Examining Repudiation Threats Using a Framework for Teaching Security Design Analysis

Nikola Luburić[*], Goran Sladić[*], Branko Milosavljević[*]
[*] Faculty of Technical Sciences, University of Novi Sad, Novi Sad, Serbia
{nikola.luburic, sladicg, mbranko}@uns.ac.rs

***Abstract*—Secure software engineering is quickly becoming the standard for software development, due to the ever-increasing number of threats and attacks to software systems. While practices such as secure coding and testing can be achieved through automated tools, security requirements engineering, and secure design are fields which heavily rely on the security expertise of software engineers. Unfortunately, this is a skill set that is both difficult to teach and learn.**

Recently, a framework for teaching security design analysis was developed, based on case study analysis and the hybrid flipped classroom. This paper builds on that work and presents an application of our framework, where we construct a laboratory exercise dedicated to teaching the security design analysis for repudiation threats. Through this work, we provide additional guidance for the usage of our teaching framework and outline a laboratory exercise, which can be used as part of a university course or a workshop in a corporate training program.

I. Introduction

The Information Age is a name used to describe the current stage in the evolution of the civilized world. It represents a knowledge-based society surrounded by software systems that enable a global economy and intertwine government and business operations, as well as everyday life, to increase overall efficiency and convenience [1]. With such transformation, threats to society that were present in the physical world, such as crime and terrorism, are increasingly moving to the cyberspace. The global interconnectedness, provided by the Internet and software, has enabled threat agents to perform attacks anonymously and from faraway parts of the world. Such attacks occur daily, costing the global economy billions of dollars each year [2].

Governments and businesses that wish to protect their users, intellectual property, and operations, are making the security of their software systems a top strategic priority [3]. This requirement propagates down the supply chain, where software vendors are required to engineer secure software. Recent years have seen the rise of the security development lifecycle (SDL), a comprehensive approach to secure software engineering that augments all parts of the software development lifecycle to make sure that security is being built into the software solution [4].

High-level software security requirements are often concerned with protecting business assets and are elicited from standards, regulations, and industry best practices. Such requirements are usually defined at a high level of abstraction, focused around protecting the security properties (i.e., confidentiality, integrity, availability) of sensitive assets (e.g., user credentials, PII, mission-critical system functions) [5]. Principal activities of the SDL called threat modeling and security design analysis (SDA), are concerned with processing high-level security requirements and deriving actionable, low-level security requirements that can be implemented and tested at the code level [6]. Through SDA, software engineers analyze how their system's design fulfills the high-level security requirements, and plan work items to increase the security posture of their system accordingly. This approach identifies vulnerabilities before they are introduced to the code when they are least expensive to fix [7].

A problem with SDA is its inherent complexity, where engineers performing SDA need to possess a combination of security knowledge and attacker-oriented thinking, referred to as the attacker or security mindset [8]. This security mindset is both difficult to learn and teach, limiting the efficient practice of SDA in the industry [6][9].

In our previous work [10], we presented a framework for teaching security design analysis, using a combination of the case study analysis technique and the hybrid flipped classroom. By utilizing our framework, we constructed laboratory exercises for a course dedicated to secure software engineering, with a focus on SDA. The resulting labs consist of preparatory materials, where students learn about various software security design patterns and controls (e.g., secure communication, key management, authentication, and authorization) before attending the lab. During the lab, the focus is placed on learning how to apply the given patterns and controls to different systems (the case studies), through security design analysis. Our experimental results showed that labs constructed following our framework provided better learning outcomes for SDA, compared to the traditional teaching approach.

One of the limitations of our approach in [10] is the complexity of using the framework itself. While it requires more investment to construct the preparatory materials, the biggest issue lies in coordinating the different parts of the lab design (i.e., the case study, the preparatory materials, and the learning outcomes) to construct a coherent, concise, and complete lab. While we provided a high-level demonstration of our framework application, more in-depth guidance is called for to understand how to utilize our framework.

This paper focuses on guiding the execution of the framework presented in [10] to help resolve the issue mentioned above. We illustrate the algorithm and thought process for constructing one lab dedicated to teaching security design analysis for Repudiation threats, defined as part of the STRIDE threat analysis methodology [6]. We discuss each step and offer insight into the intricacies of using our framework. Secondary contributions of this paper include an outline of a lab dedicated to teaching how to design logging controls and integrate them into a system to mitigate repudiation threats. While this lab can be used as part of a university course, it can also be realized as a workshop in a software vendor's training program.

