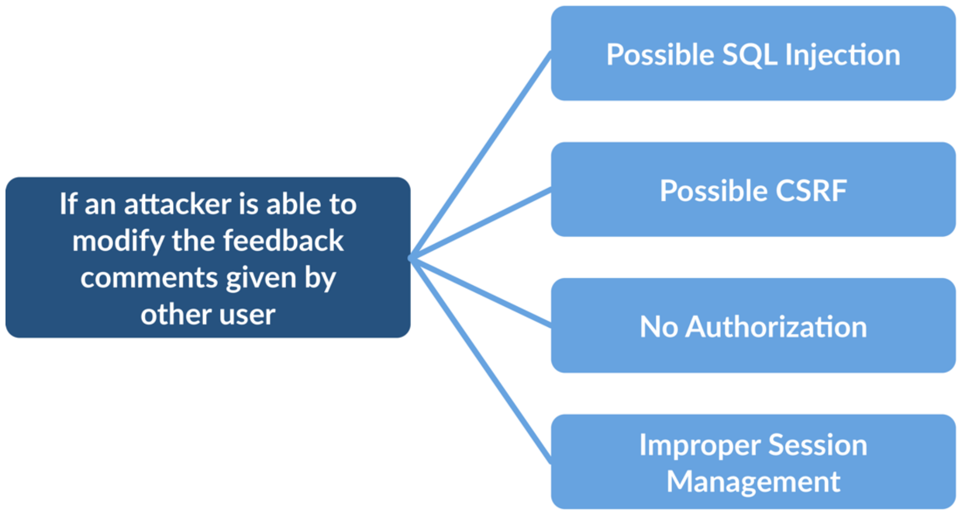
If an attacker is able to modify the feedback comments given by other user
Possible SQL Injection
Possible CSRF
No Authorization
Improper Session Management

Figure 3. Sample attack tree

Once we have the possible threats, we are able to devise mitigations that will enable us to minimize the risks associated with the threats.

14.2. Authentication for the Internet of Things (IoT)

In an IoT paradigm, people connect to web services using personal computers, laptops and smartphones, and control or receive information from a diverse set of devices (things) incorporated in various sections of our homes, cars, workplaces or even bodies. Such devices (things) can help us to create a more connected life experience. IoT devices are normally equipped with different types of sensors to gather information from what happens around us. IoT systems collect this information and provide interesting insights to users out of the processed data.

An important point about IoT devices is that they are normally less capable than our regular laptops or even smartphones. Limited resources on the devices and their low power restrictions hinder the usage of security best-practices, thereby widening the attack surface and the possible threats, as we have seen with recent DDOS attacks: "How Hacked Cameras Are Helping Launch The Biggest Attacks The Internet Has Ever Seen" and "Hackers Used New Weapons to Disrupt Major Websites."

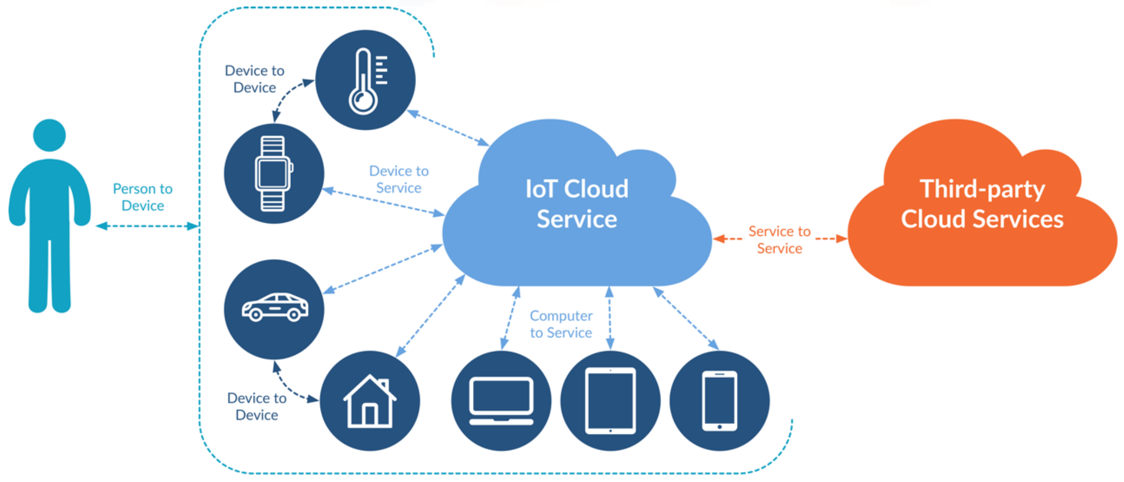

Figure 4: A sample IoT architecture

The figure above shows a very common architecture for IoT systems, especially in the domain of home automation. In a threat-modeling context, it is a high-level DFD that puts together different types of communication channels between various types of entities involved in an IoT solution.

