

Automated Threat Analysis and Management in a Continuous Integration Pipeline

Laurens Sion, Dimitri Van Landuyt, Koen Yskout, Stef Verreydt, Wouter Joosen
imec-DistriNet, KU Leuven
Heverlee, Belgium
firstname.lastname@cs.kuleuven.be

Abstract—Security and privacy threat modeling is commonly applied to systematically identify and address design-level security and privacy concerns in the early stages of architecture and design. Identifying and resolving these threats should remain a continuous concern during the development lifecycle. Especially with contemporary agile development practices, a single-shot upfront analysis becomes quickly outdated. Despite it being explicitly recommended by experts, existing threat modeling approaches focus largely on early development phases and provide limited support during later implementation phases.

In this paper, we present an integrated threat analysis toolchain to support automated, continuous threat elicitation, assessment, and mitigation as part of a continuous integration pipeline in the GitLab DevOps platform. This type of automation allows for continuous attention to security and privacy threats during development at the level of individual commits, supports monitoring and managing the progress in addressing security and privacy threats over time, and enables more advanced and fine-grained analyses such as assessing the impact of proposed changes in different code branches or merge/pull requests by analyzing the changes to the threat model.

Index Terms—threat modeling, threat analysis, security by design, privacy by design, threat management

I. INTRODUCTION

Security and privacy require continuous attention throughout the software development lifecycle (SDLC). It is well-known, though, that absolute security cannot be achieved, and compromises must be made. In practice, security efforts should therefore be directed towards conscious management of risk and security debt [1], a form of technical debt. Without sufficient attention to security, the security debt and risk may increase beyond acceptable levels, increasing the likelihood of security incidents and associated losses, and making it hard to recover without major investments and delays.

Security and privacy threat modeling techniques [2]–[5] are typically applied in the early phases (requirements and design) of the SDLC. These approaches reason at an abstract level about the system, often in the form of a Data Flow Diagram (DFD), to elicit many potential security threats. In a next step, mitigations for the most important of these threats (in terms of risk) are selected, which can then be incorporated during the software development.

Current threat modeling approaches are not well-aligned with contemporary development practices. Modern software

development happens at a fast pace with frequent changes to the code base to introduce new functionality, fix bugs, and refactor the design. Continuous integration (CI) is one of the enablers of this fast pace. Threat modeling, on the other hand, is often a manual, time-consuming, one-off (or infrequently repeated) activity conducted in workshops involving experts and numerous stakeholders [6], [7]. This prohibits frequent re-evaluation as the software design evolves. This in turn is considered problematic, because the goal of threat modeling is precisely to identify threats that carry a significant risk, and (because they are linked to the design) that may be hard to mitigate afterwards.

An additional problem associated with infrequently revisiting a threat model is that it hampers adequate management of the risk and security debt as part of project management. Indeed, effective decision-making relies on having a clear view on the current status and progress, the impact of the possible choices, and the effectiveness of past decisions and efforts. Infrequent threat modeling only yields a coarse-grained view on the progress that is being made, though, precluding swift reactions to emerging risks.

This paper proposes CTAM (Continuous Threat Analysis & Management), a toolchain that addresses these problems by technically integrating an automated threat analysis and assessment activity in a continuous integration pipeline. This enables stakeholders to monitor threat modeling results, and track and manage the evolution of risk based on information that evolves together with the implementation. CTAM leverages the possibilities offered by automated threat modeling tools to achieve traceable, systematic, and frequent re-assessments. The input for an existing threat modeling tool, which consists mainly of a model-based representation of the system (e.g., a DFD), is placed and maintained alongside the source code in a version control system. The automated threat modeling tool is then used as a standalone analysis engine in a continuous integration job. By combining the automated analysis results with the existing version information from the repository, the current state and historic evolution of the threat model, as well as the impact of suggested modifications in different branches, can be assessed and presented on a dashboard.

This paper introduces the following contributions: (i) it presents the CTAM toolchain, leveraging existing threat modeling tools, to enable systematic, automated threat analysis and management; (ii) it validates the prototype on an application

This research is partially funded by the Research Fund KU Leuven and the Cybersecurity Initiative Flanders.

case, demonstrating its capability to recognize a number of different risk evolution patterns; and (iii) it provides an in-depth discussion on automated threat modeling as part of a continuous integration pipeline.

This CTAM toolchain is a technological enabler that supports the development and evaluation of more advanced analysis techniques to measure and monitor threat modeling progress.

This paper is structured as follows. Section II describes the related work on threat modeling and continuous quality assessment. Next, Section III presents CTAM and the implementation aimed at demonstrating its feasibility. Then, Section IV applies CTAM on an application case of Software-as-a-Service (SaaS) document generation and delivery platform and illustrates the type of analysis that it enables. Afterwards, Section V provides a discussion on the use of CTAM, including the possibility of using other threat elicitation engines, the importance of the consistency of the model with the source code, and additional analysis types that become available with CTAM. Finally, Section VI concludes the paper.

