National Defense ISAC

Application Threat Modeling

September 5, 2022

Authors:
Raul Barreras
Paul Keim
Waldemar Pabon
Jacob Wilson
Andrew Zuehlke

Reviewers:
Allan Jacob
Jeffrey Fleeks
Will Jimenez
Renee Stegman

About the Authors

Dr. Waldemar Pabon, ND-ISAC Lead & Senior Contributor

Dr. Waldemar Pabon is a Cyber Security Architect with over 27 years of experience in software engineering. Dr. Pabon leads the Application Security Working Group, Software Security Automation, and the COTS Software Assessments Subgroups at the ND-ISAC. Under his leadership, the Software Security Automation Working Group has published multiple white papers. The papers provide ND-ISAC members and the industry with a roadmap on how to adopt application security best practices while leveraging automation as a catalyst to achieve efficiencies. Dr. Pabon has a Doctor of Science in Cybersecurity degree from Capitol Technology University.

Raul Barreras, ND-ISAC Contributor

Raul is an Information Security Professional with more than 20 years of experience. His multiple roles throughout his career include information security officer, systems administrator, developer, teacher, and occasional pen tester. In recent years, Raul has had the opportunity to use the accumulated experience in a new role: application security. This role has allowed him to discover how much he enjoys being a developer advocate and how fun and useful it is to build a community of security champions while improving the quality of the organization's software. This is Raul’s second opportunity to contribute to application security papers at the ND-ISAC.

Paul Keim, ND-ISAC Senior Contributor

Paul Keim is a Senior Security Architect. With ten years of experience in the information security field, with half of that leading the Application Security team, Paul's focus is constantly on how to automate his processes to reduce the workload for his team. A member of ND-ISAC for three years, Paul started in the Application Security subgroup looking to learn more about what other companies are doing to automate their workloads, and now looks to provide his expertise on what's worked and hasn't worked for him over the years.

Jacob Wilson, ND-ISAC Contributor

Jacob Wilson is an Application Cybersecurity Engineer with 5 years of experience in the Information Security field with a primary focus on developing, implementing, and maturing application security programs. Jacob is currently seeking a master’s degree in Computer Science from Georgia Tech and joined ND-ISAC in April 2022 as a member of the application security and red team working groups. Jacob continues to actively participate and collaborate in the ND-ISAC community as well as the broader Information Security community.

Andrew Zuehlke, ND-ISAC Senior Contributor

Andrew Zuehlke is a Cyber Security Architect with more than six years of experience in the information security field. As a Cyber Security Architect, he is primarily supporting Research & Development and Information Technology. Since joining the ND-ISAC, Andrew has become a member of the Cloud Security and Architecture working group as well as the Application Security working group, where he has coauthored multiple white papers. Andrew graduated from Appalachian State University in 2017 with a Bachelor of Science Degree, double majoring in Computer Science and Computational Mathematics.

Executive Summary

Enforcing a "shifting left" approach in the Software Development Lifecycle (SDLC) provides great visibility regarding threat vectors included in the software. Even though these threat vectors are included in the software, if the exploit is weaponized it could transcend from the software to service infrastructure. Despite the great benefits of the SDLC security controls, these controls have no visibility in terms of the other components of the architecture. To fill the gap, we need a systematic methodology to proactively detect threat vectors in the infrastructure as part of software implementation and identify risk mitigation strategies to reduce risk.

Application Threat Modeling provides developer and security teams a systematic process to assess risk as part of an architecture and provides guidance to organizations on the risk mitigation strategies needed. Application Threat Modeling is not a static process. To have comprehensive coverage, it must be conducted as early as possible in the SDLC to ensure risk is identified and mitigated. There are many aspects of software architecture that can change. Data sensitivity, new components, data flow, compensating controls, and many other aspects of architecture can change. Application Threat Modeling assesses the impact of all these changes by detecting potential threats and providing countermeasures to ensure the implementation is secured end to end.

Executing of Application Threat Modeling requires three main phases:

- **Phase #1**: Decomposing the architecture – this step of the process will allow the breakdown of all the components, technical and non-technical, that could be involved in a cyber security event. Software components, protocols, databases, roles, people, etc. are identified and classified in a diagram providing visibility on how they interact with each other.
- **Phase #2**: Detection and Rank of Threats – once all components of the architecture are identified, threats are detected. Since organizations do not have unlimited resources, a ranking process is executed to help assign prioritization in the resolution.
- **Phase #3**: Countermeasures and Mitigation Strategies – during this step, risk mitigation strategies are provided for each threat identified. This part of the process will alert in terms of the effort needed to secure the implementation.

The execution of these steps can be conducted manually. Manual execution will not scale well in large environments with a significant number of applications and multiple changes to the architecture. To fully enable efficiencies in the organization, organizations should consider the use of tools allowing the Application Threat Modeling methodology to be embedded in the development and operations (DevOps) pipeline. This approach provides a consistent automated evaluation of the architecture during every build cycle or sprint. Injecting the results of the architecture evaluation into development security operations

(DevSecOps) should help not only to detect potential threats but also plan for countermeasures.

As noted, the Application Threat Modeling methodology is not a replacement for the security controls in the SDLC. Even if organizations use the methodology, they will not be able to detect threat vectors in code. Therefore, Application Threat Modeling should be used in conjunction with SDLC security controls to fully enforce secure implementations. This way, the organization will have better visibility of the risks associated with the software and every other component of the architecture. This combination provides the most comprehensive approach toward the adoption and enforcement of the shifting-left approach.

Table of Content

Introduction

Objective

Early detection in application security involves not only the identification of potential threats but also risk mitigation strategies to help minimize risk in implementations. Traditional approaches include the use of several security controls in the Software Development Lifecycle (SDLC). Whether identifying threat vectors in the code developers create, risk in the dependencies used in the software, or any malicious behaviors associated with the interaction of components, security controls can prescriptively identify risk in software (for more information regarding security controls in the SDLC, please review the "Software Security Automation: A Roadmap Towards Efficiency and Security" paper). Despite the multiple benefits provided by these tools, their prescriptive nature will only focus on the code being created. This will leave out of scope all the different artifacts interacting with the software:

- What is the data flow?
- What type of users will be accessing the features?
- Infrastructure components used for the implementation
- Other concerns

The lack of visibility by these security controls regarding the rest of the components (technical and non-technical) in the architecture creates a gap in the risk analysis. To fill the gap, Application Threat Modeling provides a structured and systematic methodology to enable organizations to analyze the architecture, understand the flow of information, detect threats, and identify risk mitigation strategies using a proactive approach. This process is executed as early as possible in the implementation but is recommended to be conducted during the design stage. Performing it during the design stage of the project enables proactive visibility of the potential threats impacting the implementation before a single line of code is written. Application Threat Modeling is not a static process and will require multiple iterations as the architecture and its components change. The process is aimed at answering the following key questions in the architecture (Revuelta, 2020):

- What are we building?
- What can go wrong?
- What are we going to do about it?

This white paper's objective is to provide a high-level understanding of the importance of the Application Threat Modeling methodology and to explain the three steps involved in Application Threat Modeling. The paper also covers how to embed Application Threat Modeling in DevSecOps to create efficiencies using automation and covers an overview of the type of tools available in the industry to support the process as part of stand-alone usage or cloud native.

Audience

The audience for this white paper includes software and security engineers, and software and security architects responsible for the secure design of software implementations in the organization.

Structure of the paper

This paper introduces the Application Threat Modeling methodology and a high-level overview of the three (3) phases in the methodology. Leveraging automation and the creation of efficiencies are an important aspect of any application security strategy. The paper provides a high-level overview on how to integrate the methodology in a DevSecOps pipeline, taking advantage of existing technology capabilities to not only create a map of the architecture but also integrate a feedback loop to help with planning remediation. Finally, a discussion regarding the types of tools available in industry to conduct Application Threat Modeling are introduced.