Authentication is crucial to reduce the number of access possibilities for an attacker to explore. The authentication concept and the associated threats differ based on the communication channel type, as follows:

1) Person to device communication channel:

As the number of connected devices increases, it is important to protect them from the threat of access by unprivileged users. Just think of your smartphones without their authentication capabilities. What if you could not set any PIN, password or pattern on your smartphone? Anyone close to your phone would be able to peek at your private photos or read your texts. It is important to use devices that have at least minimum capabilities for authenticating users. A small keypad to enter a PIN, an enhanced touchscreen to enter a pattern or a password, or even a fingerprint reader would boost the security of the entire system by allowing only privileged users to access or control the devices directly. It is also important to make sure that the default passwords set on the devices are changed during the initial setup by the users.

2) Device to service communication channel:

One of the major challenges in this layer is the significant increase in the number of connected devices in the IoT world. Traditional IAM (Identity and Access Management) systems concentrate on people and managing the access privileges and attributes associated with users. However, in the IoT ecosystem we have to deal with many more devices that not only send information to the IoT service, but also can accept control commands from it. It is crucial for IoT systems to be able to handle the increasing number of devices and establish mutual authentication between the device and the cloud service. Most of the existing IAM solutions are not scalable enough to address these requirements, thereby posing a confirmed threat. New types of identity management solutions such as IDoT (Identities of Things -- https://kantarainitiative.org/confluence/display/IDoT/Home) and IRM (Identity Relationship Management -- https://kantarainitiative.org/irmpillars/) attempt to mitigate this threat.

An implementation challenge in this area is assigning unique identifiers to each device and matching those identifiers with a specific user during initial setup. Initiatives such as EPC (Electronic Product Code – https://en.wikipedia.org/wiki/Electronic_Product_Code) and "ucode system" (https://en.wikipedia.org/wiki/Ucode_system) are trying to provide unique identifiers for this purpose. A successful IoT service must be able to deal with these types of identifiers or similar ones and authenticate each device based on its unique identifier. The unique identifiers should be incorporated in the devices before their initial setup by users.

For the authentication process, devices can use either symmetric or asymmetric algorithms. In symmetric authentication, each device is equipped with a cryptographic secret key, other than its unique identifier, that is shared with the central IoT service. Using the shared secret key, the device and the IoT service can mutually authenticate each other. For asymmetric authentication, each device and the central IoT service must be able to deal with Private/Public keys signed by trusted certificate authorities. In both cases, the devices should protect their unique identifiers and secret keys by storing them on tamper-proof chips. Not doing so would expose sensitive information that could be readily used by attackers to authenticate themselves as valid users.

3) Computer (PC, Laptop, Tablet, Smartphone) to service communication channel:

This type of connection is very common in current web-based services, and personal accounts employing a username/password are commonly used to authenticate the users of a service. Similarly, we can use the same techniques in our IoT design. We just should make sure we restrict the use of simple passwords, limit the number of wrong password trials, and provide clear and secure interfaces for resetting forgotten passwords. These provisions would suitably reduce the account compromise threat arising because of an authentication control weakness.

4) Service to service communication channel:

In the future connected world that we are heading toward, it will be essential to share information with other systems and products. Users will not be bound to a single product or manufacturer for all their needs. Diversity is a crucial factor in being able to scale quickly enough in the IoT world. It is important to let our users allow third-party services to access their information in our systems with their explicit consent. For example, users of a third-party health service may want their IoT devices to gather health-related information and transmit it to that service. Without sharing capabilities, a manufacturer's product would be secluded from the rest of the IoT ecosystem.

Accordingly, authentication and authorization of third-party services are essential to any IoT system. Specifically, dominant and robust authentication protocols like OAuth and OpenID, which are widely used in the web ecosystem, can also be incorporated by any IoT system. These protocols can enhance our IoT systems so that they can securely authenticate third-party services and control their access to users' information by explicitly asking for users' agreement.

5) Device to device communication channel:

In some cases, our IoT system may have to provide peer-to-peer connections between individual devices. With respect to authentication, these kinds of communications are challenging to deal with. Our options are to inject a single shared secret key into all the devices to let them communicate securely or to make our devices capable of authenticating other devices in the same way as the central IoT. Limited resources on the IoT devices limit the scenarios for the latter case. However, the first option is also not ideal. The attackers would be able to access any device just by breaking one of them and getting their hands on the single shared secret key. Most systems avoid these kinds of peer-to-peer connections and require the devices to communicate through the more capable, central IoT service.