II. RELATED WORK

Continuous integration refers to software development practices that are centered heavily around a central version control system and code repository. These systems implement a pipeline of automated activities which typically include activities aimed at quality control (e.g., code syntax checking), automated testing (e.g., regression testing, integration testing, acceptance testing), and automated building and build management. Automating these activities allows for frequent execution at the level of individual code commits, providing the developer with rapid feedback, and shortening the time to address issues. These key principles enable ensuring a certain degree of quality assurance.

This section outlines the related work on threat modeling in this context. First, the current state of the threat modeling support during development is discussed. Next, the state of the art in continuous integration and security analysis activities in this context is outlined.

A. Threat modeling support during development

Several threat modeling tools and approaches, such as IriusRisk [8] and Autodesk CTM [9] promote the integration of threat modeling during development specifically by linking threats to issues in an issue tracker. While this enables tracking the progress regarding the identified security and privacy threats, the threat mitigation progress is monitored in the issue tracker, rather than in the system model. Furthermore, while such approaches may support versioning of the system model, they do not support analysis of the evolution of a threat model.

More closely aligned with the source code is ThreatSpec [10]. It provides a set of code annotations that can assist in constructing and maintaining a DFD model by inserting comments at the relevant locations in the source code. ThreatSpec does not perform any threat elicitation by itself, so the extracted model will have to be analyzed manually or with another tool to obtain a list of threats to further analyze and

aggregate. It does allow documenting threats and mitigations through code annotations so that the results of the threat elicitation activity can be captured as well.

Pytm [11] generates diagrams (DFDs and sequence diagrams) and threats based on a system model expressed in Python code. Such a representation enables versioning the system model together with the source code. It does not, however, provide risk estimates for threats, so monitoring progress in terms of risk reduction requires additional analysis.

SPARTA [12] is an eclipse-based threat modeling tool that elicits security and privacy threats based on XMI-files of solution-enriched DFDs and threat catalogs (for example, the STRIDE [3] and LINDDUN [13] threat types are supported). SPARTA provides risk estimates for individual threats and calculates the aggregate values for the system, which could be used to monitor the threat mitigation progress over time. SPARTA does not support any historical analysis of threat mitigation progress.

Threagile [14] generates threat model reports based on YAML-files of the architecture and its assets and provides pipeline integration to do so in a continuous fashion. However, such analyses are only focused on a single version of the system; it does not analyze how those threat models evolve over multiple versions of the system.

OWASP's Threat Dragon [15] is an open-source threat modeling platform for system modeling and threat elicitation. Its documentation mentions that future versions should provide an API for pipeline integration, but this is not supported at the time of writing.

B. Quality assessment in continuous integration pipelines

Several approaches exist that conduct frequent code analysis for measuring the impact on qualities such as performance, maintainability, security, etc. For example, the PerfCI tool [16] integrates automated performance benchmarks to identify potential performance regressions over time. Vassallo et al. [17] in turn presented an approach that automates and integrates the identification of bad practices, anti-patterns, or common misconfigurations in a CI pipeline.

The automation of these activities is a key enabler for extensive data analytics at the level of the code base: the evolution of a code base in terms of software quality can be monitored and evaluated [18] over longer periods of time.

Static code checkers (SAST) allow for the identification of vulnerabilities as a result of code-centric analysis [19], [20]. As discussed by Rangnau et al. [21], integrating dynamic security testing (DAST) is more challenging as these more advanced analysis techniques incur a more significant performance cost, to the extent that the total cost of their integration in a CI pipeline might become prohibitive.

To our knowledge, model-based analysis activities that identify threat scenarios at the level of an abstraction model of the system (i.e. threat modeling and threat-based risk assessment) have not yet been integrated in a practical CI pipeline, with the exception of Threagile [14] which does not consider the analysis over time. Yet, threat modeling experts

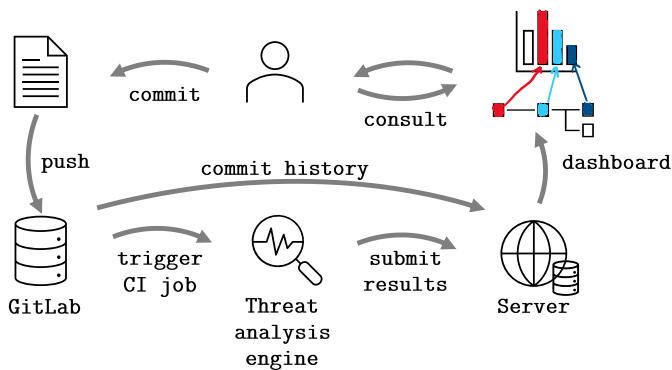


Fig. 1. Overview of the approach.

On the left-hand side, changes to the codebase and model are committed and pushed to a repository on GitLab. This triggers the CI jobs that will run the threat analysis engine (bottom-center), of which the results will be submitted to the server. Finally, the developers can consult the impact of their changes on a dashboard presenting the analysis results (right-hand side).

and advocates strongly encourage frequent re-evaluation of the outcome of a threat modeling and analysis exercise [3], [22]–[24] throughout the development of a system. In this article, we present the practical implementation of such an activity in the GitLab DevOps platform [25].

