Utilizing a Vulnerable Software Package to Teach Software 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* – As the number of threats and attacks to software systems increases, more attention is given to secure software engineering practices, such as secure coding and security testing. More abstract activities, such as security design analysis, require extensive security expertise from software engineers. Unfortunately, such knowledge is scarcely available, as it is an area that is both difficult to teach and learn.**

We developed a framework for teaching security design analysis, which is built around the hybrid flipped classroom and case study analysis. This paper enhances our framework by utilizing freely available vulnerable software packages as case studies for security design analysis. We illustrate the enhancement by using a mature vulnerable software package to construct a laboratory exercise dedicated to the security design analysis of threats originating from injection-based attacks. We provide guidance for the usage of our enhanced framework and outline a lab that can be utilized for a university course or a corporate training program dedicated to secure software engineering.

Keywords – security design analysis; threat modeling; vulnerable software package; education; secure software engineering

I. INTRODUCTION

As much of society is becoming digitalized, malicious agents interested in war, terrorism, and crime are moving their sights to the cyberspace. Motivated by profit, political gains, ideology, or malice, these individuals and organizations harm society by sabotaging industrial control systems, stealing data from information systems, and hampering individuals in their everyday lives [1].

Attacks on software systems are prevalent and are increasingly the subject of news headlines, scientific research, and discussions regarding business investments [2]. As a result, businesses have become more aware of cyber threats and are making the security of their software systems a strategic priority. A recent study surveyed 31 large corporations and showed that 81% of the surveyed businesses had upper-level management that was supportive of cybersecurity efforts conducted in the firm [3].

To fulfill the requirements for increased security, software vendors are introducing a set of security tools and activities to their software development lifecycle. This set is often referred to as the security development lifecycle (SDL) [4], and it represents a way to systematically address security concerns from the software's inception and design, down to the implementation and deployment.

Software security requirements are usually derived from relevant industry standards, regulations, and best practice documents. As these requirements are centered around assets and can be ambiguous to software developers [5], an essential activity in the SDL is security design analysis (SDA), which is sometimes called threat modeling [6]. By performing SDA, software engineers examine how the software's design fulfills the high-level security requirements, and plan work items to better fulfill the requirements. This activity discovers security issues before they are coded and when they are least expensive to fix [7].

An applicability issue with SDA stems from its complexity, where engineers need to maintain security knowledge and practice attacker-oriented thinking. This security mindset is difficult to learn and teach, hampering the efficient application of SDA in the industry [6][8].

In our previous work [9], we presented a framework for teaching security design analysis, using a combination of the case study analysis technique and the hybrid flipped classroom. Through our framework, we created laboratory exercises for a course dedicated to secure software engineering, with a focus on SDA. Our experimental results showed that labs constructed following our framework provided better learning outcomes for SDA, compared to the traditional teaching method. A significant limitation of our approach is its complexity, where additional effort is required to create the lab exercises when compared to the conventional classroom. In subsequent work [10] we tackled this limitation by offering low-level guidance for the efficient use of our framework. In [10], we illustrated the execution of our framework to create a laboratory exercise dedicated to finding and resolving repudiation threats.

This paper expands on the work presented in [9] and [10] by further reducing the complexity limitation of the SDA teaching framework. In this paper, we examine how freely available vulnerable software packages, dedicated to teaching software security, can be integrated with our SDA teaching framework. To illustrate this, we utilize our enhanced framework to generate a lab using a well-known and mature vulnerable software package. We demonstrate how the OWASP Juice Shop [11] vulnerable web

application can be used to simplify the process of developing labs, increase their overall quality of the lab, and reduce the complexity of using our framework. Through this demonstration, we offer secondary contributions through this paper, which include further guidance on how to use our SDA framework and an outline of a lab dedicated to identifying and mitigating threats caused by injection attacks. 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 organized as follows: Section II provides an overview of SDA and presents our framework for teaching it. Here we also discuss vulnerable software packages dedicated to educating on software security and examine their use in the classroom. In Section III, we present our enhanced SDA teaching framework and illustrate how it integrates with vulnerable software packages. Here we incorporate the Juice Shop project into our framework and apply the enhanced framework to generate a laboratory exercise dedicated to examining injection attacks and defenses. In Section IV, we discuss our enhanced framework and note the limitations of its use. Section V lists related work, where we examine courses and labs that have utilized vulnerable software packages in the classroom and compare them with our work. Finally, in Section VI, we conclude our work and present directions for further research.