Application Threat Modeling

Application threat modeling is the process used to identify threats and ways of mitigating risk. This process occurs in three (3) main phases, each of which should be thoroughly documented. Decomposition is the first phase and involves analyzing the application architecture, generally within the context of use cases, to understand all components, sub-components, data entry and exit points, external dependencies, and assets. It should also include evaluating trust levels, interactions with each other, and interactions with the user. Once a clear picture of the application architecture and its functionality have been generated, the second phase begins, which includes identifying and categorizing threat vectors relevant to the architecture.

After this has been done it is important to rank said threat vectors to help prioritize time and effort. The third phase begins after all major threat vectors have been identified and ranked. This phase includes defining countermeasures and creating an overall risk mitigation strategy. This can include everything from redesigning the architecture to implementing mitigating controls and should include both short-term and long-term goals. To be effective, this process should be collaborative and include developers, data engineers, security engineers, operations, and anyone else involved in the development or deployment of the application. The application threat model should also be iteratively reassessed as requirements change through the software development life cycle so that it can provide a continuously accurate and solid foundation for understanding risk.

Decompose the Application

When beginning the threat modeling process of an application, it is beneficial to first decompose the application. During this procedure, an application is broken down into its parts to help visualize and identify the pieces that together form the application and how each piece interacts with other pieces, whether internal or external. This procedure can be compared to the information gathering and documentation phase of a project. Ultimately, the process of decomposing the application into its parts allows for a better and easier understanding of the application as a whole.

Available on the Open Web Application Security Project (OWASP) Foundation’s website, Conklin (n.d.) paper, “Threat Modeling Process,” introduces the idea of breaking the decomposition process into six sections. These six sections are:

1. Threat Model Information
2. External Dependencies
3. Trust Levels
4. Entry Points
5. Exit Points
6. Assets

Information gathered during these sections can then be utilized and incorporated into a threat model document; this information can also be used to create data flow diagrams for the application. The following will further investigate each of these sections. After each section, an abbreviated example is presented in a table format to help illustrate the concept.

Threat Model Information

This section can be thought of as an overall summary of the application and any applicable information, including a description of how the application will be used. Having this information documented helps highlight important context that can assist with identifying potential threats. A detailed threat model document should contain the following:

- **Application Name:** The name of the application being threat modeled.
- **Application Version:** The version of the application being threat modeled.
- **Description:** A detailed description of the application being threat modeled.
- **Document Owner:** The owner or author of the threat model.
- **Participants:** The other authors who contributed to developing the threat model.
- **Reviewer**: The reviewer(s) of the threat model.

Table 1 provides an example of how to keep track of the threat model information**.**

Table 1

Threat Model Information

Application Name:	Survey Tool
Application Version:	1.4.2
Description:	The "Survey Tool" application will be used to create surveys, send surveys, and record/save survey responses. Survey responses can be analyzed for trends or specific comments. The users of the application can be split into two groups: 1. Survey Administrators 2. Survey Recipients Survey Administrators will create and send surveys to survey recipients. After Survey Recipients submit their responses, Survey Administrators can review existing surveys and submitted responses.
Document Owner:	Andrew Zuehlke
Participants:	Paul Keim
Reviewer:	Waldemar Pabon

External Dependencies

The external dependency section identifies items external to the code of the modeled application and have the potential to pose a threat to the application. An important component to consider when identifying external dependencies is the environment in which an application will be run, as the environment may impose additional dependencies such as compliance requirements. Each external dependency should be documented with the following:

- **ID:** A unique identifier associated with each external dependency.
- **Description:** A detailed description of the external dependency.

Table 2 provides an example of how to keep track of the external dependency information**.**

Table 2

External Dependencies Information

ID	Description
1	The Survey Tool will run in a government cloud tenant and must meet or exceed all requirements, including hardened operating systems and database configurations.
2	Communication between the users and the user-facing web application, as well as the user-facing web application and the back-end database, will be encrypted with TLS 1.2.
3	Due to the potential for personal information to be collected in surveys, privacy laws and regulations must be considered where applicable.
4	The database where data is stored will be behind a firewall and only internally accessible.

Trust Levels

In order to best identify what rights or permissions will be granted to external entities, it is helpful to identify trust levels. Each entry and exit point—described in more detail below—

should be cross-referenced with a trust level. Each trust level should have the following documented:

- **ID:** A unique identifier associated with each trust level.
- **Name:** An easy-to-understand name that represents the trust level and what access it grants.
- **Description:** A more detailed description of the trust level, including any external entity who is assigned the trust level.

Table 3 provides an example of how to keep track of the trust levels information**.**

Table 3

Trust Levels Information

ID	Name	Description
1	Anonymous Web User	A generic, unauthenticated user who accesses the Survey Tool but does not provide credentials.
2	Survey Recipient with Valid Invite URL	A user who has received an individual invite URL to participate in a survey.
3	Survey Recipient with Invalid URL	A user who attempts to open a URL that was sent to another individual.
4	Survey Administrator with Valid Credentials	A user of the Survey Tool that has logged in successfully with administrator credentials.
5	Survey Administrator with Invalid Credentials	A user of the Survey Tool that attempted to log in with invalid administrator credentials.

6	Database Read Service Account	A service account that is used to access and read data from the database to display for a Survey Administrator.
7	Database Read/Write Service Account	A service account that is used to access the database and write data after a survey is submitted.

Entry Points

Arguably one of the more critical aspects of an application threat model is identifying vulnerabilities—intentional or not—that could allow an attacker to exploit and manipulate the application. These vulnerabilities can be associated with an entry or exit point. One of the more common entry points to identify is an input field where an end-user has the opportunity to input customized text; an attacker may be able to use an input field to perform code injection attacks. Because applications often have multiple pages, each with multiple inputs, an entry point may be layered. Essential details to document for each entry point are:

- **ID:** A unique identifier associated with each identified entry point. This ID can be referenced with threats or vulnerabilities that are identified in later stages.
- **Name:** An easy-to-understand name that describes the entry point.
- **Description:** A more detailed description of the entry point, including any activity that may occur at the entry point.

- **Trust Levels:** The level of access required at the entry point. This trust level should align with the trust levels previously defined in the 'Trust Levels' section above.

Table 4 provides an example of how to keep track of the entry points information**.**

Table 4

Entry Points Information

ID	Name	Description	Trust Levels
1	HTTPS Port	The Survey Tool is only accessible via TLS; therefore, all web pages within the Survey Tool are subsequently layered under this entry point.	(1) Anonymous Web User (2) Survey Recipient with Valid Invite URL (3) Survey Recipient with Invalid URL (4) Survey Administrator with Valid Credentials (5) Survey Administrator with Invalid Credentials
1.1	Survey Tool's Admin Login Page	All administrators must first log into the Survey Tool administration page before creating new surveys or reviewing existing survey results.	(1) Anonymous Web User (2) Survey Recipient with Valid Invite URL (3) Survey Recipient with Invalid URL (4) Survey Administrator with Valid Credentials (5) Survey Administrator with Invalid Credentials
1.1.1	Survey Tool's Admin	The admin login function accepts credentials provided by an	(4) Survey Administrator with Valid Credentials (5) Survey Administrator with Invalid Credentials

	Login Function	end-user and compares them with administrator credentials.	
1.2	Survey Tool's User Interaction Page	All survey recipients must first log into the Survey Tool via a specific URL that was sent to the recipient's email address.	(1) Anonymous Web User (2) Survey Recipient with Valid Invite URL (3) Survey Recipient with Invalid URL (4) Survey Administrator with Valid Credentials (5) Survey Administrator with Invalid Credentials
1.2.1	Survey Tool's User Interaction Functions	The user interaction function accepts credentials provided by a unique URL with a valid auth token.	(2) Survey Recipient with Valid Invite URL (3) Survey Recipient with Invalid URL

Exit Points

Like entry points, exit points highlight potential vulnerabilities by identifying where data may leave the system. While an entry point may be a text input field, a common exit point example could be dynamic output. As exit points can provide a method for data extraction, each exit point should have appropriate security controls in place to protect the confidentiality, integrity, and availability of the data. The same fields documented for entry points should also be documented for exit points.