The rest of the paper is structured as follows: In Section II, we provide the background needed to understand the presented work, including a brief overview of SDA and STRIDE, as well as the components of our framework and the usage process. Section III describes the framework application, where we define the preparatory materials and case study to be used for the lab. Here we also present the lab exercise, created as a result of the framework application, and illustrate the lab flow designed to achieve the specified learning outcome. Finally, Section IV concludes this paper with a discussion, offering additional insight and ideas for further research.

II. BACKGROUND

In this Section, we provide the reader with the necessary knowledge to grasp the content of this paper fully.

Section II-A outlines security design analysis and the STRIDE methodology in general, and places focus on Repudiation. In Section II-B, we present an overview of our teaching framework and highlight its components and usage process.

A. Security Design Analysis

Security design analysis (SDA), sometimes called threat modeling, is the practice of assessing the design of a (software) system and its ability to resist attacks from threat agents and protect the security properties (i.e., confidentiality, integrity, availability) of its sensitive assets [6][9][10].

The term *module* is used to describe the target of SDA, where a module can be anything from a software component to an enterprise system. The inputs for SDA include a set of design artifacts, such as data flow and deployment diagrams, that describe the module, as well as a set of high-level security requirements detailing the parts of the system that need explicit protection. The output of SDA is a list of work items (e.g., design changes, user stories, research spikes) that need to be completed to increase the security posture of the examined module. A prerequisite to successful SDA is that the input design artifacts are correct and that the team performing SDA has a clear understanding of the module they are analyzing.

In general, the software engineering team performing SDA needs to answer the following questions [6]:

- What are we building? – Define the scope of the examined module.
- What can go wrong? – Identify applicable threats and decompose them to determine attacks that can realize them and vulnerabilities that can be exploited.
- What are we going to do about that? – Plan and prioritize work items to resolve discovered vulnerabilities.
- Did we do a good enough job? – Examine the quality of the previous steps.

STRIDE [6] is a method for security design analysis that helps answer the second question, offering a taxonomy of threats to guide threat identification. STRIDE is an acronym for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege, where each threat represents a class of attacks that can compromise a system or its assets. When applied to data flows, STRIDE can be used to generate relevant threats (following the STRIDE-per-element or STRIDE-per-interaction method [6]), which can then be decomposed to identify attacks and vulnerabilities.

We examine repudiation threats in more detail, as they are the primary learning outcome of the lab presented in this paper. Repudiation is the denial of action or the denial of inaction by a user. The term *user* throughout the rest of this paper includes human users, services, devices, and any entity that takes part in an action (e.g., sends or receives a message, calls a function) [11]. Repudiation is claiming that a button was not clicked (when it was) or that a message was received (when it was not). Repudiation can be deliberate (e.g., when the user wishes to deceive) or it can be accidental (e.g., when the UI is poorly designed, or a system error occurs causing the user to believe an action did or did not happen when the opposite is the case) [6].

B. Framework for Teaching Security Design Analysis

In [10], we presented a framework that utilizes the case study analysis method, and the hybrid flipped classroom to generate lab exercises dedicated to teaching SDA. Our framework consists of the following parts:

- The SDA method that is the learning objective.
- One or more case studies which describe a software system and that are modules for SDA.
- The preparatory materials that describe security concepts (i.e., attacks, vulnerabilities, mitigations) and which the trainees need to examine before the lab.
- The labs themselves, as the output of the framework execution, which describe how to apply SDA on the selected case studies, utilizing the knowledge provided by the preparatory materials.

Once the SDA method is selected, it is decomposed into aspects, where each aspect is the learning objective of one or more laboratory exercises. STRIDE, for example, can be decomposed into six aspects, one for each letter.

Each aspect is analyzed to define relevant security concepts (i.e., attacks, vulnerabilities, mitigations), for which preparatory materials are created. These materials can take any form, from text constructed for the lab to publicly available resources (e.g., videos, blogs).

Considering both the SDA aspects and the identified security concepts, requirements are defined for the case study, so that a relevant case study is selected for SDA analysis. For the example of repudiation, a suitable case study should include sensitive functions and data, where an essential requirement is to audit access to the data and calls to the functions.