II. BACKGROUND

The following Section presents the concepts and necessary background required to understand our work. Section II-A provides a brief overview of security design analysis (SDA), focusing on the activity's purpose, its inputs, and outputs. In Section II-B, we present our framework for teaching SDA, explaining its components and workflow. Finally, in Section II-C, we examine different vulnerable software packages and explore their use in the classroom.

A. Security Design Analysis

SDA is an activity for assessing the design of a software module 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]. A module in this context can be a software application component, a complete application, or an enterprise system.

The inputs for SDA consist of the module's design artifacts (e.g., data flow diagrams) and high-level security requirements. 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. For successful SDA, the input design artifacts need to be correct, where the team performing SDA has a clear understanding of the module they are analyzing.

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

- What is being built? – The scope of the module.
- What are the security issues? – The applicable threats and attacks that realize them, as well as the module's vulnerabilities that can be exploited.
- What will be done about it? – The work items to resolve discovered vulnerabilities.
- Did we do a good enough job? – A retrospective regarding the quality of the previous steps.

B. Teaching Framework for SDA

In [9] and [10], we demonstrated a framework that combines the hybrid flipped classroom with the case study analysis method to generate lab exercises dedicated to teaching SDA. The framework components include:

1. The SDA method which is the learning objective.
2. Preparatory materials describing security concepts (i.e., attacks, vulnerabilities, countermeasures), which the trainees examine before the lab.
3. Case studies, representing modules under analysis.
4. The labs which define how to apply SDA on the case studies, using the knowledge from the preparatory materials.

The selected SDA method (1) is decomposed into aspects. An aspect is the learning objective of one or more laboratory exercises. Each aspect is analyzed to define relevant security concepts (i.e., attacks, vulnerabilities, countermeasures), for which preparatory materials (2) are created. The materials can take the form of textbook chapters, public articles, videos, etc.

Based on the SDA aspects and related security concepts, the requirements for the case study (3) are elicited. The goal is to construct a relevant case study for SDA analysis. For example, when examining injection attacks [12], an essential requirement for the analyzed case study entails the use of command interpreters that handle externally-supplied input (e.g., SQL database receiving SQL queries, XML parsers receiving XML documents, LDAP interpreters for LDAP queries).

The final step combines all the previous work into a lab exercise (4). As demonstrated in [10], the general lab flow entails that the trainees go over the preparatory materials before attending the lab so that they can use this knowledge during the lab. At the start of the lab, the trainer discusses the preparatory materials with the trainees, after which the case study is presented. The trainees, guided by the trainer, perform SDA on the case study, and additional assignments are defined for the trainees to achieve successful learning outcomes. At the end of the lab, the participants conduct a retrospective of the lab, and the trainer presents any additional materials that the trainees can examine after the lab.

We have evaluated and successfully applied our teaching framework at our university, to develop laboratory exercises for an undergraduate course covering secure software engineering, with a focus on SDA. One limitation of the approach detailed in [9] and partially addressed in [10] is the complexity of the framework

when compared to the conventional classroom. To create a lab, the trainer needs to build a suitable case study, construct the preparatory materials, and plan the lab flow to determine assignments that bring everything together to achieve better learning outcomes.

C. Vulnerable Software Packages

Vulnerable software packages offer software engineers, security auditors, and penetration testers a playground to practice software security skills, both from the attacker's and defender's perspective and for this reason, are often used in training programs [13][14]. Additionally, these packages are used to test the efficiency of hacking tools, such as vulnerability scanners [15] as well as security controls which mitigate the vulnerabilities of the software package [16].

Due to a large number of vulnerable software packages, several attempts have been made to classify them and offer guidance on how best to utilize them. In [13], the authors review resources in secure software engineering education, listing and examining different hands-on exercises provided by these software packages and discussing a roadmap on how to best integrate them into the classroom. Next, a dedicated OWASP project was set up in 2013, providing and maintaining a directory of vulnerable software packages [14], which includes over sixty packages, as of January 2019.

These packages vary in their maturity, from basic examples demonstrating a few attacks and vulnerabilities, to sophisticated systems containing a vast plethora of teaching assignments. Furthermore, they differ on the security concepts that they cover, as well as the underlying technology on which they are built. Out of all the available options, we selected the OWASP Juice Shop Project [11] to integrate with our framework, due to the following reasons:

- It is mature, categorized as a Flagship project by the OWASP organization, signifying its value to the field of application security. Furthermore, it has previously been utilized in the classroom [11][13], as a tool for testing the efficiency of a hacking tool [15] and a security defense [16].
- It is rich with content, covering a wide array of attacks and defenses and containing 74 challenges as of version 8.3, released in January 2019.
- It is easy to use, offering detailed documentation, presentation, and video material to aid with its use and a companion guide which details each challenge and its solution [11].
- It is built on a technological stack (Angular, Node.js, SQLite) familiar to our students.
- It is a fully functioning web shop, offering browsing and shopping functionalities similar to applications which the students built on earlier courses.