- **ID:** A unique identifier associated with each identified exit point. This ID can be referenced with threats or vulnerabilities that are identified in later stages.
- **Name:** An easy-to-understand name that describes the exit point.
- **Description:** A more detailed description of the exit point, including any activity that may occur at the exit point.
- **Trust Levels:** The level of access required at the exit point. This trust level should align with the trust levels previously defined in the 'Trust Levels' section above.

Table 5 provides an example of how to keep track of the exit points information**.**

Table 5

Exit Points Information

ID	Name	Description	Trust Levels
1	HTTPS Port	The Survey Tool is only accessible via TLS; therefore, all web pages within the Survey Tool are subsequently layered under this entry point.	(1) Anonymous Web User (2) Survey Recipient with Valid Invite URL (3) Survey Recipient with Invalid URL (4) Survey Administrator with Valid Credentials (5) Survey Administrator with Invalid Credentials

1.1	Survey Tool's Admin Review Page	Authenticated administrators can review existing surveys and responses.	(4) Survey Administrator with Valid Credentials
1.1.1	Survey Tool's Admin Review Function	The admin review function returns surveys and survey responses from the database.	(4) Survey Administrator with Valid Credentials (6) Database Read Service Account
1.2	Survey Tool's User Interaction Page	Authenticated survey recipients may receive dynamic survey questions that change based on survey input.	(2) Survey Recipient with Valid Invite URL
1.2.1	Survey Tool's User Interaction Functions	As authenticated survey recipients enter responses to survey questions, the responses are written to the database, and the next question is displayed as applicable.	(2) Survey Recipient with Valid Invite URL (7) Database Read/Write Service Account

Assets

Assets are another critical component of the application threat model. Assets identify what an attacker may be interested in—and therefore what an attacker's target(s) may be. The same details should be documented for each asset as were documented for entry and exit points.

- **ID:** A unique identifier associated with each identified asset. This ID can be referenced with threats or vulnerabilities that are identified in later stages.

- **Name:** An easy-to-understand name that describes the asset.
- **Description:** A more detailed description of the asset, including any activity that may occur at the asset.
- **Trust Levels:** The level of access required at the asset. This trust level should align with the trust levels previously defined in the 'Trust Levels' section above.

Table 6 provides an example of how to keep track of the assets associated with the application**.**

Table 6

Assets Information

ID	Name	Description	Trust Levels
1	Administrators and Survey Recipients	Assets relating to the administrators or recipients of surveys	(see breakdown below)
1.1	Administrator Login Details	The login credentials used by administrators to log into the Survey Tool Admin Review Page	(4) Survey Administrator with Valid Credentials
1.2	Survey Recipient Login Details	The login credentials (including URLs and tokens) are used by survey recipients to access surveys.	(2) Survey Recipient with Valid Invite URL
1.3	Database Service Account Details	The authentication credentials for the service accounts that are used to interact with the underlying database.	(6) Database Read Service Account (7) Database Read/Write Service Account

1.4	Personal Data	Some surveys may collect the personal data of survey recipients.	(2) Survey Recipient with Valid Invite URL

As illustrated by Table 6, capturing, and presenting assets will require a breakdown of components starting with a high-level categorization. For each category, the listing of assets needs to account for all the different components related to the asset. When continuing to document additional assets with their corresponding trust level, the example below provides an idea on how to continue to do so.

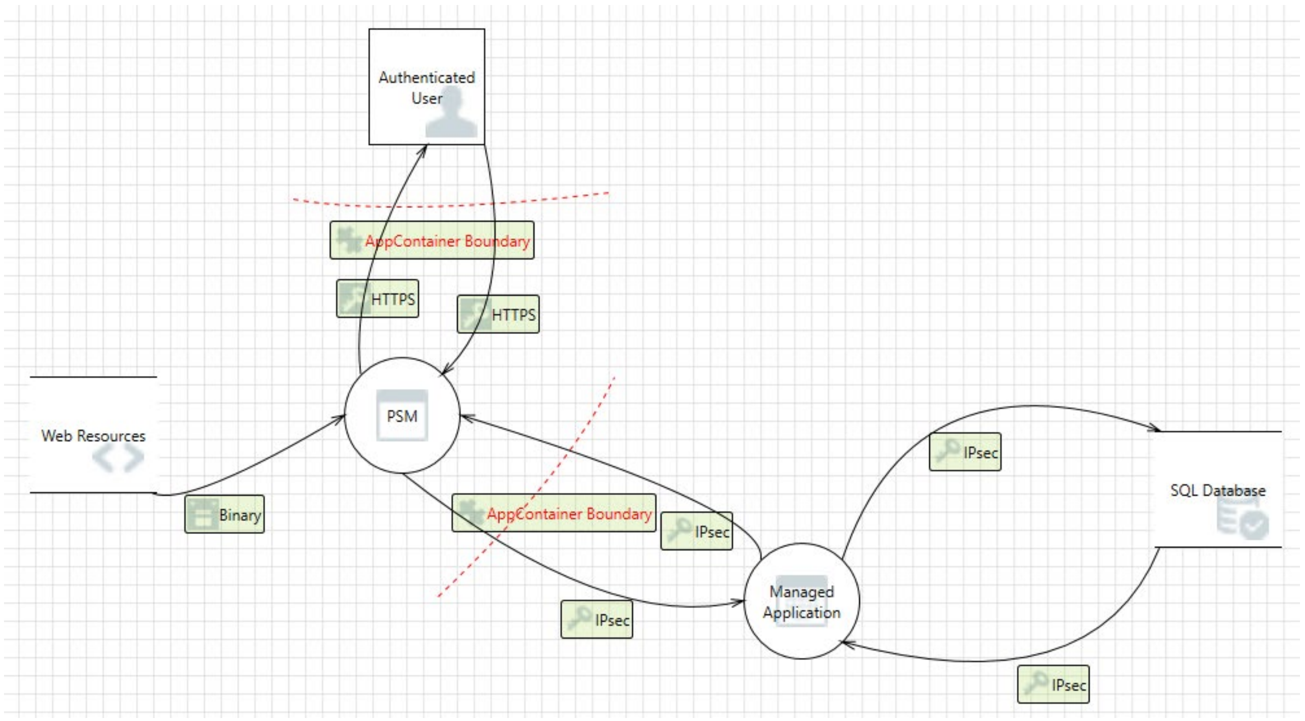
<u>2</u>	System	Assets relating to the underlying system	(see breakdown below)
2.1	…	…	…

Data Flow Diagrams

Using data collected during the decomposition of an application, a Data Flow Diagram, can be created to provide a visual representation of how the application processes data. A Data Flow Diagram helps illustrate how data logically flows through application end-to-end and allows easier identification of affected components through critical points. Due to the complexity of applications, it is possible—and likely—to have multiple layers of data flow diagrams to represent the application at various levels; for example, an application may have a high-level data flow diagram that conveys the overall scope of an application while lower-level diagrams might focus on specific processes and their associated details. Figure 1 provides an example of a Data Flow Diagram.

Figure 1

Data Flow Diagram Example



Note: Data Flow Diagram showing assets, entry and exit points along with trusted boundaries and types of users accessing the application.

Determine and Rank Threats

Armed with the information gathered from our application decomposition, we are now ready to begin the core component of the threat modeling process – identifying the threats faced by our application. There are many ways to approach this problem, but it is recommended to use a dedicated framework to guide thinking in addition to accessing organizational knowledge of historical threats and attacks against the business.

