R·I·T

Rochester Institute of Technology
Golisano College of Computing and Information Sciences
Department of Computing Security

Project 1 – Vulnerable Site Creation

Lab Information

Due Date:

Project 1 Dropbox Deadlines

Objectives/Goal:

You are charged with developing a web application with 4 preexisting vulnerabilities. This web application will demonstrate fluency in complex web application security notions across various platforms. Students will be required to introduce vulnerabilities in specific locations, while not inadvertently introducing other vulnerabilities. Your web application will be designed in accordance with modern development standards. This includes test driven development (TravisCI), DevOps technologies (like Docker), and agile-like methodologies (buzz words alert). Leveraging these students will demonstrate their familiarity with industry accepted development practices by using common development techniques.

Format

The project is designed to be completed in groups of three.

Deliverables:

- Dockerfiles and associated Docker-compose file
- Design Documents
- A zip of your repository
- A writeup that answers the questions outlined in the project

Table of Contents:

Activity 1: Setup GitHub

Story

As a developer I would like to layout my development path so that I can more successfully and quickly develop my application as a team.

Goal

This project is to be hosted, publicly, on GitHub. If you do not have a GitHub account you should set one up. This project will also require Travis CI. This tool is free when used with open source projects. Similar open source alternatives exist such as Drone (https://drone.io).

Use the activities below as guides to setup your project within Github. You are required to breakdown your repository along the following Agile-like elements: Epics, Sprints, and User Stories. An Epic typically represents a shippable piece of work. In this case you can imagine it as the major ‘chunks’ of the project. Typically such epics are broken down into one or more ‘*sprints’* of a fixed time (ex. 1 sprint is two weeks). Each sprint consists of several tasks, which are adequately explained and assigned *before* the work is undertaken. The aforementioned steps are broken down loosely according to the Agile methodology, if you are not familiar check out the following videos: https://www.youtube.com/watch?v=502ILHjX9EE (Agile Product Ownership in a Nutshell).

Generally, the most common way to provide for an Agile-like enviorvment on Github is to use ‘*Github Projects’* to represent ‘*Epics*’, ‘*Github Milestones*’ to represent ‘*Sprints*’, and ‘*Github Issues’* to represent ‘*User Stories*’.

Note: However you complete this task, your repository will be viewed at the culmination of the project to ensure that you leveraged the methodology. That is to say, points may be deducted from your Activity One/two score if you failed to close and update tickets as you complete other aspects of the project.

Questions

- What is the URL of your Github project?
- How did you breakup your projects and what are the security ramifications?
- How did you choose to break down your Epic into various issues (tasks)?
- How long did you assign each sprint to be?
- Did you deviate from the Agile methodology at all? If yes, what is your reasoning for this?
- How do you ensure that after each issue/milestone that security has been verified? How would you identify such issues in an ideal environment?

Activity 2: Generate Tests/Design (UML)

Story

As a developer I would like to have documentation about how to implement the application

As a developer I would like to use test-driven development in order to write my code, as a result test cases need to be generated.

Goal

Just like when designing a network it is important to have documentation about how a web application will act. Different teams will go about this in different ways. Unified Modeling Language is designed in order to aid with generalizing how designs and flows should work. The end goal of this activity is to have a set of design documentation that would allow someone to create your vision, even if you were not present (hit by a car for instance). You should use UML along with other mockups to allow your team to more easily complete the subsequent sections.

The output of this step should include at minimum Use Cases (*and* diagrams), Network Topology, Flow Graph(s), Database Schema(s), and State-diagram(s) for each part of the application (This may be done using tools like Visio or online tools like Draw.io). This will require you to understand, review, and consider the requirements for the subsequent activities. You are also required to prepare a site mockup. This may be done in a graphics tool like Photoshop or an online design tool like Invisionapp (https://www.invisionapp.com/)

In addition to this you should setup a testing service using TravisCI. For the remainder of the project activities we will use these tests will ensure that each requirement is fulfilled. **You will not be able to pass an activity without tests for that section.** For activity 2 we will setup a basic example to ensure this is working. For this step you'll need two different parts. A Docker container that starts up an Apache web server with a 'Hello World' index page (see HW1), and a unit test written in Python (I tend to like pytest but you may use whichever unit testing framework you like) that reaches to your containers website and checks to see if the page contains 'Hello World'. You will configure TravisCI to load your Docker container and then run your pytest tests against that instance

Backend constraints

- You should setup continuous integration that fires on each Pull Request, use Travis CI for this.
- You should have at least one example test case running, using an established testing framework, in python. You should use this unit testing framework for all subsequent test cases.
- You should mock up your page design and interaction.
- You should use UML to describe user interactions
- All design documentation should be available on your project's GitHub

Questions

- What Web Application security mechanisms are involved in your topology? What security mechanisms would ideally be involved?
- What testing framework did you choose and why?

Activity 3: Authentication (PHP/Python/Ruby/Go/NodeJS)

Story

As a developer I would like to provide the capability for users to log into my web application.

Note: No registration is needed (although you can add it), it is acceptable to manually add users via backend database manipulation.

Goals

Most web applications today have some sort of session capability tied to user accounts. You will write this functionality for your web application. You should come up with a page that prompts a user for a username and password combination. On submission the page will check in a database if that username and password are correct. If successful the user should be redirected to a protected landing page. This page should not be accessible if a user has not authenticated. If the user enters the wrong information, they should be prompted that their information is incorrect. Users that log in must be able to logout of your application. You are free to use PHP/Python/Ruby/Go/NodeJS for a backend language. You may use any frontend technologies you wish, HTML, JS, any JS frameworks (JQuery, Angular).