Is that it? No!

Addressing authentication-related threats would significantly reduce the attack surface. But, that is just one piece of the equation. Post-authentication threat scenarios are equally important. For a low-powered device such as a refrigerator, here are a few threats along with their potential impacts and mitigation:

1. A government representative's life coming under threat upon reaching home due to his/her refrigerator being directed to draw excessive power that would lead to a short circuit and a fire;
2. A competitive threat where a vendor hires IoT hackers to change refrigerator temperatures to make it ineffective: the contents rot, and buyers are left with no other option than to buy a competing product. Mitigation for (1) and (2): preventing critical field values from the fridge from being altered in a man-in-the-middle attack;
3. A competitor becoming an authenticated user by buying a device – this could lead to threats involving reverse engineering, negation of security-by-obscurity (https://cwe.mitre.org/data/definitions/656.html), etc.;
4. Hardware threat: back doors that chip manufacturers might be putting in, in the absence of supply chain assurance.

15. Threat Modeling and Agile Development Practices

As stated earlier in this document, threat modeling is an activity that belongs to the architectural and design stages of your SDLC. When using the Agile methodology for software development, the design stage might not be as clearly defined as it would be when using other development methodologies, such as Waterfall. Some of the challenges may include trying to fit threat modeling to relatively short sprints.

A pitfall may lie in trying to create threat models for each sprint and then trying to merge them in order to threat-model the entire system. At the beginning of a set of sprints, not enough is known to threat-model the whole system, because every sprint may affect the threat model.

In this section, we offer some recommendations on when best to do threat modeling in Agile development, and how to reduce overhead so as to minimize the impact on your development schedule.

15.1. When To Do Agile Threat Modeling

Even though with the Agile methodology there are often multiple teams working concurrently on different user stories, there should always be sprint planning for the entire release, where the sprints and user stories are distributed. Additionally, most people use Sprint Zero to prepare the basic skeleton for the project. Whether or not Sprint Zero is used, the process of threat modeling should start before Sprint 1. Specifically, at least a draft threat model should be completed by the time the teams are ready for Sprint 1. Thereafter, while developing user stories within each sprint, you need to be mindful about whether this user story invalidates or perhaps requires an update to your threat model. The table below summarizes these steps:

Sprint 0/ Sprint Planning	Sprints 1 through N	Release Definition of Done (DoD)
Starting and completing the release threat model (TM) based on the overall design	Update the TM only if new user stories or source code changes invalidated the original TM.	Make sure that the TM reflects the system state, is up-to-date and controls are implemented.

(Also see the section on page 4 of this document, "Updating the Threat Model.")

15.2. Threat Modeling in the DevOps World

DevOps typically embraces development processes that enable frequent software changes, similar to Agile and Continuous Integration (CI) processes, with emphasis on automation. Threat modeling in DevOps differs from what is seen in Agile and CI in that it also includes a focus on enabling frequent operational and infrastructure changes, typically driven directly by the development process. Most DevOps processes minimize or outright eliminate the traditional change review and management processes that were historically typical in service infrastructure. Without a robust change management process providing a gateway to detect changes that lessen the security posture of the infrastructure, threat modeling becomes the logical mechanism to provide a similar review.

As DevOps blurs the line between software changes and infrastructure changes, there are additional triggers for threat modeling that historically would have been covered by the infrastructure change management process. The two most prevalent additional triggers in DevOps are:

1. Alteration of the software deployment process. Most DevOps environments include tools that automatically build components based on activity in a source code repository, and then deploy those components to production. Compromising any portion of this process can lead to a full compromise of the service, so any change to the authentication/authorization, data flow and elements involved in the deployment automation warrants a threat model review.
2. Changes to the executing environment of the services, including new ports/protocols, alterations to the application server or OS configuration, account and permissions alterations, etc. There are numerous resources describing what events should trigger a review during a change management process, and those triggers map equally well to threat modeling.

16. In Closing

The main purpose of this paper is to demystify the practice of threat modeling. The collective feeling among the authors is that the return on investment from threat modeling justifies the effort necessary to get it started; from there on, it is just about practice.

We encourage you to start now. We would like to receive your feedback on this paper. We are very interested in hearing about your experiences doing threat modeling in your particular industry and environment. Please send your comments to feedback@safecode.org.