A non-comprehensive list of threat modeling and risk assessment frameworks are listed below with key purposes and implementation considerations:

STRIDE

Standing for "Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, and Elevation of Privilege," STRIDE is a methodology that is targeted toward identifying what could go wrong in each of the different components of the decomposed application (Donovan, 2021). STRIDE is useful for serving as a launch point for considering what problems might arise from a system but does not in and of itself provide any framework around the further analysis of the threats or how to categorize the risk level of each of the threats.

DREAD

Standing for "Damage, Reproducibility, Exploitability, Affected users, Discoverability," DREAD is a risk-assessment framework that looks at attempts to assign a score from 1 to 10 for each of the component categories. Identified threats are filtered through the framework and given a score that then can be used to rank and prioritize (EC-Council, n.d.). If choosing to implement DREAD or a similar framework, consider how to reduce as much subjectivity as possible for the score assignment so that disparate stakeholders can still arrive at similar agreements on the scores.

PASTA

Standing for “Process of Attack Simulation and Threat Analysis,” PASTA is a combined threat modeling and risk assessment framework (Nick, 2022). It uses a multi-staged process to complete the overall threat modeling process, with application decomposition being stage 3 and stages 4 through 7 being threat identification and analysis. Key implementation considerations here are the somewhat complicated nature of the PASTA process, and the associated overhead with each threat model associated.

TARA

Standing for “Threat Analysis and Risk Assessment,” TARA is a threat-focused methodology focusing on threat identification and impact analysis (Gossett, 2021). It uses predetermined documentation to encourage consistency between threat modeling agents and environments. The TARA method focuses on asset identification and priorities risk assessments based on each asset. Then the threat modeler determines and measures impact scenarios along a rating scale according to each asset.

For a new threat modeling team, the STRIDE methodology is likely to be the methodology of choice due to its ease of execution for identifying threats. Similarly, the Common Vulnerability Scoring System (CVSS) or DREAD framework can be used to quickly generate prioritization guidelines. At the end of this step, the threat modeling team will have a list of identified threats to the applications and their relative risk scores. There are different sites which provide different templates for public consumption to help capture the relevant information associated with a threat (Conklin, n.d.).

A cautionary note about this step, – no methodology is perfect, and one cannot guarantee that all possible threats have been identified by any given threat model, either from unknown information or simple slips of the mind by the modeling team. To assist in the identification of threats, the threat modeling team should document what was identified and how, as well as what threats were identified by outside partners (SOC, Legal, Fraud, etc.) to better grow the team's knowledge and capabilities. Centralizing the data would require the implementation of a metrics program to help with visualization, guide remediation efforts, leverage the information to create efficiencies and strategies to minimize the associated risks. The ND-ISAC white paper Software Security Controls: Metrics Automation (Barreras et al., 2021) can be used as a starting point. The paper provides great information on how to implement a metrics system and identifies key metrics to consider.

Table 7 provides an example of an executive summary template that can be used to establish the ranking of the threats.

Table 7

Executive Summary Template

ID	Risk Score	Threat Title	Short Description
<u>**1**</u>	<u>**(High/Medium/Low)**</u> <u>**(Numerical Value)**</u>	Assets relating to the administrators or recipients of surveys	1-2 Sentences conveying the business impact of risk
<u>**2**</u>	<u>**High**</u> <u>**(10.0)**</u>	Loss of Confidential information due to Unauthenticated APIs	High risk of financial and reputational damage from unauthorized access to customer data.

Table 8 provides an example of a template to help with the ranking of the threats which will support the prioritization of the efforts needed to mitigate/remediate the threats.

Table 8

Detailed Record Template

<Risk Ranking>	Threat Title				
Risk Ranking	**Category 1**	**Category 2**	**Category 3**	**Category 4**	**Total**
Score	Value 1	Value 2	Value 3	Value 4	Sum
Threat Scenario	High-level summary of the threat, used for easy digestion				
Associated Threat Actors	What threat actors were identified as likely to target this risk				
Associated Components	What components from the decomposition are involved in this threat				

Impacted Business Flows	What business flows and capabilities are impacted if this threat were to happen
Identified Gaps	List known gaps with company policy and provided business or security requirements.
Notes	Any additional notes or comments for the team to use as future reference or collaboration

Countermeasures and Mitigation Strategies

The final step of the application threat modeling process is answering the question, *How are the identified threats mitigated in our environment?* As inputs to this question, the threat modeler uses the list of threats generated by the prior step, as well as the understanding of the business's environment and its security features. Regardless of whether we use a manual process or automation for the execution of the Application Threat Modeling, the outcome of both strategies will produce a list of mitigation strategies we can implement to lower the associated risk. The list of threat vectors/scenarios in the previous phase is used by this phase of the process to map the risk mitigation strategies needed which will guide the team toward a more secured implementation. Elements such as risk tolerance/acceptance are an important consideration when deciding on the risk mitigation strategies. Organizations may find themselves in situations where the risk of a specific threat vector would be accepted (because of additional considerations such as there are other compensating controls, impact is low, cost implications, etc.).

To help with the determination of the risk mitigation strategies and risk tolerance/acceptance, several questions must be explored. Starting with the highest risk threat vectors identified, we can iterate and ask the following questions:

- Is this risk mitigated by any of the controls in the environment?
- If so, what is the adjusted risk level?
- Is the adjusted risk level an acceptable level of risk to the business?
- Are there any additional controls that could be added to further mitigate the risk?

As a brief example, let’s review the potential risk to an API endpoint in the environment. In the theoretical example, the endpoint has access to confidential information, is unauthenticated, and is open to the internet for access. While such a design decision may cause the threat modeler to find a new place to work, the threat modeler can also consider the design of the application and how to better protect it.

- Is the risk mitigated by any of the controls in the environment? Currently, no additional controls are in place that mitigates this risk.
- What is the adjusted risk level? Still high.
- Is the adjusted risk level acceptable? No.
- Are there any additional controls that could be added to further mitigate the risk?

Here we step back and consider the possibilities. The threat modeler can start with some of the most obvious recommendations – putting authentication on the endpoint and removing the sensitive data from the response unless it is needed for operation. But the threat modeler can also investigate the other controls in the environment - the endpoint may have a network control (firewall, Web Application Firewall, or gateway) that may allow the implementation of an Access Control List (ACL) to limit access. Should those controls fail, the use a host-based firewall is recommended.

At the end of the fourth question, the adjusted risk level and any recommended controls can be provided to the project teams and management, as well as the final adjusted risk level after all proposed mitigations are in place. At this point, the business will need to decide what to do with the risk. The final disposition is typically broken down into three categories:

- Risk Accepted – The risk presented by the threat is within risk tolerance/acceptance levels that the business accepts the risk as reasonable operating risk
- Risk Mitigated – The risk presented by this threat is not above risk tolerance/acceptance levels, but additional controls (those output in question 4) can be added to lower the adjusted risk to where it can be accepted
- Risk Not Accepted – The risk presented by this threat is not able to be satisfactorily mitigated

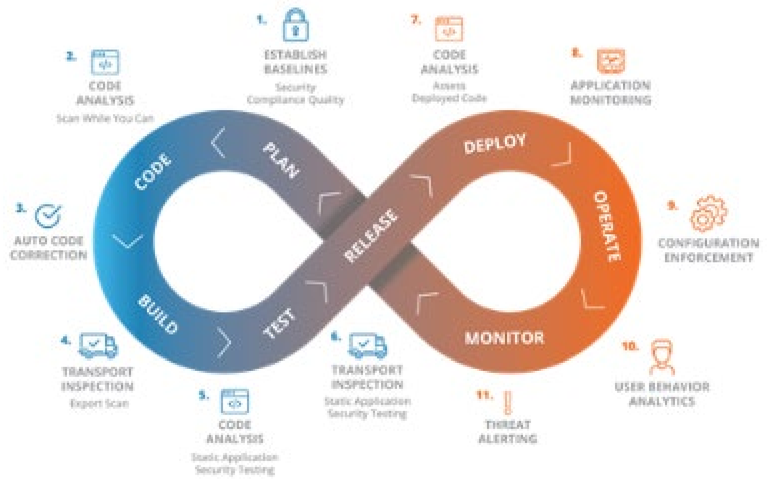
The final action on the risk should be taken by the appropriate party, though the threat modeling team should still be kept in the loop to evaluate changes and provide input on other points of view.