Backend constraints

- There must be a test case that logs into the application successfully
- There must be a test case that tries to log in with the wrong password and fails.
- There must be a test case that tries to log in with the wrong username and fails.
- Information regarding account information should be stored in a MySQL/MariaDB instance.
- You should host both your database and your web frontend as, separate, Docker containers.
- Ensure that your provided Dockerfiles and Docker-compose file still function correctly

Questions

- Provide a link to the test cases you generated for this activity.
- How do you ensure that users that navigate to the protected pages cannot bypass authentication requirements?
- How do you protect against session fixation?
- How do you ensure that if your database gets stolen passwords aren't exposed?
- How do you prevent password brute force?
- How do you prevent username enumeration?
- What happens if your sessionID is predictable, how do you prevent that?

Activity 4: The Content (PHP/Python/Ruby/Go/NodeJS)

Story

As a developer I would like to have a mock video sharing platform.

Goals

After logging in, users will be able to upload new videos. Uploads can be done by supplying a URL to an online movie (which will be downloaded by your code), or by uploading a movie file directly. Existing videos from all users will be available to watch and should have some indication of the user who uploaded them. If you uploaded the video you should be able to delete it. You are free to use PHP/Python/Ruby/Go/NodeJS for a backend language. You may use any frontend technologies you wish, HTML, JS, any JS frameworks (JQuery, Angular).

Backend constraints

- There should be a test case where a user authenticates and uploads a new video, accesses their video, and then deletes it.
- Videos should be stored on the server with metadata stored in the Database
- When a video is deleted it should be deleted from the disk.
- Ensure that your provided DockerFiles and Docker-compose file still function correctly

Questions

- How do you prevent XSS is this step when displaying the username of the user who uploaded the video?
- How do you ensure that users can’t delete videos that aren’t their own?

Activity 5: SQL Injection (Classic/Blind)

Story

As a user I'd like to be able to demonstrate two different form of SQL Injection (Classic and Blind).

Goal

Take the application that you've made and introduce two different types of SQL injection vulnerabilities and then demonstrate them. You shouldn't drastically modify the functionality of your application to do this; you have plenty of database reads and writes.

Backend constraints

- You should develop a test that demonstrates that your application is vulnerable to classic SQL Injection
- You should develop a test that demonstrates your application is vulnerable to blind SQL Injection.
- Provide a video that demonstrates you manually exploiting the classic SQL Injection
- Provide a video that demonstrates you manually exploiting the blind SQL Injection.
- Ensure that your provided DockerFiles and Docker-compose file still function correctly.

Questions

- How would you fix your code so that these issues were no longer present?
- What are the limitations, if any that, of the SQL Injection issues you've included?

Activity 6: SSRF

Story

As a user I'd like to be able to demonstrate Server Side Request Forgery. You shouldn't drastically modify the functionality of your application to do this.

Goal

SSRF often occurs in more complex sites that have a lot of backend. In this case you'll need to introduce a SSRF vulnerability and provide a test to demonstrate that it works (feel free to be creative).

Backend Constraints

- You should develop a test that demonstrates that your application is vulnerable to SSRF
- Provide a video that demonstrates you manually exploiting the SSRF
- Ensure that your provided DockerFiles and Docker-compose file still function correctly

Questions

- How would you fix your code so that this issue is no longer present?
- How does your test demonstrate SSRF as opposed to just accessing any old endpoint.

Activity 7: Command Injection

Story

As a user I’d like to be able to demonstrate Command Injection

Goals

Command injection often can occur when developers pass code to the system to be executed. You’ll be purposely introducing this issue in your code to demonstrate the ease with which this issue can occur You shouldn’t drastically modify the functionality of your application to do this.

Backend Constraints

- You should develop a test that demonstrates that your application is vulnerable to command injection
- Provide a video that demonstrates you manually exploiting the command injection
- Ensure that your provided DockerFiles and Docker-compose file still function correctly

Questions

- How would you fix your code so that this issue is no longer present?

Submission Tests

Activity 1 – Public Github Project is setup in accordance with Agile methodology Questions are answered in provided document (Submission 1 50% total)

Activity 2.1 – You're able to demonstrate that you have a testing environment setup with a minimal test on an established testing platform (Submission 1 25%)

Activity 2.2 – You can show that you have developed documentation prior to development (Submission 1 25%)

Activity 3 –The authentication environment is provided in submitted Docker containers and it can be demonstrated that authentication works via testing. Questions are answered. (Submission 2 20%)

Activity 4 – The content of the application is provided in submitted Docker containers and it can be demonstrated that authentication works via testing. Questions are answered. (Submission 2 20%)

Activity 5 – Both forms of SQL Injection are present along with testing to demonstrate their presence and a video showing them being manually exploited. Questions are answered (Submission 2 20%) – Requires Activity 4

Activity 6 – SSRF is present along with testing to demonstrate its presence and a video showing it being manually exploited. Questions are answered (Submission 2 20%) – Requires Activity 4

Activity 7 – Command Injection is present along with testing to demonstrate its presence and a video showing it being manually exploited. Questions are answered (Submission 2 20%) – Requires Activity 4