The final step of framework execution entails the merger of all previous work (the SDA aspect, the preparatory materials, and the case study) into a lab exercise, where the flow of the lab is defined. The general flow requires the trainees to go over the preparatory materials before attending the lab. During the lab, they use this knowledge to perform SDA on the selected case study, guided by the trainer.

Following the hybrid-flipped classroom approach, we found that students had less trouble understanding specific security controls and attacks. On the other hand, they had more difficulty identifying when and where to invoke the control in the software's design. For these reasons, we offloaded the easier subject matter to the preparatory materials, while putting more emphasis on growing the security mindset during the labs.

The complete framework presented in [10] is more complex and entails additional steps not mentioned here, as it is designed to produce multiple lab exercises to cover the whole SDA method. For this paper, we take a simplified look at our framework, as only one laboratory exercise is produced comprising the Repudiation aspect of STRIDE.

III. Framework Application for Teaching Repudiation Threat Analysis

In this Section, we present an application of the teaching framework described in [10], to offer guidance for its use. We construct a lab exercise that tackles repudiation threats and mechanisms for their mitigation in software systems.

Section III-A describes the security concepts (i.e., attacks, vulnerabilities, mitigations) relevant for handling repudiation threats and presents the preparatory materials for the lab. Section III-B discusses the case study requirements and describes a suitable case study. Finally, in Section III-C, we show the flow of the laboratory exercise.

A. Preparatory Materials

The main vulnerability that enables repudiation threats in software systems is missing or poorly design logging mechanisms [6]. Log files contain entries that track the events of a system. On the one hand, they offer insight into a system's (mis)behavior, aiding software engineers in debugging issues. On the other hand, they offer non-repudiation, by recording user actions. While the concept of an event logger is simple, correctly implementing logging controls throughout the system to achieve non-repudiation can be difficult [11].

Recently, the IEC organization has released a standard describing technical security requirements for industrial automation and control systems [11], detailing many security controls and component requirements (CR), including a logging mechanism for non-repudiation. We use this document as a basis for our preparatory materials and from it derive the following requirements for our logging mechanism (the related requirements from the standard are noted in the braces):

1. Completeness – Each log entry needs to contain enough data to prove non-repudiation of an action (CR 2.8) and each event for which non-repudiation is required needs to be logged (CR 2.12).
2. Reliability – Logging needs to be reliable, which is achieved by ensuring the availability of the mechanism (CR 2.9, CR 2.10) and integrity of the log files (CR 3.9, CR 6.1).
3. Accuracy – Log entries across the system need to state their creation time precisely (CR 2.11).

Apart from the requirements derived directly from the standard, we add two requirements that improve the efficiency of the logging mechanism:

4. Usability – The logging mechanism needs to be designed so that security-relevant events (e.g., those that provide non-repudiation) can be easily extracted from the log files.
5. Minimalism – The logging mechanism should create the minimal amount of log entries needed to serve its purpose, to avoid cluttering the log files.

As log files contain system events that are used primarily for debugging, we need to make sure that security events are not buried and lost due to a large amount of non-security events.

Based on these requirements, we construct a three-page white paper to serve as preparatory materials for the lab. The document explains the danger of repudiation, illustrates it through real-world examples and describes the motivation behind it. The paper concludes by explaining event logging and details the requirements for an efficient and secure logging mechanism.

B. Case Study

Audit records need to be generated for access control, request errors, critical system events, backup and restore events, configuration changes, and audit log events, as noted in [11], CR 2.8. Furthermore, CR 2.10 defines additional activities that require logging, including performing system actions, creating or changing information, and sending messages.

Based on this list, we conclude that any software system that interacts with human users and has some sensitive assets can be used as a case study. As the SDA aspect and relevant security concepts do not impose significant limitations for our case study selection, we look to find a case study that is familiar to the audience that will be attending the lab. In our case, the lab is conducted as part of a fourth-year undergraduate course on the topic of secure software engineering. For this context, we choose the software system of a software vendor as our case study.

The information system of a software vendor contains a wide array of sensitive assets, including intellectual property (e.g., source code, design documents), data (e.g., employee PII, financial data, customer correspondence), and mission-critical systems (e.g., source code repositories, testing servers). Software vendors can be targets of sophisticated attackers, including cybercriminals looking to steal intellectual property and sensitive data and corporate espionage aiming to take intellectual property and sabotage the vendor's systems. Perhaps the most significant threat is posed by disgruntled employees, as they have internal access to the system and software engineering skills.