III. CONTINUOUS THREAT ANALYSIS & MANAGEMENT

The main goal of CTAM is to automate continuous threat analysis, management, and progress monitoring by integrating it in continuous integration pipelines. This is achieved by: (i) storing the model together with the source code in version control (this model contains the DFD, the applied security and privacy solutions, and inputs for the risk analysis); (ii) for every push to the repository, running a continuous integration analysis job to elicit security or privacy threats and perform a risk analysis on them; (iii) collecting and aggregating these results in the CTAM server; and (iv) making these results available as feedback to the developers. Figure 1 provides a graphical overview of the approach. The next subsections will elaborate on the technical realization of steps (ii) and (iii), followed by a description of the types of analysis activities that can be performed in this context.

A. Threat analysis engine

For the individual threat analysis, CTAM currently leverages SPARTA [26] and its enablers, namely (i) the automated generation of threats at the basis of a (customizable) threat catalog, (ii) per threat, a risk estimation step [27] that takes into account many factors documented in the input model (e.g., the application of security solutions [12], a description of the affected data subjects in case of privacy [28]), and finally, (iii) the aggregation and disclosure of these outcomes. All the relevant data (i.e. the DFD model, solutions, and threat type catalogs) for SPARTA’s analysis are contained in one (or more) model files that will be read by the analysis engine.

SPARTA elicits threats by performing model queries on the supplied model. The threat type catalogs contained in the model specify the criteria for the threats to be applicable and can be used to encode, for example, element- or interaction-based STRIDE threats as well as more complex threat patterns. As the main input of the analysis is the model of the system under development, the scope of the analysis is necessarily limited to the design of this system, as there is no operational context to take into account. It is, however, possible to explicitly include such details in the model, but these details will not correspond with concrete elements in the source code.

For the risk analysis and prioritization, SPARTA processes the information in the model (i.e. asset value, strength of security solutions, etc.) to determine how effective the countermeasures are to protect against the elicited threats. The resulting value is the expected loss, expressed in the same unit as the asset value. It is up to the developers or business stakeholders to provide this information in a unit that is convenient to them.

We created a dockerized version of the SPARTA threat elicitation and assessment engine, which reads a configuration file (specifying the model file and the submission server), analyzes the model (i.e. elicits security or privacy threats and performs a risk assessment of these threats), and submits the threat elicitation results to the submission server. The bottom of Fig. 1 depicts these steps graphically. The threat analysis engine in the center runs on the last commit, analyzes the model contained therein, and submits the results.

The docker container enables the use of SPARTA in GitLab CI jobs [25]. The only additional information required in the repository is the DFD model file and the aforementioned configuration file. Because individual commits are analyzed, that model file will need to accurately reflect any changes that are made to the codebase. Section V further discusses the need for an accurate model of the system under analysis.

Finally, it is possible to use other threat modeling tools for the elicitation, as long as they yield appropriately formatted threats and risk estimates for submission to the CTAM server (see Section V-A for a discussion on this).

B. Server

The server component is a Spring boot application. Registering a new project requires a deployment token and the repository URL. This is used by the server to retrieve the commit history from GitLab. When analysis results are submitted by the threat analysis engine, these results are associated with the corresponding git commits to enable the construction of an overview dashboard (depicted in the right-hand side of Fig. 1).

When a developer consults the CTAM dashboard, the server constructs a historical overview of the evolution of the aggregated risk by combining the analysis results for the commit ancestors on the main branch of the repository. This comprises the following calculations per commit: (i) the threat count; (ii) the total inherent risk (by aggregating the inherent risk of the individual threats); (iii) the total residual risk (also by aggregating); (iv) the risk reduction (as the mitigated risk over