DevSecOps Automation

DevSecOps stands for development, security, and operations (Red Hat 2018), and is a set of practices that combines software development and security, to increase efficiency and improve security in the software development lifecycle. DevSecOps is not a single tool, process, or methodology. Rather, it is an approach to development that incorporates security into every stage of the development process. Figure 2 provides a general overview of the DevSecOps process and its different stages.

Figure 2

An example of a DevSecOps iteration and all its stages.



Note: From *DevSecOps Cycle* [Image], by Onapsis, 2020, (https://onapsis.com/sites/default/files/infographic.png)

The first step in DevSecOps is planning. This includes identifying what security risks are associated with the application and architecture and determining how to mitigate those risks. Security risks associated with applications and architecture can be identified by performing a threat modeling exercise to determine potential threats and attacks. By identifying and addressing potential security threats early in the development process, developers can help to ensure that software is more secure and less likely to be exploited by attackers, saving time and money.

Whether developers are decomposing the application, detecting threats, or identifying risk mitigation strategies, the implementation of Application Threat Modeling in DevSecOps automation will ensure better visibility of threats in a proactive fashion. This proactive approach is a key component in early detection initiatives aimed at implementing a shift-left approach to minimize risk in the implementation. Just like in the SDLC, Application Threat Modeling can also benefit from efficiencies, standardization, and enhanced visibility provided by automation in the delivery pipeline.

How to integrate Threat Modeling in DevSecOps

The use of classical methods (whiteboards and other drawing tools) to make threat models is cumbersome, slow, and does not fit an agile development model. Applications will change frequently, making it difficult to keep track of those changes and consider all the implications these changes will have on the final product. For this reason, developers need tools that allow them to keep threat models as close as possible to the product. Developers should use a language that all can speak, so the models are kept updated as the software is developed, or new threats are discovered. Another problem with traditional methodologies is the inability to find implementation-based flaws because the process of determining threats happens long before features are implemented.

DevSecOps provides a unique opportunity to enable efficiencies in the SDLC, and Application Threat Modeling is no exception. Leveraging automation minimizes human friction and enables faster threat detection and resource allocation for remediation. To understand the compelling benefits associated with DevSecOps and Application Threat Modeling, the use of cloud environments would help rationalize the value behind the enforcement of automation in the process. Even though some Application Threat Modeling tools can provide a semi-automated process to evaluate changes in architectures, the configuration-based nature of cloud environments provides the best scenario for full automation and efficiencies.

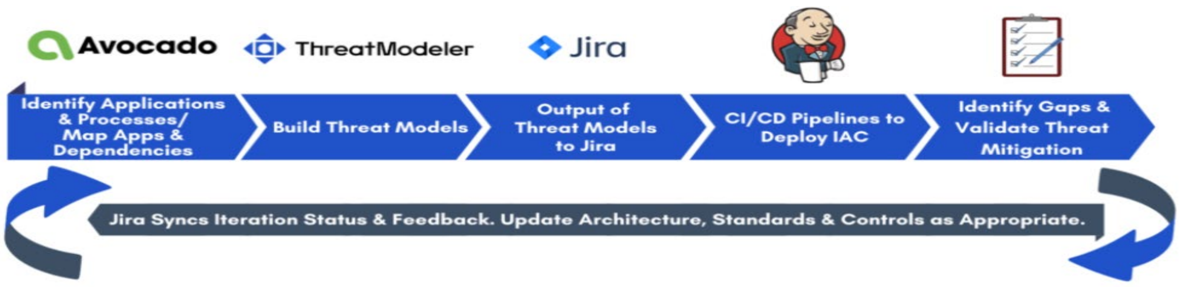
When resources are created in cloud environments, these resources maintain a JSON configuration file documenting all the settings used. This format can then be consumed by Application Threat Modeling tools to identify and decompose an architecture automatically. The consumption of cloud configuration information in tandem with the automation capabilities enable the methodology tools to identify all resources created in a cloud account and facilitates the creation of data flow diagrams based on this information without any kind of human intervention. As discussed in the section "Decompose the application," this is the first step of the methodology which will provide all the important information needed for the detection of threats and countermeasures.

Once the data flow diagram is generated by these tools from the JSON configuration files, the rest of the methodology can execute without any type of human friction. The next step in the pipeline can use the automatically generated data flow diagram and identify threat vectors and risk mitigation strategies. This capability provides a good opportunity to use the information generated by the tools in the DevOps pipeline and inform the development/implementation team of the newly discovered risks. Integration with an issue tracker is possible, providing a constant flow of risk information to the team which will help with the planning of risk mitigation in each subsequent build cycle or sprint.

These efficiencies enable teams to plan and allocate resources to tackle the most pressing issues in an automated fashion. Achieving highly automated threat modeling requires the combination of multiple tools. Considerations for the automation must consider the Continuous Integration (CI) platform, issue tracker, risk gates, and the threat modeling tool itself as minimum requirements. Figure 3 provides an example of the integration between an Application Threat Modeling tool and the DevOps pipeline in support of DevSecOps.

Figure 3

Representation of the integration of an Application Threat Modeling tool in the DevOps pipeline.


Avocado
ThreatModeler
Jira
Identify Applications & Processes/ Map Apps & Dependencies
Build Threat Models
Output of Threat Models to Jira
CI/CD Pipelines to Deploy IAC
Identify Gaps & Validate Threat Mitigation
Jira Syncs Iteration Status & Feedback. Update Architecture, Standards & Controls as Appropriate.

Note: From *HOW TO AUTOMATE THREAT MODELING USING THREATMODELER AND AVOCADO* [Image], by ThreatModeler, 2022, (https://threatmodeler.com/automate-threat-modeling-using-threatmodeler-avocado/)

From an agile methodology perspective, it makes sense to implement these tools in the pipelines to automatically execute the threat modeling as part of the early stages of the build cycle or even in isolation to proactively detect changes in resources that may have introduced risks to the implementation. Having the ability to include risk gates is recommended as this feature will allow teams to stop the deployment of artifacts in a production environment if critical/high-risk threats are identified. In CI platforms, plugins are created to support these types of functions.

The previous discussion helps with the understanding of how fully automated implementations can embed Application Threat Modeling in a DevSecOps strategy. Using out-of-the-box plugins in the CI will help minimize the need to maintain implementations using code. This out-of-the-box approach enables the implementation of great efficiencies. Despite these benefits, there are instances where code may be needed to provide more flexibility in the DevOps pipeline. To support these cases, there is also the availability of using threat modeling tools capable of allowing the use of code to manage the methodology process. Tarandach and Coles (2020) defined two main approaches to this task:

- Threat modeling from code
- Threat modeling with code

To review information regarding the option of using code as part of the DevOps pipeline through code, please see the different tools presented in Appendix A and Appendix B.

Importance of Feedback Loop

The output of a threat model consists of Flow Diagrams, Sequence Diagrams, a list of threats, and a list of vulnerabilities. These lists are essential in the selection of controls that should be used in applications, to prevent or mitigate the effect of a successful exploit. Unmitigated threats or vulnerabilities can be used to create issues in the development team's bug-tracking system.