For these reasons, the standard on secure software engineering issued several requirements for securing the environment in which the software is developed [4], including requirements for non-repudiation.

C. Lab Flow

The trainees are required to go over the white paper describing repudiation and logging before attending the lab. At the start of the lab, the trainer conducts a brief discussion with the trainees to summarize the main points of the preparatory materials.

After the initial recap, the trainer presents the case study, introducing ACME corporation as a software vendor that produces software for industrial automation and control systems. The context of the vendor is given, the different software systems used by its employees, as well as the critical assets that need to be protected. The trainer takes care to introduce the main points of the system that need to be protected, without making them obvious. This information is masked with irrelevant information and low priority assets. However, care is taken not to bloat the presentation too much, to avoid loss of interest from the trainees. The presentation concludes with data flow and deployment diagrams of the ACME corporation, as they offer a view of the module suitable for security analysis.

Once the case study is presented, a discussion takes place to examine the security considerations for the given system. It is guided by the trainer and is an excellent opportunity to repeat course materials from previous labs or courses, especially if the presented case study is new and has not been examined during previous labs.

Ultimately, the discussion arrives at repudiation threats. The trainees examine the ACME system and try to find actions that a user might have reason to rebut. They identify interfaces between the human users and the software and discuss where and how the actions need to be logged. The goal of this discussion is to fulfill the *Completeness* requirement of the logging mechanisms, as well as obtain an understanding that logs can be generated at different levels of the software system (e.g., operating system, web server, application software).

Once most of the system events requiring non-repudiation have been mapped, the trainees expand the data flow diagrams with log data stores. At this point, the trainer directs the discussion towards the *Reliability* requirement, examining how the logging mechanism can be protected from tampering and denial of service. Scenarios that detail attacks are discussed, and the trainees determine appropriate security controls and design changes to protect the logging mechanisms.

The trainer addresses the final security requirement, *Accuracy*, by explaining how the network time protocol and GPS time synchronization protocols [12] can be used to create system-wide time synchronization. The design of ACME's system is expanded with these controls, and their security is discussed.

Finally, the software engineering requirements of *Usability* and *Minimalism* are addressed. The trainer divides the trainees into teams and asks them to design an application logging mechanism that can answer the following user stories:

"As a data protection officer, I want to quickly examine all access requests to GDPR [13] related data, so that I can examine if there is an anomaly in the system's behavior."

"As a reliability engineer, I want to quickly examine all mission-critical function calls, so that I can monitor performance to prevent a denial of service."

"As a software engineer, I want to examine log entries when an error occurs in a system, so that I can triage the bug and resolve the issue."

At the end of this exercise, each team presents their design and argues how it can fulfill the listed user stories. All trainees take this opportunity to discuss the pros and cons of each approach.

At the end of the lab, the trainer summarizes the main learning points of the lab and offers additional exercises and reading materials. The flow of the lab is illustrated in Figure 1, where the arrows originating from the trainees and trainer signify whether the user is the driver of an activity.

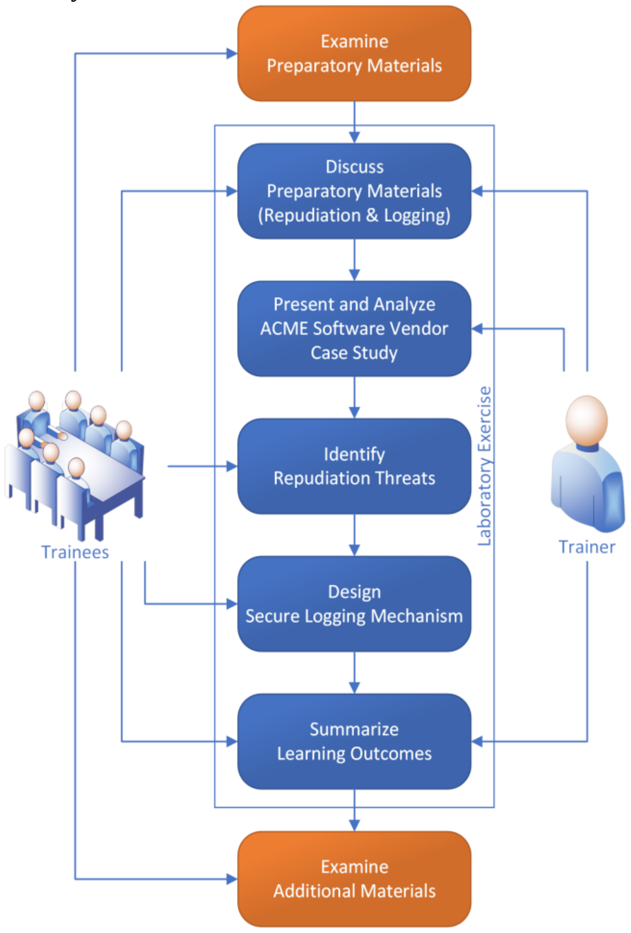
Examine Preparatory Materials
Discuss Preparatory Materials (Repudiation & Logging)
Present and Analyze ACME Software Vendor Case Study
Identify Repudiation Threats
Design Secure Logging Mechanism
Summarize Learning Outcomes
Examine Additional Materials
Laboratory Exercise
Trainees
Trainer