III. Enhancing the Teaching Framework with Vulnerable Software Packages

In this Section, we propose an enhancement to the teaching framework presented in [9], by utilizing publicly available vulnerable software packages (VSP) to offload some of the work required from the lab constructor. Namely, we propose that the vulnerable software package can be used as a case study and the target of security design analysis. Once requirements for the case study are gathered, the lab constructor searches for a suitable VSP instead of manually constructing a case study. Furthermore, when determining the lab flow, assignments for the lab trainees can be derived directly from the features of the VSP.

To illustrate the enhancement to our framework and provide low-level guidance, we demonstrate its usage by constructing a lab with the learning objective of examining and mitigating injection attacks and vulnerabilities [12].

The preparatory materials for this lab are created by utilizing the latest OWASP Top Ten list [12]. This list is an authoritative document that presents a broad consensus regarding the most common and most critical security risks for web applications. It is updated and published every several years to keep up with the shifting threat landscape.

The OWASP Top Ten list provides a thorough overview of each class of security risks, offering insight into the risks impact, examples of attacks, pointers for vulnerability discovery and mitigation planning. For our context, trainees are meant to go over the following categories before attending the lab:

- A1 – Injection, as the primary subject matter of the lab.
- A4 – XML External Entity, due to its similarities with XML injection attacks from A1.
- A7 – Cross-Site Scripting, which can be seen as an injection attack aimed at the browser's command interpreter.

Furthermore, trainees are required to find and examine at least one example of each of the listed attacks (e.g., SQL injection, LDAP injection, Stored XSS) to get a sense of what the attack vectors look like and how they might be deployed.

Guided by the learning objective and preparatory materials, we go over the list of different attacks and vulnerabilities and define the following requirements for a suitable case study:

- It should provide a web user interface, to demonstrate cross-site scripting issues.
- It should have an SQL database where at least one command sent from the application is dependent on user-supplied input, to explain SQL injection issues.
- It should process XML documents supplied by external entities, to demonstrate XML injection issues.

- (Optional) It should provide functionality suitable for showing additional injection issues (e.g., OS command injection, LDAP query injection).
- (Optional) It should be built using modern technologies, ideally those directly utilized by the trainees, to increase the perceived relevance and their engagement.

As noted in Section II-C, we selected the OWASP Juice Shop [11] VSP as a suitable case study that fulfills both the mandatory and optional requirements listed above.

The lab constructor utilizes the companion guide to select challenges related to the learning objective and marks them as assignments for the lab. With these assignments, the final lab flow can be created, as illustrated in Figure 1.

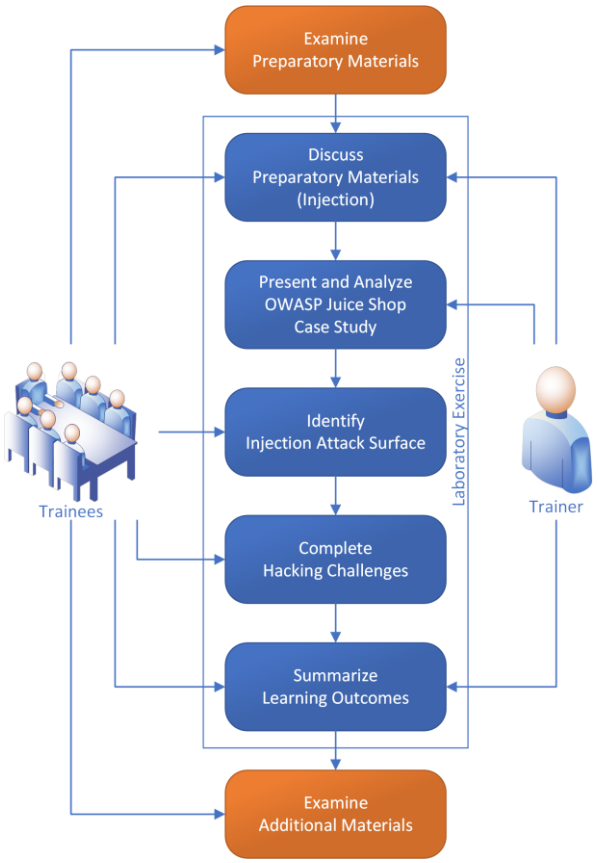

Figure 1. Lab flow for security design analysis of injection issues

At the start of the lab, the trainer goes over the different injection attacks and answers any questions which the trainees might have.

