Defining Secure Software Requirement

Overview

In this lab, you are going to define the secure software requirements for your project. This is a very important step before you come up with the design of your application. During this lab, we will walk around and understand how the team define the requirements for your application.

Outcomes

Upon completion of this session, you and your team should be able to

- Understand the secure software requirement
- Define the secure software requirement of your project

Tasks:

Work in teams, discuss with your teammates on the following, and write down the requirements:

1. Functional requirements of your application (During this lab, please limit the discussion to 2 – 3 major requirements)
2. Nonfunctional requirements of your application (During this lab, please limit the discussion to 2 – 3 major requirements)
3. Based on the functional and nonfunctional requirements, define the following security requirements, you may refer to SANS Securing Web Application Technologies [SWAT] Checklist for better understanding of the requirements: https://www.sans.org/cloud-security/swat/
    - Confidentiality
        1. Define the data has to be protected in your application.
        2. Identify the confidentiality risks of those data.
        3. Define the requirements protect those data.
    - Integrity

1. Discuss and identify the data that has to ensure the integrity in your application.
2. Identify the integrity risks of those data.
3. Define the requirements ensure the integrity of the data.

- Availability
  1. Discuss how to ensure the availability of your application.
  2. Identify the availability risks of your application.
  3. Define the availability of your application.
- Authentication
  1. Define the users of the application.
  2. Identify the authentication risks of your application.
  3. Define the requirement for your application authenticate the users.
- Authorization
  1. Determine If there are difference type of users in your application.
  2. Identify the authorization risks of your application.
  3. Define the different permission in your application.
- Accountability
  1. Define what information that required for accountability.
  2. Identify the accountability risks of your application.
  3. Define the requirement for your application to records user's activities.

Lab:

OWASP SecurityRAT: https://owasp.org/www-project-securityrat/ is a tool to simplify security requirement management during development using automation approaches.

The core functionality of SecurityRAT ("Security Requirement Automation Tool") can be described in the following steps:

1. You tell SecurityRAT what kind of a software artifact you're going to develop / are running
2. SecurityRAT tells you which requirements you should fulfill.
3. You decide how you want to handle the desired requirements.
4. You persist the the artifact state in an issue tracker and create tickets for the requirements where an explicit action is necessary
5. Throughout the continuous development of the particular artifact, you respect the rules defined in SecurityRAT and document relevant changes in requirement compliance whenever appropriate.
6. If you don't want to install, you can try the demo via: https://securityrat.org

   Username: demo

   Password: SecurityRATdemo10!

   You can also play around with changing the requirements, the database is reset every 24 hours.

Installation:

Step 1: Install docker: https://www.docker.com

Step 2: In command prompt run:

```
docker pull securityrat/all_in_one
```

Step 3: In command prompt run:

```
docker run -it -p 9002:9002 securityrat/all_in_one
```

Step 4: once the image has started, navigate to https://localhost:9002 and accept the self-signed certificate in your browser

Step 5: authenticate with one of the default users admin/admin or user/user

Step 6: start playing around, the detail video demo and document located via:

https://securityrat.github.io

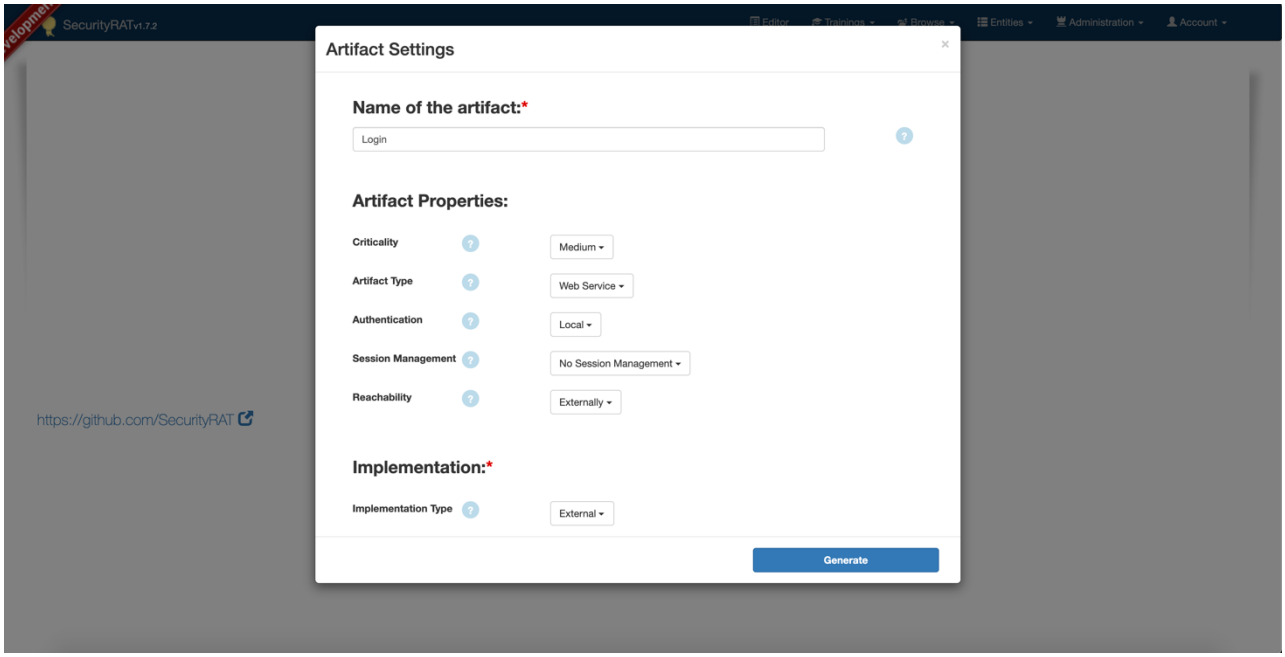
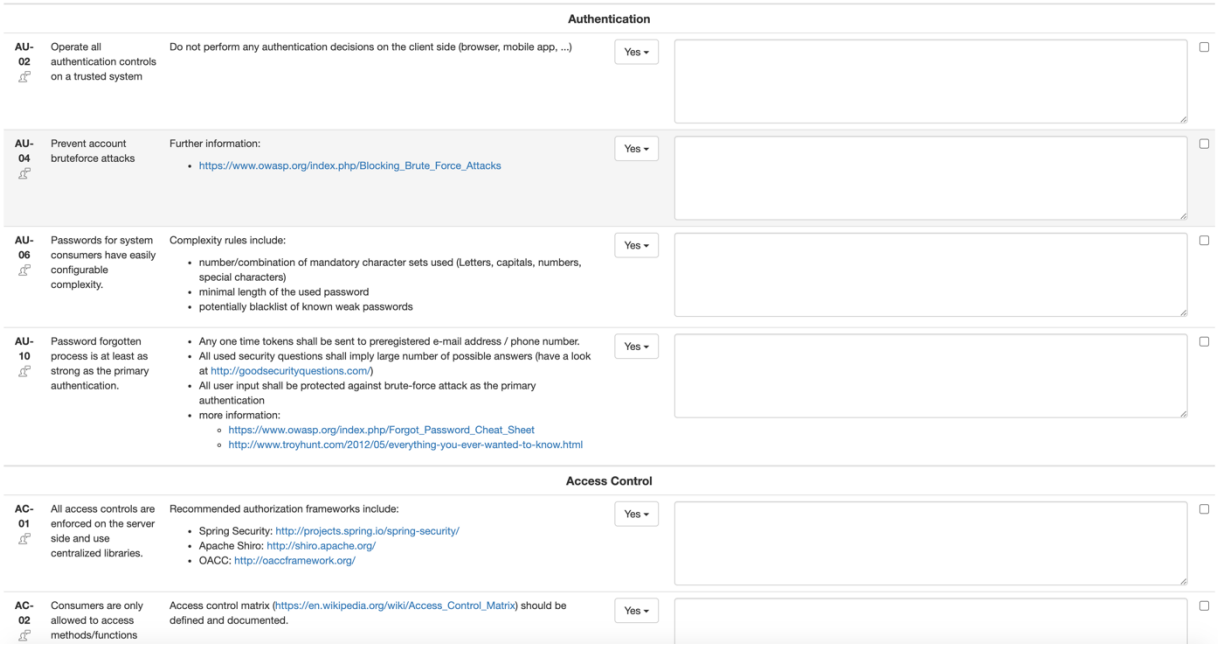


Figure 1: Create Artifact



Authentication				
AU-02	Operate all authentication controls on a trusted system	Do not perform any authentication decisions on the client side (browser, mobile app, ...)	Yes	
AU-04	Prevent account bruteforce attacks	Further information: • https://www.owasp.org/index.php/Blocking_Brute_Force_Attacks	Yes	
AU-06	Passwords for system consumers have easily configurable complexity.	Complexity rules include: • number/combination of mandatory character sets used (Letters, capitals, numbers, special characters) • minimal length of the used password • potentially blacklist of known weak passwords	Yes	

AU-10	Password forgotten process is at least as strong as the primary authentication.	• Any one time tokens shall be sent to preregistered e-mail address / phone number.
		• All used security questions shall imply large number of possible answers (have a look at http://goodsecurityquestions.com/)
		• All user input shall be protected against brute-force attack as the primary authentication
		• more information:
		  ◦ https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet
		  ◦ http://www.troyhunt.com/2012/05/everything-you-ever-wanted-to-know.html
			Yes	

Access Control				
AC-01	All access controls are enforced on the server side and use centralized libraries.	Recommended authorization frameworks include: • Spring Security: http://projects.spring.io/spring-security/ • Apache Shiro: http://shiro.apache.org/ • OACC: http://oaccframework.org/	Yes	
AC-02	Consumers are only allowed to access methods/functions	Access control matrix (https://en.wikipedia.org/wiki/Access_Control_Matrix) should be defined and documented.	Yes	

Figure 2: The requirement to consider after generation

END OF DOCUMENT