17. About SAFECode

The Software Assurance Forum for Excellence in Code (SAFECode) is a non-profit organization exclusively dedicated to increasing trust in information and communications technology products and services through the advancement of effective software assurance methods. SAFECode is a global, industry-led effort to identify and promote best practices for developing and delivering more secure and reliable software, hardware and services. Its charter members include Adobe Systems Incorporated, CA Technologies, Dell EMC, Intel Corp., Microsoft Corp., Siemens AG, and Symantec Corp. Associate members include Autodesk, Boeing, Huawei, NetApp, Security Compass, Synopsys, Veracode, and VMWare. For more information, please visit www.safecode.org.

18. Glossary

Risk – the effect of uncertainty on objectives

SDL – Secure Development Life cycle

SDLC – Software Development Life Cycle

Threat – a potential cause of an unwanted incident that may result in harm to a system or organization

Vulnerability – an instance of a weakness as manifested in a specific implementation

Weakness – a condition that can potentially be exploited by an attacker

19. Acknowledgments

Contributors

Authors (created core content)

- Josh Brown-White, Microsoft
- Loren Brent Cobb, The Boeing Company
- Jim DelGrosso, Synopsys (formerly Cigital)
- Ehsan Foroughi, Security Compass
- Afshar Ganjali, Security Compass
- Souheil Moghnie, Symantec
- Nick Ozmore, Veracode
- Ragavendran Padmanabhan, CA Technologies
- Brook Schoenfield, Intel
- Izar Tarandach, Dell EMC

Technical Reviewer (reviewed, commented on core content)

- Prithvi Bisht, Adobe

SAFECode also acknowledges the efforts of its Technical Director, Tom Brennan, in the development of this paper.

20. References

1. https://en.wikipedia.org/wiki/Attack_tree
2. D. Dhillon, "Developer-Driven Threat Modeling: Lessons Learned in the Trenches," IEEE Security and Privacy, vol. 9, no. 4, 2011: https://www.computer.org/cms/ComputingNow/homepage/2011/0911/W_SP_DeveloperDrivenThreatModeling.pdf
3. https://www.microsoft.com/en-us/download/details.aspx?id=49168
4. https://blog.mozilla.org/security/2015/04/02/introducing-project-seasponge-quick-and-easy-threat-modeling/
5. "Participatory Threat Modeling Class," version 3.3, Brook S.E. Schoenfield, 2016
6. Zachman: https://www.zachman.com/ea-articles-reference/58-conceptual-logical-physical-it-is-simple-by-john-a-zachman
7. https://www.visual-paradigm.com/tutorials/data-flow-diagram-dfd.jsp
8. https://en.wikipedia.org/wiki/Threat_model
9. https://www.microsoft.com/en-us/SDL/adopt/eop.aspx
10. "Securing Systems – Applied Security Architecture and Threat Models" - Brook S.E. Schoenfield – CRC Press, ISBN 978-1-4822-3397-1
11. "6 Myths of Threat Modeling" – Jim DelGrosso and Brook Schoenfield, OWASP AppSec California 2016: https://www.youtube.com/watch?v=yBDNiBE3Xho
12. "The STRIDE Threat Model:" https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx
13. OWASP Top 10: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
14. CVSS: https://www.first.org/cvss/
15. CWSS: https://cwe.mitre.org/cwss/cwss_v1.0.1.html
16. CWRAF: https://cwe.mitre.org/cwraf/
17. CWE: https://cwe.mitre.org/data/slices/2000.html
18. Presentation on rapid threat modeling by Akshay Aggarwal: http://www.blackhat.com/presentations/bh-usa-05/bh-us-05-aggarwal-update.pdf
19. TARA methodology qualifies threat attackers and attack types: Rosenquist, M. (2009). "Prioritizing Information Security Risks with Threat Agent Risk Assessment." IT@Intel White Paper, Intel Information Technology: http://media10.connectedsocialmedia.com/intel/10/5725/Intel_IT_Business_ValueIntel_IT_Business_Value_Prioritizing_Info_Security_Risks_with_TARA.pdf
20. Tony UcedaVelez, Marco M. Morana, Risk Centric Threat Modeling: Process for Attack Simulation and Threat Analysis, John Wiley & Sons, 2015

21. https://en.wikipedia.org/wiki/Sequence_diagram
22. CVE: https://cve.mitre.org/
23. ISO 27005: http://www.27000.org/iso-27005.htm
24. NIST SP 800-30: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-30r1.pdf
25. Threat Modeling: Designing for Security", Adam Shostack, ISBN 978-1-118-80999-0, 2014