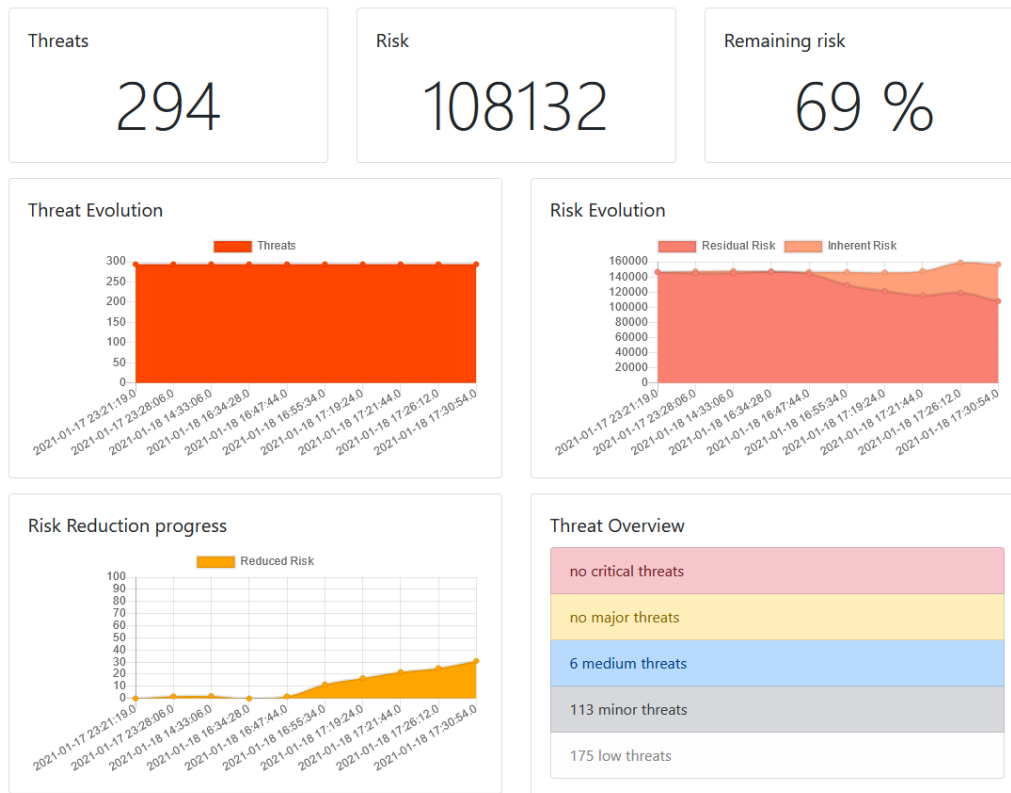


Fig. 2. CTAM dashboard.

The dashboard presents the main metrics of the last commit (top row), the evolution of the number of threats and the residual and inherent risk (middle row), and the progress in reducing the risk and the overview of the prioritized threats in the last analysis results (bottom row).

the inherent risk); and (v) the classification into categories (by binning the threats in equal intervals based on the largest inherent risk encountered in the analysis of the commit). This initial set of measurements can be expanded with additional ones that can be calculated from the submitted threat results such as the most frequently occurring threat types, the system elements with the largest residual risk, etc.

Figure 2 shows the CTAM dashboard containing information on the evolution of threats, the estimated risk, and the progress in reducing that risk for a specific project. In addition to a project-wide overview, the developer can also select any analyzed commit from the overview to obtain the detailed analysis results for that specific version of the system, including the full list of elicited threats.

C. Analysis

CTAM enables several types of analysis activities through its systematic collection of threat analysis information as the software system evolves over time. It currently leverages SPARTA for the threat analysis results and hence also relies on the residual and inherent risk values that SPARTA provides. Inherent risk represents the risk disregarding any security or privacy solutions (i.e. the risk if fully vulnerable). Residual risk represents the risk taking into account security and privacy solutions (i.e. the inherent risk minus the effect of security and privacy solutions). However, it is also possible to use different

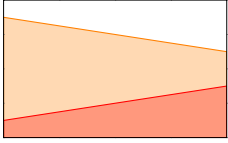
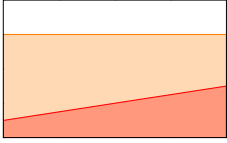
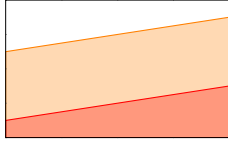
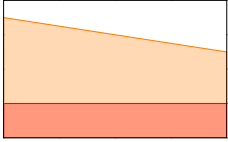
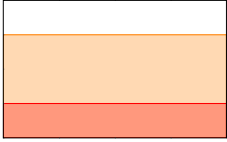
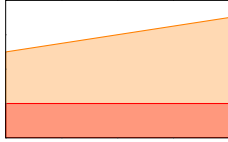
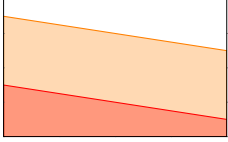
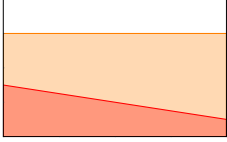
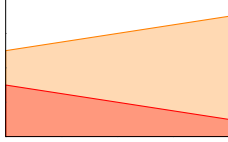
risk scores, as long as they incorporate the effect of partially or completely mitigating threats in the system.

By collecting the inherent and the residual risk for every committed version of the system under consideration, the overall progress in securing the system can be assessed. Table I presents an overview of the different risk evolution patterns that may emerge through the combination of a decrease, stable, or increase of the inherent and residual risk values.¹ These patterns allow developers to gain insight into the progress that is being made over time. For example, these patterns will show which commits focus on security (reducing the residual risk) or on expanding functionality (increased residual risk). It will also show how these types of changes manifest themselves over time (e.g., whether security is always considered after new functionality has been introduced).

The next section discusses the functional validation of our tool and illustrates with a number of concrete changes in an example application how the effects of these changes can indeed be observed in the aggregated overview.

¹While Table I shows those patterns as nine distinct possibilities, there is actually a continuum as one sort of risk may, for instance, decrease more rapidly than the other one. For example, if both secure and insecure functionality (cells VII and IV in Table I) is removed from the system in a single commit, both the residual and the inherent risk plot lines will decrease, but the residual risk line will have a more shallow slope.

TABLE I
RISK EVOLUTION PATTERNS.

Residual Risk <i>(bottom line)</i>		Inherent Risk <i>(top line)</i>			
		Decrease	Stable	Increase	
Increase	I		II		
		Remove security solutions*	Remove security solutions	III	
				Add insecure functionality	
Stable	IV		V		
		Remove secure functionality	No security-relevant changes	VI	
				Add secure functionality	
Decrease	VII		VIII		
		Remove insecure functionality	Add security solutions	IX	
				Add security solutions*	

Plots for the different risk evolution patterns due to decreasing, stable, or increasing inherent and residual risk values. The area plots are not stacked (i.e. the inherent risk consists of the entire area under the line including the residual risk). Combinations of these patterns are possible to express to different slopes of the inherent and residual risk plot lines. For example, combining VII and IV results in a more slowly decreasing residual risk, combining III and VI in a more slowly increasing residual risk, etc.

* Solutions that introduce additional risk with regard to, for example, cryptographic key material.

IV. FUNCTIONAL VALIDATION

This section presents the functional validation of CTAM on an illustrative application case. First, the application itself is described. Next, a number of deliberate change scenarios are introduced to assess the effect of different types of changes (e.g., new functionality, securing existing functionality). After each of these changes, the resulting model is analyzed, and the analysis results are collected. Finally, the results for each change scenario are discussed, highlighting the usefulness of CTAM in measuring and monitoring the security impact during software development.

A. Description of the case

We apply our prototype on a SaaS application for generating and delivering PDF documents (e.g., invoices, pay slips), via different delivery channels (e.g., email, print) to end users. One of those channels is a hosted Personal Document Store (PDS) on which users can login to retrieve documents sent to them. Figure 3 shows the DFD of this system. The center part of the figure contains the core of the system's delivery services. The left-hand side contains the integration with third parties for delivery via print, email, etc. The right-hand side models the hosted PDS from which users can directly access their documents. The next section will refer to this diagram when explaining the different changes that will be made to this system to validate the CTAM prototype.

B. Change scenarios

We validate our approach with five specific change scenarios (affecting both functionality and security solutions). Each of these changes are applied to the DFD model of the document processing system in separate commits to enable the analysis of their impact. The security solutions mentioned below include encryption, authentication, and access control to protect against information disclosure, tampering, and spoofing.

- C0 The initial version of the system does not contain the PDS functionality (i.e. no E4, DS2, DS3, P3, nor any of the data flows to or from them), nor the banking integration (i.e. no E2 or any its data flows), nor any security solutions to protect the communication with E3.
- C1 The first commit introduces secure functionality by adding the banking integration (E2) together with some security solutions to protect the communication with E2.
- C2 This commit exclusively affects security, by introducing a security solution to protect the communication with the email provider (E3).
- C3 This commit adds the PDS functionality (i.e. E1, DS2, DS3, P3, and the data flows), but does not introduce any security solutions to secure this functionality.
- C4 This commit adds security solutions to protect the communication between the PDS users (E4) and the PDS. This does not secure all the functionality introduced by C3.

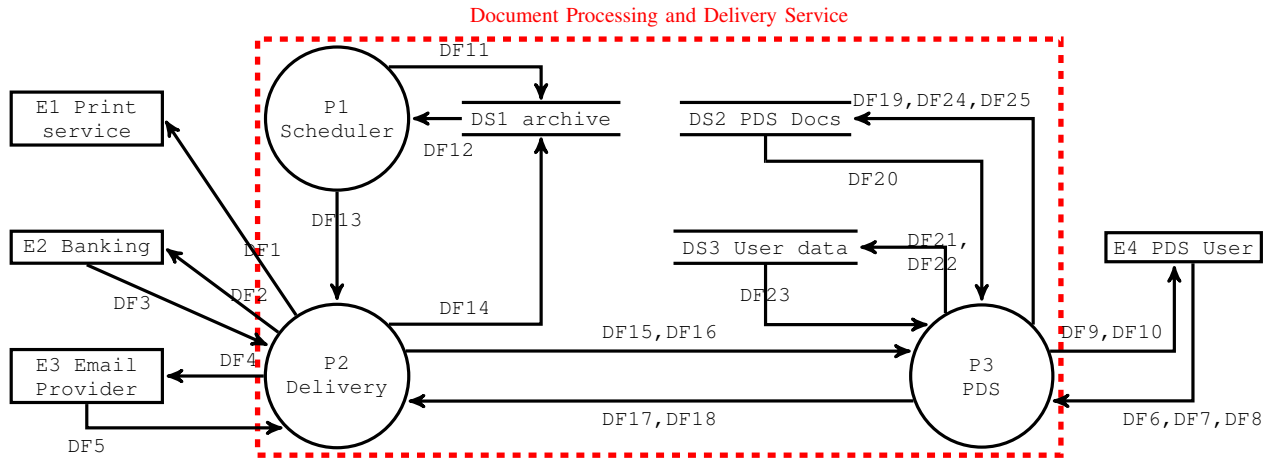


Fig. 3. Data Flow Diagram (DFD) of the document processing and delivery service.

This diagram shows the delivery components of this system together with the storage in the center of the diagram. The left-hand side shows various third party delivery services, while the right-hand side shows the hosted personal document store (PDS) from which users can retrieve the documents sent to them. To improve the readability of the diagram, multiple flows in the same direction are combined together (e.g., DF17, DF18).

C5 Finally, this commit removes all data stores and the scheduler (i.e. remove P1, DS1–3, and their data flows).²

Each of these changes are introduced in separate commits and pushed to a GitLab instance to trigger the continuous integration jobs which analyze the modified DFD model and submit the analysis results to the CTAM server.

C. Results

Figure 4 shows the analysis results as reported by the CTAM server after receiving the results from SPARTA for each of the introduced changes. This section revisits each of these changes to explain the risk evolution pattern encountered in the results and refers to the corresponding cells in Table I.

C1 As shown in Fig. 4, the residual risk line is not perfectly stable: the change actually did result in an increase of the residual risk due to the fact that the solutions do not fully mitigate the total risk introduced by the new functionality. Hence, the change corresponds with cells VI and III in Table I.

C2 With the exception of some small variance in the risk estimation, the inherent risk remains stable, while the residual risk is reduced. This corresponds with cell VIII in Table I.

C3 This change scenario involves a substantial modification, as also visible from the analysis results. As this change scenario does not consider security, it results in both an increase of the inherent risk and the residual risk. As such, this is an example of the pattern in cell III in Table I.

C4 As this change scenario only secures the interaction between the end-user and the PDS, it does not mitigate all the newly-introduces risk from the previous change scenario. As it only introduces security solutions, it again corresponds with cell VIII in Table I.

C5 The final change removes insecure functionality from the model (all internal storage and the scheduling process).

²While this is an unrealistic modification, it demonstrates the impact of removing insecure functionality from the system.

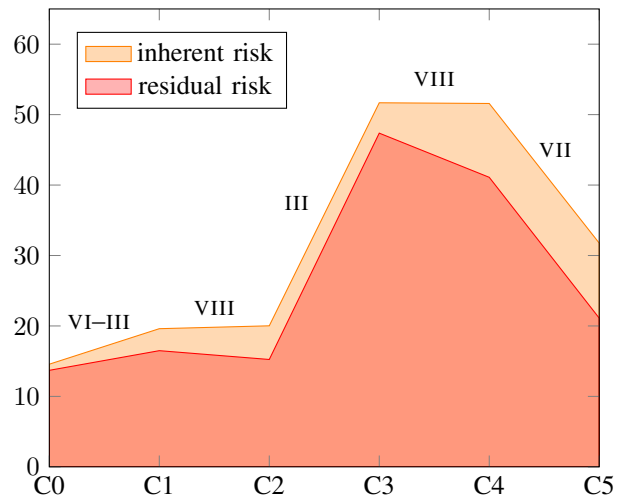


Fig. 4. Overview of the analysis results for the different changes. This plot shows the resulting inherent and residual risk values in the example application for each of the changes.

This results in a substantial reduction of both the inherent and the residual risk. This corresponds with cell VII in Table I.

In future work, we plan to perform a more elaborate evaluation by providing reconstructed DFD models for historical commits of an existing open-source application to support an historical analysis of the threat modeling progress in mitigating threats, construct a better picture on how security is considered in practice, and verify the recovery of the presented risk evolution patterns in a real-world application.

V. DISCUSSION

This section discusses the use of different threat analysis engines, the need for a model that is consistent with the source code of the system under consideration, how the approach requires the construction of an explicit design model of the

system, and the suitability of different metrics to monitor the evolution of a security architecture over time.

A. Threat elicitation engines

As discussed in Section III-A, the presented prototype is built around the threat elicitation engine of SPARTA [26] because of its powerful enablers. Any alternative threat modeling tool could in theory be adopted, provided that it generates a list of threats, with, for every threat, (i) the threat type, (ii) the affected DFD model element, (iii) the data flow, and (iv) the inherent (i.e. risk if fully vulnerable) and residual risk (i.e. risk assessment taking into account security and privacy solutions) indications. For example, CTAM could be integrated with Pytm [11] to elicit threats, provided that Pytm is extended with (i) a risk estimation approach, such as FAIR [29], or (more pragmatically) a translation of its current severity categories to numerical values; and (ii) the possibility to elicit threats that have been mitigated by a solution, to ensure that these can be taken into account when tracking the progress that is being made. This would, however, still lack SPARTA's support for security and privacy solutions and its detailed risk analysis breakdown (see Section V-C).

B. Input DFD model accuracy

As discussed above, the current version of CTAM is predicated upon the inclusion of an input DFD model in the code base that is kept up to date with the different commits and is gradually updated throughout the development activities. In case this model deviates (e.g., as a consequence of architectural drift [30]) from reality, the usefulness of the presented approach decreases drastically, as not all the generated threats will be relevant (false positives), or not all the relevant threats will be identified (false negatives). Additionally, the modifications in a single commit may not always necessitate changes to the model itself, as this depends on the granularity of the commits. There are opportunities, however, to systematically revisit the accuracy of the model as part of, for example, merge requests that introduce more considerable changes.

While the above argument applies to any threat modeling approach, the integration of these threat analysis activities into the code versioning system presents two opportunities for improvements in this regard. First, existing techniques of architectural reconstruction can prove useful to ensure or validate the accuracy of the input model vis-a-vis the committed code. Software reflexion models [31], [32], for example, could be used to automatically inform stakeholders when source code changes deviate significantly from the system model. Second, code-oriented threat modeling tools, such as ThreatSpec [10], that rely on code annotations for the construction of the input model, can remove the need for a separate centralized input model altogether. In future research, we hope to achieve a tighter integration between the threat modeling information and the corresponding source code.

C. Supported threat analysis activities

As demonstrated in Section IV, CTAM provides immediate feedback on the progress being made in creating a secure and

privacy-preserving design in terms of the *inherent risk* and the *residual risk* which are both aggregated results. These values are calculated and reported for each commit.

This degree of integration with version control systems allows for a number of interesting analyses on the evolution of a code base. For example, proposed changes in other branches or merge/pull requests can be analyzed and compared with the main branch to evaluate their security and privacy impact.

Because SPARTA performs a fine-grained risk assessment, more detailed intermediary risk analysis results can be used (e.g., the effectiveness of specific solutions, or the impact on specific data subject types) instead of the aggregated risk estimates per threat. This would allow the developer to perform more targeted assessments, e.g., the analysis of privacy risk from the perspective of a specific data subject type and its evolution over time, or focused on specific assets (e.g., credit card numbers or user data).

D. Security metrics

The systematic analysis and measurement of a software product necessarily brings us to the domain of software security metrics, a difficult, if not infeasible [33] endeavor. Despite the inherent difficulties, many proposals have been made in the literature to measure different security-relevant properties, such as dependency graphs [34], attack surfaces [35], and software metrics [36]–[38]. While the risk assessments of the elicited threats may not be suitable as a metric to compare the security of different software products in absolute terms, it does allow monitoring the progress that is being made in securing one specific system throughout its development. Furthermore, our prototype lays the groundwork and provides a generic framework for future evaluation of, and experimentation with, calculating and comparing different security or privacy metrics over time.

VI. CONCLUSION

Threat analysis is commonly performed in a single-shot operation, in the early stages of software development. Because of this, progress in threat mitigation is not actively revisited and monitored throughout later development stages such as the implementation and as the system evolves over time. Furthermore, as changes are made to the system, the originally anticipated threats may become obsolete while novel threats remain undiscovered.

In this paper, we introduced CTAM, a continuous threat analysis and management prototype that supports continuous threat modeling and elicitation and integrates this activity into a continuous integration pipeline in GitLab. By revisiting threat analysis as new changes are pushed to the source code repository, threat management becomes a continuous activity, and the progress in mitigating threats (both in applying appropriate security and privacy solutions as in making changes to existing functionality) can be more accurately monitored.

Integrating threat analysis activities in a continuous integration pipeline provides the following benefits. First, threat management becomes a continuous concern, rather than a

single-shot analysis on an outdated version of the system. Second, it provides guidance towards mitigating threats and keeps track of the progress. Third, it creates the need to maintain the architectural abstraction model of the system and forces developers to reflect on the broader architectural impact of their changes in terms of security and privacy.

