CSC 515: Software Security Course Project

UPDATED – 4/13/2023 to clarify Phase Three deliverables.

Throughout the rest of the semester, you will work as a team to perform and document a technical security review of an open-source source system. Suggestions are provided at the end of this document.

You must work on the course project in teams of 2-3

As a team, you must decide which software system (from the list above) you will use for the entire course project. You cannot change software systems after phase 1. If you do not register a team by 11:00 PM on 3/7/2022, you will be placed into teams.

Project Milestones

The following table list the milestones for the project. The deliverables will be submitted in Gradescope.

Milestone	Deadline	Description
Team Registration	3/7/2023, 11:00 PM	List team members, open-source source system
Phase 1	3/28/2023, 11:00 PM	Security Requirements, security review
Phase 2	4/11/2023, 11:00 PM	STRIDE Threat Analysis, Attack / Defense Tree, Design Principles
Phase 3	4/24/2023, 1:00 PM	Risk-based security tests, OWASP Application Security Verification Standard, Usability

Each phase counts equally to the overall project grade

Phase 1

The initial project phase consists of four parts, each weighted 25% of the phase grade.

Part 1: OWASP Top Ten / Ethiical Hack

Perform a security review of your selected system using the 2021 OWASP Top Ten (https://owasp.org/Top10/). You should select five of the top ten items. Test your system to see if the application is vulnerable for each item.

If you find any vulnerabilities, document the necessary steps to replicate the vulnerability. If you do not find any vulnerabilities, document how you tested for each vulnerability and how the development team mitigated against such attacks. In addition to class slides, you should utilize the OWASP Top 10 materials on the OWASP website. You may perform this manually as well as using automated tools such as Zap and Burp (https://portswigger.net/burp/communitydownload). You may also use fuzz testers.

Part 2: Static Analysis

Run a static analysis tool against your selected system as well as the OWASP Dependency Checker. Some possible tools based on the language:

- Java: Find Bugs - http://find-sec-bugs.github.io/
- Python: Bandit - https://bandit.readthedocs.io/
- Node.js possibilities: https://geekflare.com/nodejs-security-scanner/

Review the generated security reports (from your selected tool and the OWASP Dependency Checker). Based upon the reports, create a prioritized list of five changes the systems developers should make to the system to correct the deficiencies found. For each change, document the change required, what weakness the change mitigates, and provide a cross-reference back to the originating report(s) were the issue was documented. Submit the generated reports in addition to your prioritized list of changes.

Part 3: Security Requirements

Develop a list of ten security requirements to the system. Map these requirements back to one o more of the security objects (e.g., confidentiality). Ensure the requirements are "SMART" (e.g., specific)

Part 4: Abuse / Misuse Diagram

Create an abuse/misuse diagram. Your diagram should have at least 2 attackers and four possible attacks. You must also complete the template presented in the requirements lecture for 1 of the attacks.

Phase 2: Threat Analysis and Design Principles

The first part is worth 34% and the other two parts are worth 33%

Part 1: Perform a STRIDE Threat Analysis

Create a data-flow diagram for your system. Create a list of at least 10 threats identified using STRIDE. Each threat should include the threat category, threat description, mitigation description, and DREAD score.

Part 2: Attack / Defense Trees

Develop an attack tree for part of your selected system. The attack tree should have at least three levels and ten "nodes".

Part 3: Design Principles

Consider which security design principles are relevant to your system. elect at least 3 relevant design principles described by Saltzer & Schroeder and 3 relevant design principles from IEEE. For each of the selected design principles, clearly identify the design principle then briefly explain how the design principle is relevant to promoting security.

Phase 3: Testing and Usability

Parts one and three are worth 40% each while part two is worth 20%.

Part 1: OWASP Application Security Verification Standard

Use the OWASP Application Security Verification Standard to determine whether your selected system meets the following requirements:

- V2.1.1
- V3.1.1
- V3.3.1
- V4.1.1
- V5.1.3
- V6.1.2
- V6.2.2
- V7.1.3
- V7.1.4
- V7.3.3

For each requirement, give a brief summary of how you checked the system, and clearly indicate whether the system meets or does not meet the requirement. If you are unable to confidently verify any of the requirements above, then you should still describe the steps you used and briefly indicate why you are not confident in your findings.

Part 2: Usability

Describe how the system either meets or does not meet each of Jakob Nielson's Ten Usability Heuristics. Additionally, find one warning message and one explanation and apply the NEAT / SPRUCE mnemonics to each. (NEAT for a warning message, SPRUCE for an explanation). You may use system documentation for this as well. List the warning message and how NEAT applies to the message. List the explanation and how SPRUCE applies to the explanation.

Part 3: Testing

Create **8** black box test cases to start a repeatable black box test plan for you Open Source System. You may find the OWASP Testing Guide (https://owasp.org/www-project-web-security-testing-guide/v42/) and OWASP Proactive Controls(https://owasp.org/www-project-proactive-controls/) helpful references in addition to the references provided throughout the ASVS document (https://owasp.org/www-project-application-security-verification-standard/).

For each test case, you must specify:

- A unique test case id that maps to the ASVS, sticking to Level 1 and Level 2. Provide the name/description of the ASVS control. Only one unique identifier is needed. The ASVS number should be part of the one unique identifier.
- Detailed and repeatable (the same steps could be done by anyone who reads the instructions) instructions for how to execute the test case
- Expected results when running the test case. A passing test case would indicate a secure system.

- Actual results of running the test case should be provided in a separate page at the end of your complete submission. As you will see, another team will run your test cases -- we will ask them to record their actual results as well.
- Indicate the CWE (number and name) for the vulnerability you are testing for [as found in the ASVS document].

In choosing your test cases, we are looking for you to demonstrate your understanding of the vulnerability and what it would take to stress the system to see if the vulnerability exists. You may have only one test case per ASVS control.

Open-source System

You are free to choose any open-source system for this project. Some criteria to consider when making this choice:

- Familiarity with the implementation language and deployment architecture
- Sufficient size such that there's enough source code and functionality to examine
- Small enough that the system is not overwhelming.

Possible choices: (you are not limited to one of these)

- OpenMRS: This open-source system provides electronic health care functionality for resource-constrained environments. While the system has not been designed for deployment within the United States, security and privacy concerns are still a paramount security concern for any patient.
- OpenEMR: This open-source system provides electronic health care functionality for several countries (including the United States). OpenEMR currently claims to be used to manage over 1 million patient records across the globe.
- Beancount/ Fava: double-entry bookkeeping software
- MifosX Community App: open-source financial platform
- Open Source Event Manager
- Zen-Cart: eCommerce web application
- Wagtail: Content management system in Python/Django.
- List of open-source applications: https://github.com/unicodeveloper/awesome-opensource-apps