Next, the trainer introduces the OWASP Juice Shop and demonstrates the basic functionality of the application, after which the architecture and data flow diagrams of the module are presented (available at [11]).

The first assignment for the trainees is to identify the attack surface where injection attacks might be deployed, as part of a group discussion.

Once much of the attack surface is discovered, the trainees are given a list of challenges from the companion guide to complete on the laboratory computers.

At the end of the lab, the trainer highlights the learning objective, summarizes the activities conducted during the lab and notes the crucial takeaways. During this discussion, the participants:

- Define the impact of the attacks they performed;
- Determine which vulnerabilities exist in the software to allow those attacks to succeed;
- Specify mitigations which resolve the vulnerabilities and discuss ways in which the mitigations can be circumvented.

IV. Discussion

In this Section, we examine the benefits that the presented enhancement brings to our framework, describe the evaluation of the enhancement, and list the limitations that prevent the enhancement from being utilized in all cases.

By integrating a vulnerable software package (VSP) into our lab, we drastically reduced the amount of time it takes to prepare a lab using our framework. Initially, before enhancing our framework, the lab constructor needed to invest significant time (approximately a week) to build the case study, the assignments and related code samples for the lab covering the topics of injection attacks. With the Juice Shop application, the build time was reduced to a single day, which included the time it took to find and explore the VSP.

Furthermore, by introducing a mature VSP to our lab, we have increased its quality and the quality of its learning outcomes. Examples of isolated, vulnerable code were replaced by a fully fleshed out web application with vulnerabilities hidden across its various functions. The result is that the trainees can fully grasp the impact of the different vulnerabilities and attacks, which is more difficult when presented on a code sample that exists in a vacuum.

We conducted an informal evaluation of our enhancement by analyzing observations of the trainers and conducting semi-structured interviews with the trainees. During the exercise, the trainers noted a higher than expected level of engagement from all trainees. Concretely, the VSP with its challenge system facilitated a competitive atmosphere, where trainees were rushing to complete the next challenge before their colleagues. For more difficult challenges, the trainees gathered in smaller groups to brainstorm solutions, maintaining the competitive spirit by contesting other groups. Near the end of the lab, the participants summarized the learning outcomes, where the trainer noted that the trainees clearly understood the impact injection attacks could have on the software, why they occur and how to conduct them. Finally, the trainer facilitated a discussion around the quality of the lab and the use of the VSP. The consensus was that the lab was fun and engaging, where trainees made requests for similar labs in the future.

The main limitation of our enhancement is the relative shortage of VSPs. While web technologies and common web security issues [12] are covered by several high-quality VSPs [14], domains such as mobile or embedded application are scarcely covered. While basic code samples and toy projects can be found, sophisticated solutions reaching the quality of the OWASP Juice Shop project are few and far between.

V. Related Work

In this Section, we examine other teaching approaches which utilize a vulnerable software package (VSP) and discuss how their approach differs from ours.

In [17], the authors present BREW, a VSP designed to teach trainees how to find and exploit vulnerabilities as an attacker and to subsequently identify the issues in the code and resolve them as a defender. The authors define different educational usage scenarios, offering guidance on how to integrate BREW into different classrooms. Furthermore, they list settings where BREW has been successfully integrated into lectures and lab exercise.

Compared to the OWASP Juice Shop, BREW covers a broader scope by addressing both the hacker (attacker) and the developer (defender) perspective. While our lab design addresses the defender perspective with the discussion held at the end of the exercise, the ability to mitigate vulnerable code after exploiting it would improve our exercise. However, BREW has several disadvantages compared to the OWASP Juice Shop. Most importantly, its code repository has not been updated for five years, there is a lack of documentation regarding installation and deployment, and the collection of challenges is significantly smaller.

In [18], the authors utilized the OWASP WebGoat VSP to teach web security attacks and vulnerabilities, focusing on SQL injection attacks. They created several labs, where they attacked WebGoat manually and with the aid of security testing tools.

The OWASP WebGoat is a maintained VSP, currently standing as a medium-level project in the OWASP organization. While it provides an impressive array of challenges, it is less developed than the OWASP Juice Shop. Most significantly, its challenges are presented as a collection of vulnerabilities “in a vacuum,” meaning that unlike Juice Shop, they are not integrated into a setting that makes it appear as a real application, but rather a collection of tasks.