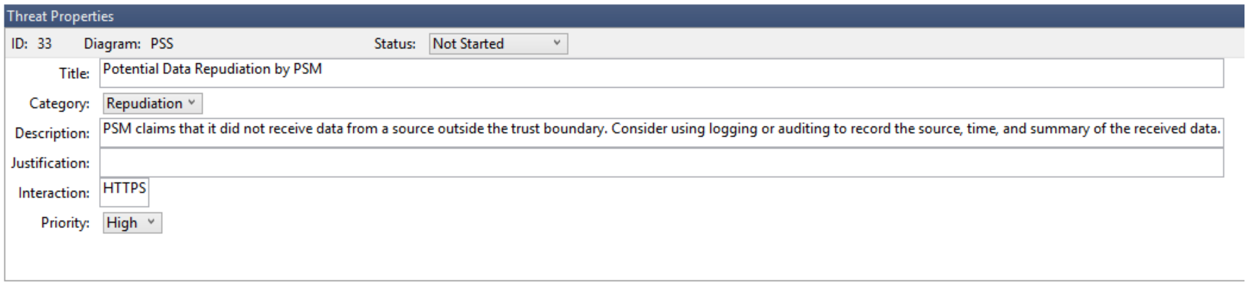
There are several approaches to integrating threat modeling tools in the CI/CD.

Including findings detected by the methodology should be considered as an important step to improve the security posture of an implementation. One mechanism often used to create efficiencies is the integration of the Application Threat Modeling into an issue tracker. This efficiency becomes more actionable when the teams use tools to help automate the methodology. For example, in a scenario where the Application Threat Modeling gets embedded as part of a DevOps pipeline, the outcome of the analysis during the build cycle has the potential to directly connect to the issue tracker used by the organization and create an issue with all the related details.

This type of integration enables teams to plan as part of their build cycles which security countermeasure will be included throughout the lifecycle of the application and at which stage. It becomes important to take into consideration the ranking mechanisms provided by the Application Threat Modeling tools to bring to the attention of the teams those pressing issues which may increase the risk exposure. Leveraging a ranking for those findings in the issue tracker will enable teams to select the highest severity issues. Figure 4 shows how Application Threat Modeling tools provide a field for priority to help support better resource allocation during remediation.

Figure 4

Application Threat Modeling Tool enables the ranking of the identified threats through the priority field.


Threat Properties

ID: 33 Diagram: PSS Status: Not Started

Title: Potential Data Repudiation by PSM

Category: Repudiation

Description: PSM claims that it did not receive data from a source outside the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.

Justification:

Interaction: HTTPS

Priority: High

There are other tools where integration is also possible to support a feedback loop. Pre-commit git-hooks, GitHub Workflows or GitLab Pipelines can be used to generate a threat model, and as mentioned before, issues that should be addressed by developers and other stakeholders. Organizations need to ensure findings are actionable, and there are steps to implement remediation. In Appendix A, the Autodesk Continuous Threat Modeling methodology is presented as an option to use code and integrate the threat modeling into the DevOps cycle.

Tools to facilitate Threat Modeling

Executing an Application Threat Modeling exercise manually is not an efficient approach. However, before you can automate there are parts of the process that will require human interaction and cannot be automated. For example, defining and discussing architecture components, data flow, and interactions. Having a conversation regarding those elements will require interaction from multiple teams (database administrators, developers, architects, etc.). Nevertheless, once there is a clear understanding of all the architectural pieces associated with an implementation, using a tool to automate some of the processes is beneficial and will lead to efficiencies in threat detection and countermeasures. Instead of creating an artifact manually documenting data flows, threat vectors, and risk mitigation strategies and mapping them, multiple solutions in the industry will help with phase 2 Determine and Rank Threats as well as phase 3 Countermeasures and Mitigation Strategies.

The main objective of the tools is to allow any practitioner of the Application Threat modeling to define the architecture components in the tool and then enable it to produce results to help with the detection of threats as well as the risk mitigation strategies. This section will cover two important aspects of the Application Threat Modeling tools. First, a discussion of cloud-native tools will help understand what types of tools are available in cloud-native environments. Finally, a section is included to discuss how stand-alone tools can support basic Application Threat modeling exercises along with the integration of the tool in the DevOps pipeline to enable efficiencies through automation.

Cloud Native Tools

Understanding the level of responsibility in cloud deployments is essential before discussing the cloud tools available to conduct Application Threat Modeling and how to map the different steps in the process. Validating risk in cloud environments is no different than evaluating an application on-premises. Security vulnerabilities exist in cloud applications in the same manner as on-prem deployments. The fundamental difference between on-prem and cloud implementations is associated with the level of responsibility of the components involved in the architecture. When conducting application threat modeling against cloud environments, the shared responsibility model will dictate the components an organization is responsible for. Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), all have different levels of responsibility.

For instance, resources such as a server in a SaaS deployment are outside the reach of the organization because cloud providers control the back-end components. Nevertheless, the data residing in those servers is a key responsibility for any organization deploying applications in cloud environments. This scenario exemplifies how the level of responsibility will shift the type of risk mitigation strategies and will define the responsible party. This reality exposes the need for organizations adopting cloud services to perform application threat modeling as part of the implementation and are cognizant of the shared responsibility model to determine which risk mitigation strategies are the responsibility of the organization and which are not.

Recognizing these different levels of responsibilities will help understand how cloud tools can help support the three phases of Application Threat Modeling. When dealing with the decomposition of the application as well as threat detection, there are some solutions available in the cloud providers to help. Cloud providers provide a marketplace feature where several solutions exist to support the execution of the Application Threat Modeling. Some of the core features of these solutions include the creation of an architecture based on the resources deployed which is a key component of decomposing the application. Also, these solutions have a detection capability that uses the cloud architecture identified and mapped against the cloud environment.

When evaluating the offerings to support the identification of risk mitigation strategies and countermeasures, these solutions can help guide the organization through the necessary steps to mitigate any cloud-related risk. Some of these mitigation strategies will include cloud services already available in the cloud provider. For example, a risk mitigation strategy identified against a security misconfiguration may require adjusting security groups which are inherent components of the cloud environment. Other types of countermeasures may require the implementation of integration with other solutions based on the strategy used by the organization. For example, the lack of logging in a cloud account could be resolved by utilizing native cloud capabilities but it can also be tackled through the routing of all logging information to a centralized SIEM used by the organization. This option may prove to be more cost-effective and efficient in the long run. Therefore, it becomes important to integrate application Threat Modeling solutions that not only will take advantage of cloud-native solutions but also provide integration capabilities as a countermeasure.

From an automation standpoint, there are solutions embedded in cloud environments providing integration capabilities to evaluate the cloud resources as part of a DevOps pipeline and create issues in the issue tracker software regarding the threats identified. These tools can read the configuration from the cloud environment and automatically create a data flow diagram showing all the artifacts and the data flow associated with the implementation. This type of integration enables teams to have a constant feedback loop across each build cycle. Any change applied to the cloud environment architecture (new resources, changes in security groups or network ACLs, etc.) can be evaluated by these tools as part of the DevOps pipeline. This capability enables teams to not only capture from the pipeline the outstanding security work that needs to be done but also provide the ability to stop the automation if a critical security misconfiguration is detected. This level of proactive issue detection positions the organization on a risk mitigation roadmap that will help minimize risk in the implementation.

Implementing automation as part of Application Threat Modeling positions the onboarding of new environments or resources in a cloud environment in a constant security posture evaluation and feedback loop. The constant evaluation that is possible through a DevOps pipeline will enforce a proactive approach toward decomposition, threat detection, and countermeasures. Since Application Threat Modeling is not a static process, this approach serves as a standard mechanism to increase visibility as it relates to risk and is a great asset when minimizing risk in cloud implementations.

Standalone Tools

Performing an Application Threat Modeling exercise requires human collaboration and is an important step to define the components of the architecture. It would be extremely challenging for a security engineer alone to identify all the artifacts or components included in architecture without any kind of collaboration with other team members such as database administrators, server administrators, developers, architects, etc. Despite the reality of how the Application Threat Modeling exercise can be executed, conducting a manual exercise creates a challenge when organizations have multiple projects running at once. The amount of time that will be required to manually document threat detection, as well as the risk mitigation strategies from each project, will create friction when trying to achieve efficiencies in the process. This is one of the reasons why using a tool becomes a key approach. By using a tool, organizations can minimize friction as part of the Application Threat Modeling exercise fostering efficiencies.