REFERENCES

- [1] K. Rindell, K. Bernsmed, and M. G. Jaatun, “Managing Security in Software - Or: How I Learned to Stop Worrying and Manage the Security Technical Debt,” in *Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES’19)*. ACM, aug 2019, pp. 1–8.
- [2] A. Shostack, “Experiences threat modeling at Microsoft,” in *Modeling Security Workshop. Dept. of Computing, Lancaster University, UK, 2008*.
- [3] —, *Threat Modeling: Designing for Security*. Indianapolis, Indiana: John Wiley & Sons, 2014.
- [4] M. Deng, K. Wuyts, R. Scandariato, B. Preneel, and W. Joosen, “A privacy threat analysis framework: Supporting the elicitation and fulfillment of privacy requirements,” *Requirements Engineering*, vol. 16, no. 1, pp. 3–32, 2011.
- [5] K. Wuyts, “Privacy Threats in Software Architectures,” PhD Thesis, KU Leuven, Jan. 2015.
- [6] K. Yskout, T. Heyman, D. Van Landuyt, L. Sion, K. Wuyts, and W. Joosen, “Threat modeling: from infancy to maturity,” in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*. ACM, jun 2020, pp. 9–12. [Online]. Available: <https://dl.acm.org/doi/10.1145/3377816.3381741>
- [7] K. Tuma, C. Sandberg, U. Thorsson, M. Widman, T. Herpel, and R. Scandariato, “Finding security threats that matter: Two industrial case studies,” *Journal of Systems and Software*, p. 111003, May 2021.
- [8] IriusRisk, “IriusRisk,” 2021, <https://www.iriusrisk.com/>.
- [9] Autodesk, “Autodesk Continuous Threat Modeling,” 2021, <https://github.com/Autodesk/continuous-threat-modeling/>.
- [10] ThreatSpec, “ThreatSpec,” 2021, <https://threatspec.org/>.
- [11] I. Tarandach, “Pytm,” 2021, <https://github.com/izar/pytm>.
- [12] L. Sion, K. Yskout, D. Van Landuyt, and W. Joosen, “Solution-aware Data Flow Diagrams for Security Threat Modelling,” in *Proceedings of The 6th Track on Software Architecture: Theory, Technology, and Applications*, 2018.
- [13] L. Sion, K. Wuyts, K. Yskout, D. Van Landuyt, and W. Joosen, “Interaction-based Privacy Threat Elicitation,” in *Proceedings of the 4th International Workshop on Privacy Engineering – IWPE 2018*. IEEE, 2018.
- [14] Christian Schneider, “Threagile,” 2021, <https://threagile.io/>.
- [15] OWASP, “Threat Dragon,” 2021, <https://owasp.org/www-project-threat-dragon/>.
- [16] O. Javed, J. H. Dawes, M. Han, G. Franzoni, A. Pfeiffer, G. Reger, and W. Binder, “PerfCI: a toolchain for automated performance testing during continuous integration of Python projects,” in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2020, pp. 1344–1348.
- [17] C. Vassallo, S. Proksch, A. Jancso, H. C. Gall, and M. Di Penta, “Configuration smells in continuous delivery pipelines: a linter and a six-month study on gitlab,” in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 327–337.
- [18] R. Kozik, M. Choraś, D. Puchalski, and R. Renk, “Platform for software quality and dependability data analysis,” in *International Conference on Dependability and Complex Systems*. Springer, 2018, pp. 306–315.
- [19] S. T. Datko, “Static code analysis with gitlab-ci,” Tech. Rep., 2016.
- [20] G. B. Simpson, “CI/CD Software Security Automation,” Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2018.
- [21] T. Rangnau, R. v. Buijtenen, F. Fransen, and F. Turkmen, “Continuous security testing: A case study on integrating dynamic security testing tools in ci/cd pipelines,” in *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, 2020, pp. 145–154.
- [22] L. Sion, D. Van Landuyt, and W. Joosen, “The never-ending story: On the need for continuous privacy impact assessment,” in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2020, pp. 314–317.
- [23] D. Van Landuyt, L. Pasquale, L. Sion, and W. Joosen, “Threat models at run time: the case for reflective and adaptive threat management (nier track),” 2021.
- [24] Z. Braiterman, A. Shostack, J. Marcil, S. de de Vries, I. Michlin, K. Wuyts, R. Hurlbut, B. S. Schoenfield, F. Scott, M. Coles, C. Romeo, A. Miller, I. Tarandach, A. Douglan, and M. French, “Threat Modeling Manifesto,” <https://www.threatmodelingmanifesto.org/>, Nov. 2020.
- [25] GitLab, “GitLab CI/CD,” 2021, <https://docs.gitlab.com/ee/ci/>.
- [26] L. Sion, D. Van Landuyt, K. Yskout, and W. Joosen, “SPARTA: Security & privacy architecture through risk-driven threat assessment,” in *IEEE 2018 International Conference on Software Architecture (ICSA 2018)*, IEEE, 2018, [freely].
- [27] L. Sion, K. Yskout, D. Van Landuyt, and W. Joosen, “Risk-based Design Security Analysis,” in *Proceedings - 2018 IEEE/ACM First International Workshop on Security Awareness from Design to Deployment, SEAD 2018*, Gothenburg, Sweden, 2018.
- [28] L. Sion, D. Van Landuyt, K. Wuyts, and W. Joosen, “Privacy risk assessment for data subject-aware threat modeling,” in *2019 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2019.
- [29] J. Freund and J. Jones, *Measuring and Managing Information Risk: A FAIR Approach*. Butterworth-Heinemann, 2014.
- [30] B. Tekinerdogan, “Architectural drift analysis using architecture reflexion viewpoint and design structure reflexion matrices,” in *Software Quality Assurance*. Elsevier, 2016, pp. 221–236.
- [31] G. Murphy, D. Notkin, and K. Sullivan, “Software Reflexion Models: Bridging the Gap between Design and Implementation,” *IEEE Transactions on Software Engineering*, vol. 27, pp. 364 – 380, 05 2001.
- [32] J. Buckley, S. Mooney, J. Rosik, and N. Ali, “Jittac: A just-in-time tool for architectural consistency,” in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 1291–1294.
- [33] S. Bellovin, “On the brittleness of software and the infeasibility of security metrics,” *IEEE Security and Privacy*, vol. 4, no. 4, pp. 96–96, Jul. 2006.
- [34] V. H. Nguyen and L. M. S. Tran, “Predicting vulnerable software components with dependency graphs,” in *Proceedings of the 6th International Workshop on Security Measurements and Metrics*, ser. MetriSec ’10. New York, NY, USA: Association for Computing Machinery, 2010.
- [35] P. K. Manadhata and J. M. Wing, “An attack surface metric,” *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 371–386, May 2011.
- [36] M. Lanza and S. Ducasse, “Understanding software evolution using a combination of software visualization and software metrics,” in *In Proceedings of LMO 2002 (Langages et Modèles à Objets)*. Lavoisier, 2002, pp. 135–149.
- [37] T. Mens and S. Demeyer, “Future trends in software evolution metrics,” in *Proceedings of the 4th International Workshop on Principles of Software Evolution*, ser. IWPESE ’01. New York, NY, USA: Association for Computing Machinery, 2001, pp. 83–86.
- [38] N. Medeiros, N. Ivaki, P. Costa, and M. Vieira, “Software metrics as indicators of security vulnerabilities,” in *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*, Oct. 2017, pp. 216–227.