VI. Conclusion

Secure software engineering is a growing field that is faced with significant obstacles, from insufficient budgets to lack of expertise. Due to the continually rising number of cyber threat agents and their economic impact on the civilized world, it is a field that requires much attention. With our teaching framework, we strive to thwart cyberattackers, by providing an efficient way for software engineers to learn about security design analysis.

In this paper, we presented an enhancement of our framework for teaching security design analysis, by introducing vulnerable software packages to the lab. Vulnerable software packages enable trainers to demonstrate security issues in a real-world context and allow trainees to apply their security knowledge to both perform attacks on and build defenses in a real-world software system. Thus, vulnerable software packages serve the first principles of instruction [19] to increase trainee learning.

By utilizing these tools, we have reduced the effort it takes to develop a laboratory exercise through our framework. Additionally, we have increased the overall quality of the lab by replacing a case study description and vulnerable code samples with a fully functioning application that contains vulnerable code.

Concretely, we have demonstrated how the OWASP Juice Shop vulnerable software package can be integrated into a lab, generated using our framework, that covers the topic of injection attacks and defenses. The Juice Shop application is the case study of the lab and offers challenges which are mapped to assignments for the lab. An outline of the injection lab is given, explaining how the Juice Shop is utilized. The presented lab can be used for a university course or any training program for software engineers.

As part of our further work, we plan to explore gamification to see if it can increase the quality of the learning outcomes. Secure software engineering workshops are often created around hackathons and capture the flag events, and we plan to examine and evaluate this approach. As some vulnerable software packages, such as the OWASP Juice Shop, provide capture the flag infrastructure out of the box, we feel that our framework can be further expanded to utilize this teaching approach.

References

[1] Uma, M. and Padmavathi, G., 2013. A Survey on Various Cyber Attacks and their Classification. IJ Network Security, 15(5), pp.390-396.

[2] Page, J., Kaur, M. and Waters, E., 2017. Directors’ liability survey: Cyber attacks and data loss—a growing concern. Journal of Data Protection & Privacy, 1(2), pp.173-182.

[3] Moore, T., Dynes, S. and Chang, F.R., 2015. Identifying how firms manage cybersecurity investment. Southern Methodist University. Source: blog.smu.edu/research/files/2015/10/SMU-IBM.pdf. Retrieved: 19.1.2019.

[4] Howard, M. and Lipner, S., 2006. The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software (Developer Best Practices).

[5] Ransome, J. and Misra, A., 2018. Core software security: Security at the source. CRC press.

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

[9] 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.

[10] Luburić, N., Sladić, G., Milosavljević, B. 2019. Examining Repudiation Threats Using a Framework for Teaching Security Design Analysis. In International Conference on Information Society and Technology (ICIST 2019).

[11] Kimminich, B., OWASP Juice Shop Project, https://www.owasp.org/index.php/OWASP_Juice_Shop_Project, retrieved: 19.1.2019.

[12] Stock, A.V.D., Glas, B., Smithline, N. and Gigler, T., 2017. OWASP Top 10 2017. The Ten Most Critical Web Application Security Risks.

[13] Yuan, X., Yang, L., Jones, B., Yu, H. and Chu, B.T., 2016. Secure software engineering education: Knowledge area, curriculum and resources. Journal of Cybersecurity Education, Research and Practice, 2016(1), p.3.

[14] Siles, R., Bennetts, S., OWASP Vulnerable Web Application Directory Project, https://www.owasp.org/index.php/ OWASP_Vulnerable_Web_Applications_Directory_Project, retrieved: 24.1.2019.

[15] Esposito, D., Rennhard, M., Ruf, L. and Wagner, A., 2018. Exploiting the potential of web application vulnerability scanning. In ICIMP 2018, Spain, July 22-26, 2018 (pp. 22-29). IARIA.

[16] Pupo, A.L.S., Nicolay, J. and Boix, E.G., 2018, September. GUARDIA: specification and enforcement of javascript security policies without VM modifications. In Proceedings of the 15th International Conference on Managed Languages & Runtimes (p. 17). ACM.

[17] Pohl, C., Schlierkamp, K. and Hof, H.J., 2015. BREW: A Breakable Web Application for IT-Security Classroom Use. arXiv preprint arXiv:1506.03325.

[18] Walden, J., 2008, October. Integrating web application security into the IT curriculum. In Proceedings of the 9th ACM SIGITE conference on Information technology education (pp. 187-192). ACM.

[19] Merrill, M.D., 2002. First principles of instruction. Educational technology research and development, 50(3), pp.43-59.