Once the architecture is defined conceptually, the use of tools and automation activities associated with Application Threat Modeling can commence. There are two types of applications in the industry supporting Application Threat Modeling: standalone and automation tools. The first type of application is standalone tools. These types of tools are going to help organizations with the definition of their data flow and the basic identification and mapping of threats and the corresponding countermeasures and risk mitigation strategies. These capabilities build the foundation of an Application Threat Modeling exercise, but they do not accomplish the task of creating efficiencies in the process and enforcing automation, especially in the DevOps pipeline.

The standalone tools are a good step for those organizations new to Application Threat Modeling. It enables them to create a data flow diagram and allow the rest of the process to take place automatically (mapping of threats and risk mitigation strategies). Some of these tools provide templates that can be adjusted to add/change threat vectors, risk mitigation strategies and the ranking of the threats detected. But despite these benefits, they lack a fully automated approach of Application Threat Modeling. To enable efficiencies in the process, consideration for the use of automation tools is a must.

Automation Tools

Organizations need tools that will enable the ability to inject Application Threat Modeling into the DevOps pipeline. By its nature, DevOps will support automation which requires removing the need for human friction. Therefore, Application Threat Modeling tools should be able to be embedded as part of the automation stages to help with the detection and the risk mitigation strategies proactively in the pipeline. This level of efficiency will provide teams with a clear direction to reduce risk in the application implementation as part of the normal DevOps cycle. As changes are introduced in the architecture, the integration of tools for Application Threat Modeling in the DevOps pipeline can help identify those pressing security issues that must be addressed.

There are several scenarios under which the use of Application Threat modeling tools in the DevOps pipeline will help with the planning of the activities needed to secure applications and their implementation. For instance, creating issues in the issue tracker as a result of the detection and risk mitigation strategies identification enabled by the use of an Application Threat modeling tool is a great example of how to be proactive with those activities that are intended to reduce threat vectors in software. With this approach, every cycle or Sprint during the project implementation will ensure that any changes impacting the security level or security requirements of the application are going to be considered and included as part of the planning through a feedback loop. This approach will reduce the need for a human to manually document all the issues associated with changes in the architecture.

As stated early on in this paper, the Application Threat Modeling exercise is not a static process. Application architectures will change over time. Data sensitivity could change, new components could be added or there could be changes in the way components interact. Those changes will trigger the need to reassess the application architecture to detect and mitigate new threat vectors. This dynamic exemplifies why embedding threat modeling in the DevOps pipeline is so crucial. Having a feedback loop supported by Application Threat modeling in the DevOps pipeline will help detect threat vectors associated with changes in the architecture or the associated components. This detection will be supported by the automated creation of associated security issues in the issue tracker for the development team. Once the issues are created, teams can plan any associated remediation and address the risk mitigation strategies in their implementation. This constant feedback loop provides a proactive mechanism to be on top of any potential risk associated with recent changes in the architecture.

Conclusion

The ability to detect/mitigate threat vectors and risks before they even materialize provides a huge advantage to organizations against threat actors. This proactive approach enables the organization to be more resilient and efficient in its operational execution including the allocation of resources. Having the ability to spend resources in areas that will add value to the business is a better approach than having to spend those same resources on reactive countermeasures as a result of a compromise. To enable organizations to be proactive, Application Threat Modeling provides a systematic methodology for the detection and identification of risk mitigation strategies needed to reduce risk.

The Application Threat Modeling methodology provides mechanisms to efficiently map the resources in an architecture. Once these resources and the architecture is laid out, the methodology enables the detection of threat vectors using different approaches. Even though this paper presented STRIDE as a methodology to detect threat vectors, there are many others (PASTA, TARA, etc.). The adoption of a detection methodology will depend on the approach needed by the organization. All these methodologies have different key implementation features to be considered (See ***Determine and Rank Threats*** section for more information).

Once the threats have been identified and ranked, the Application Threat Modeling provides an approach to map risk mitigation strategies and countermeasures for each threat. This process will provide great visibility in terms of the effort needed to secure the architecture. This mapping exercise can leverage approaches such as DREAD to ensure proper coverage for the alignment of remediation efforts. Leveraging the ranking along with the countermeasure needed creates a roadmap for the organization to decide how to spend its resources on remediation tasks.

Even though the methodology can be executed manually, having multiple applications changing as well as new applications included in the mix will not allow the methodology to scale. The white paper provided examples of how to leverage automation to create efficiencies while executing the three steps of Application Threat Modeling. Embedding the methodology in DevSecOps must be part of the strategic vision of organizations. There are multiple options available depending on whether we are analyzing architecture on-premises or in the cloud. Cloud tools provide better opportunities to fully automate the methodology. As resources are added to a cloud tenant or environment, existing tools can interpret these changes, map the new architecture, detect threats, and provide risk mitigation strategies.

For proper planning, the paper covered the importance of the feedback loop. The identification of risk mitigation strategies for the corresponding threats will require proper coordination within the project due to the limited nature of resources. Using the ranking of the threats will help facilitate the analysis. Having an integrated feedback loop to the issue tracker directly from the evaluation in the DevOps pipeline is a key component to supporting better decision-making during the planning of each build cycle.

The combination of threat detection, ranking, risk mitigation strategies, and automation integration not only in the DevOps pipeline but also in the feedback loop enables the organization on multiple fronts. First, the constant process of evaluating changes in the architecture that could potentially put it in harm's way. Second, because of the limited nature of the resources, automation ensures proper and constant visibility of the outstanding risks. Finally, the feedback loop serves as the final piece in the planning and analysis of the features to be included in each build cycle iteration to ensure risk is mitigated on time.

References

Autodesk. (2020). *GitHub - Autodesk/continuous-threat-modeling: A Continuous Threat Modeling methodology*. GitHub. https://github.com/Autodesk/continuous-threat-modeling

Barreras, R., Keim, P., Pabon, W., Vega, J., & Zuehlke, A. (2021). Software Security Controls: Metrics Automation. ND-ISAC. https://ndisac.org/blog/wp-content/uploads/2022/02/ND-ISAC-Security-Controls-Metrics-Automation-Final.pdf

Conklin, L. (n.d.). *Threat Modeling Process*. Owasp.org. https://owasp.org/www-community/Threat_Modeling_Process

Donovan, F. (2021, January 11). *What is STRIDE Threat Modeling?* Cloud Security. https://securityintelligence.com/articles/what-is-stride-threat-modeling-anticipate-cyberattacks/

EC-Council. (n.d.). *DREAD Threat Modeling: An Introduction to Qualitative Risk Analysis*. EC-Council. https://www.eccouncil.org/cybersecurity-exchange/threat-intelligence/dread-threat-modeling-intro/

Gossett, S. (2021, July 8). *What Every Developer Should Know About Threat Modeling*. Builtin.com. https://builtin.com/cybersecurity/threat-modeling

Nick. (2022, July 24). *PASTA Threat Modeling*. Threat-Modeling.com. https://threat-modeling.com/pasta-threat-modeling

Object Management Group. (2022). *OMG SysML Home | OMG Systems Modeling Language*. Object Management Group. https://www.omgsysml.org/

Object Management Group. (2022b). *Welcome To UML Web Site!*. Object Management Group. https://www.uml.org/

Onapsis Research Labs. (2020). *DevSecOps Cycle* [Image]. Onapsis. https://onapsis.com/sites/default/files/infographic.png

OWASP. (2022). *Abuse Case - OWASP Cheat Sheet Series*. OWASP Cheat Sheet Series. https://cheatsheetseries.owasp.org/cheatsheets/Abuse_Case_Cheat_Sheet.html

OWASP. (2022, July). *OWASP pytm | OWASP Foundation*. OWASP Pytm. https://owasp.org/www-project-pytm/

Feiler, P. H., Lewis, B. A. and Vestal, S. (). "The SAE Architecture Analysis & Design Language (AADL) a standard for engineering performance critical systems," 2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006, pp. 1206-1211, doi: 10.1109/CACSD-CCA-ISIC.2006.4776814.

Revuelta, S. (2020, August 26). Threat modeling: 4 key questions. Securing Software. https://securingsoftware.blog/2020/08/26/threat-modelling-4-key-questions/

Schneider, C. (2020). Threagile - Agile Threat Modeling Toolkit. Agile Threat Modeling. https://threagile.io/

Snyk. (2020, October 23). User Story Threat Modeling: It's the DevSecOps Way - Alyssa Miller [Video]. YouTube. https://www.youtube.com/watch?v=hU3-33wDyAo

Tarandach, I., & Coles, M. J. (2020). Threat Modeling: A Practical Guide for Development Teams (1st ed.). O'Reilly Media.

ThreatModeler. (2022, Feb 4). HOW TO AUTOMATE THREAT MODELING USING THREATMODELER AND AVOCADO. ThreatModeler. https://threatmodeler.com/automate-threat-modeling-using-threatmodeler-avocado/we45. (2020). Hello from Threat Playbook | Threat Playbook. Threat Playbook. https://threatplaybook.io/

What is DevSecOps? (2018, April 12). Red Hat Technology Topics. https://www.redhat.com/en/topics/devops/what-is-devsecops

@zeroXten. (2019, June 27). threatspec — Continuous threat modeling, through code. Threatspec. https://threatspec.org/

Appendix A

Autodesk Continuous Threat Modeling (A-CTM)

Some tools can express threat models as code to provide more flexibility. To manage such an implementation, a methodology is needed to maintain the code. For the purpose of this paper let’s have a high-level overview of Autodesk Continuous Threat Modeling (A-CTM). A-CTM (Autodesk, 2020) was developed by Autodesk to allow development teams to perform threat modeling with minimal initial security knowledge and lower dependency on security experts. In this methodology, we have two main roles: Curators and other stakeholders (developers, architects, sys admins, etc.).

The Curator is responsible for keeping the threat model current as it evolves. They own a queue in the development team’s bug tracking system where tickets are treated as input events for consideration in the threat model. The Curator is responsible for adding the required information to tickets in the bug tracking system and updating or closing them, as appropriate. As a first step, the Curator and the stakeholders define the scope of the threat model, identifying all the important assets.

With this information, the organization should be able to leverage the tool to threat model every story. A-CTM provides a checklist that can be used as a starting point for threat modeling. Only a few changes will be "security notable events:" modifications to the attack surface, the security posture, or the secure configuration of the system. Threat modeling can be considered complete if the following criteria are met (“Continuous Threat Modeling Handbook - GitHub”):

- All the relevant diagrams are in the document
- The findings are documented
- If we are dealing with a "security notable event", a review request should be created
- Every bug is documented in the development team’s bug tracking system
- Request a review of the threat model from your security team for new releases

Appendix B

Threat Modeling from code tools

There are instances when threat modeling needs to provide more flexibility than the basic functionality and capabilities offered by Application Threat Modeling standalone and automation tools. In those instances, teams need to leverage a different approach that would provide capabilities to help with all the phases of Application Threat Modeling. For example, if organizations have a need to embed Application Threat Modeling in the source code they are creating, there are some tools that would allow the use of annotations in support of such a strategy. Below are some existing tools that can help provide more flexibility for those instances when code is a must-have as part of the methodology implementation. The different tools presented below are high-level examples of available capabilities and are not intended to be an exhaustive list.

Tools leveraging Code annotations

An example of this approach is ***threatspec*** (@zeroXten, 2019). This tool relies on code annotations to describe threat specifications and generates data-flow diagrams and reports. Using annotations, developers can describe the application components and their relations, threats, mitigation controls, tests, what we will do with the risk, and custom fields, among other concepts.

Figure 3 provides relevant data related to a vulnerability and its impact as detected by the threat modeling phases.

Figure 3

Annotation, Function, and coding used in threatspec to describe threat specs.

threatspec comments in Go (golang)

```
/*
@exposes WebApp:App to XSS injection with insufficient input validation:
“description: An attacker can inject malicious JavaScript into the web form”
impact: high
ref: #TRACKER-123
*/

func editHandler(w http.ResponseWriter, r *http.Request, title string) {
p, err := loadPage(title)
if err != nil {
p = &Page{Title: title}
}
renderTemplate(w, "edit", p)
}
```

threatspec can create PDFs, text, and JSON files. These files can be consumed by CI/CD pipelines to generate tickets or break the build.

Tools leveraging Abuse Cases

Another approach, followed by ***ThreatPlaybook***, is the use of Abuse Cases (we45, 2020). The developer provides abuse cases for their use cases/features, using a threat library supplied by the tool or creating custom threats. ***ThreatPlaybook*** will generate threat model diagrams including vulnerabilities and will correlate those vulnerabilities to their related threat models, based on CWE IDs. Using the Robot Framework, developers, or security engineers can capture security test cases, describing how security testers could validate specific vulnerabilities using tools like ZAP (Zed Attack Proxy) or NMAP.

Architecture Description Language

In this case, we can use an architecture description language (ADL) to describe the representation of a system based on its source code. Architecture Analysis & Design Language (P. H. Feiler, B. A. Lewis and S. Vestal, 2006) and the Acme description language for component-based system modeling are two examples of this approach. Architecture Analysis & Design Language is an international standard used in the fields of avionics and automotive systems and requires a license to use. Acme is freely available and is used on less complex systems.

Both languages can represent processes, data stores, their relationships, specific configurations, and point of interaction between them. Unified Modeling Language (UML) and Systems Modeling Language (SysML) are Object Management Group (OMG) standards that can also be useful to represent threat models. But we still need a tool to perform an automated analysis of those models. Such a tool could be used to describe the system/application as a collection of objects in an object-oriented way, including the paths followed by the data (data flows) when it is transferred between the different components. These data flows should be labeled, indicating business value, criticality, or other factors that may impact how they are used. This model should be easy to read to developers.

Now that we have a system description, we can generate diagrams and threats from that description. Next, we will mention some tools that meet these characteristics.

The pytm Tool

OWASP pytm is a Pythonic framework for threat modeling with code. You can define your system in Python using the elements and properties described in the ***pytm*** framework. Based on your definition, ***OWASP pytm*** can generate a Data Flow Diagram (DFD), a Sequence Diagram, and threats to your system. ***pytm*** uses a subset of threat vectors defined in the Common Attack Pattern Enumeration and Classification (CAPEC). Currently, ***pytm*** can detect more than 100 threats. Table 9 provides some examples of common threat vectors.

Table 9

List of some common threat vectors impacting an application as detected by pytm.

Threat Id	Threat Vector
INP01	Buffer Overflow via Environment Variables
INP02	Overflow Buffers
INP03	Server Side Include (SSI) Injection
INP04	HTTP (Hypertext Transfer Protocol) Request Splitting
CR01	Session Side jacking
CR02	Cross-Site Tracing

OWASP pytm can create reports in any text-based format, including HTML, Markdown, RTF, and plain text.

Threagile Tool

Threagile is another tool for threat modeling with code, but it uses a YAML file to model architecture with its assets in an agile declarative fashion. Upon execution of the toolkit, a set of risk rules execute security checks against the architecture model and create a report with potential risks and mitigation advice, generating data-flow diagrams. Using this approach, we can track the risk inside the **Threagile** YAML model file, reporting the current state of risk mitigation as well. **Threagile** can either be run via command line or started as a REST-Server.