Figure 1. Flow of laboratory exercise dedicated to security design analysis for repudiation threats

IV. Conclusion

In this paper, we demonstrated the application of our framework for teaching security design analysis [10], to offer low-level guidance for the framework's use.

We focused on the Repudiation threat, one of the six aspects of the STRIDE threat analysis method, selecting it as the learning goal of the lab. By examining a recent industry standard [11], we identified requirements for security controls that mitigate repudiation threats. We constructed the preparatory materials guided by this source. Next, we selected a suitable case study for security design analysis, considering the trainees' familiarity with the chosen software system. Finally, we merged all the materials to design a laboratory exercise that utilizes the case study analysis, and hybrid flipped classroom teaching approach.

The presented lab, along with the outlines of the preparatory materials and case study, can be used to construct a lab for a university course and is suitable for a workshop held as part of a software vendor's training program.

Exploring new teaching methods that increase the quality of the learning outcomes is a never-ending task. When it comes to secure software engineering, there is a clear need to advance the security expertise of software engineers, to combat the growing threat of cyberattackers. With our teaching framework, we aim to tackle this issue and provide an efficient way for software engineers to learn about security design analysis.

Further work includes exploring alternative teaching approaches, such as gamification or e-learning, to see if they increase the quality of the learning outcomes. Furthermore, we need to determine the appropriate balance between the burden of examining preparatory materials and the learning value derived from them.

References

[1] Castells, M., 1996. The Information Age: Economy, Society and Culture. Volume I. The rise of the network society.

[2] Tarter, A., 2017. Importance of Cyber Security. In Community Policing-A European Perspective (pp. 213-230). Springer, Cham.

[3] James, L., 2018. Making cyber-security a strategic business priority. Network Security, 2018(5), pp.6-8.

[4] International Electrotechnical Commission (IEC). 2018. 62443-4-1: Security for industrial automation and control systems, part 4-1: Product security development life-cycle requirements. USA.

[5] Luburić, N., Sladić, G., Milosavljević, B. and Kaplar, A., 2018. Demonstrating Enterprise System Security Using an Asset-Centric Security Assurance Framework. In International Conference on Information Society and Technology (ICIST 2018).

[6] Shostack, A., 2014. Threat modeling: Designing for security. John Wiley & Sons.

[7] Security Innovation Europe, The Business Case for Security in the SDLC, source: cdn2.hubspot.net/hub/355303/file-559719186-pdf/whitepapers/business-case-appsec.pdf?t=1471855549672, retrieved: 5.12.2018.

[8] Mansfield-Devine, S., 2017. Threat hunting: assuming the worst to strengthen resilience. Network Security, 2017(5), pp.13-17.

[9] Brook SE Schoenfield. 2015. Securing systems: Applied security architecture and threat models. CRC Press.

[10] Luburić, N., Sladić, G., Slivka, J. and Milosavljević, B., 2019. A Framework for Teaching Security Design Analysis Using Case Studies and the Hybrid Flipped Classroom. ACM Transactions on Computing Education (TOCE), 19(3), p.21.

[11] International Electrotechnical Commission (IEC). 2018. 62443-4-2: Security for industrial automation and control systems, part 4-2: Technical security requirements for IACS components. USA.

[12] Mills, D.L., 2016. Computer network time synchronization: the network time protocol on earth and in space. CRC Press

[13] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation).