SYNOPSYS®

Defending JavaScript

Lab Guide

React and Node.js

Synopsys Inc.
185 Berry Street, Suite 6500
San Francisco, CA 94107 USA

www.synopsys.com/software

Table of Contents

Lab Objectives

During this lab, you will be presented with an application written in React and Node.js that has security defects. You will need to find the vulnerabilities in the code, modify the code using the best practices discussed, and verify that the issues have been remediated.

The objectives of the lab are to:

- Understand the risks using trusting user-generated content
- Apply server-side validation correctly and identify client-side security faults
- Securely execute code dynamically

Curated Links Inc.

The application in the following exercises is currently a work in progress. The developers are creating an application that is a mixture of HackerNews and Medium, where users are able to create posts linking to external URLs.

Lab Instructions

This lab utilizes a virtual machine hosted in the Skytap environment. The instructor will provide an URL and password to access the environment. If you are experiencing issues, please let the instructor know or try the resources below:

Troubleshooting Resources:

Connecting to VMs Troubleshooting Guide:
http://help.skytap.com#SmartClient_Help_Page.html

Connectivity Checker: Use this tool to check your connectivity to Skytap:
https://cloud.skytap.com/connectivity

Speedtest: Use this tool to check your network performance to Skytap:
http://speedtest.skytap.com/

Source Code: The source code within for the lab is located on Skytap:
/home/student/ILT/React

General Notes and Useful Commands

VM Access
Log in to the Ubuntu VM with the following credentials:

User name: *student*

Password: *student*

To restart the hot-loading front-end and server:

```
pm2 restart index
```

Exercise 1: Cross-Site Scripting via Markdown

In this exercise, you will identify a mistake the developers have introduced unintentionally by explicitly trusting user-supplied HTML. You already know that React automatically escapes any tags to prevent JavaScript from being executed. The developers must keep the HTML markup because they want to display the rendered markdown. Your task is to find the vulnerability introduced and to fix the issue.

Objectives

1. Find the vulnerability introduced in src/Pages/CreatePost.js
2. Fix the vulnerability such that user-input is sanitized instead of being explicitly trusted
3. Verify that the issue has been fixed by preventing the execution of arbitrary code

Instructions

1. Start the VM, open the Firefox browser, and navigate to http://curatedlinks.tld:4000
2. Create a new account by clicking the ‘Register’ on the top right of the navigation bar
3. Authenticate to the application using the credentials you supplied at registration, if you were not automatically logged in
4. Create a new post by clicking ‘Create Post,’ and enter the following values:
    a. Title: NEW!
    b. URL: https://abc.xyz/
    c. Description:
        i. *hello world*
        ii. Manually start typing: <img src=x onerror=alert(document.domain) />
            - Notice the execution of the onerror event
    d. Image URL: https://i.imgur.com/4AiXzf8.jpg
    e. Click ‘Submit Query’
5. Click the Title ‘NEW!’ on the newly created blog post
6. Notice that the JavaScript is stored on the server, and is executed against the client
7. Open the text editor Atom by pressing the green symbol on the task bar, or this can be done manually by opening a terminal and writing atom

8. Modify the React code used to render the Post component located at src/Components/Post.js so that it sanitizes the input from the user containing markdown

Exercise 2: XSS via JavaScript URIs

The application also includes functionality to provide a URL linking to an external website when creating a post. This has created a vulnerability, and it is your job to find and remediate the issue.

Objectives

1. Identify the vulnerability introduced within src/Pages/CreatePost.js
2. Fix the vulnerability so that it does not render insecure URL schemes
3. Verify the issue has been resolved, ensuring JavaScript is not able to execute

Instructions

1. Authenticate to the application with the user you created in Exercise 1
2. Click 'Create Post'
3. When filling out the form, in the URL section, enter the following:
    a. javascript:alert(document.domain)
4. Click the 'Visit' tab on the newly created blog post
5. See that the JavaScript has executed
6. Open your Text Editor (Atom)
    a. Look within the src/Pages/CreatePost.js file
    b. Investigate how the URL are processed in src/controllers/posts.js
7. After your analysis of the server and client configuration, change the source code to ensure only https:// URLs are accepted

Exercise 3: Node Remote Code Execution

The developers of this application have administrative functionality where they can query logs on the web application. This is done by using some of the insecure practices we discussed earlier. In this task, you should find the vulnerability by following the discussion guidance on the best way to process user input on the server when accessing the file system.

Objectives

1. Find the vulnerability within server/controllers/admin.js

2. Fix the code so that users can only execute a list of whitelisted commands
3. Verify that user-input can no longer be dynamically executed

Instructions

1. Click the Burp Suite icon pinned to the navigation bar on the left and open Burp Suite
2. Open the Firefox browser, and navigate to http://curatedlinks.tld:4000
3. Right-click the FoxyProxy Icon (Red Fox Icon) on the taskbar:
    a. Select Use proxy "ILT Proxy" for all URLs
    b. Verify that the proxy has been configured by seeing the blue icon
4. Refresh the page; this can be done by pressing F5 or the Icon
5. Verify that BurpSuite is intercepting the traffic to and from the server:
    a. Open BurpSuite
    b. Click the 'Proxy' tab
    c. Within the Intercept tab under the proxy settings, if the third button says 'Intercept is on,' click the button so it says intercept is off
    d. Click the 'HTTP History' tab and see that traffic has been sent to the host http://curatedlinks.tld:4000
6. On the homepage, login as an administrator with the following credentials
    a. Username: admin
    b. Password: admin
7. Click the link in the menu bar called 'Admin'
8. Click 'Parse Logs'
9. View that the logs are returned to the user
10. Go back to BurpSuite:
    a. Open the 'Proxy' tab
    b. Within the Intercept section, change 'Intercept is off' to 'Intercept is on'
11. Within FireFox, on the Administrative page, click 'Parse Logs' and see that the request is being intercepted within BurpSuite
12. Replace the body of the request with the payload:
    a. {"input":"var fs = require('fs');fs.readFileSync('/etc/passwd', 'utf8')"}
13. Notice that the response has returned the passwd file

14. Open your Text Editor (Atom), and ensure user input is not evaluated. Instead, set a requirement to only allow certain methods to execute the retrieve the logs.

Exercise 4: Client-Side Validation

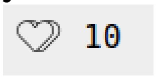
The developers of this application have added client-side validation on user actions to provide a fast and prompt user experience. However, they forgot to add any authentication checks on the server-side!

Can you figure out a way for your post to get to the #1 spot on the front-page?

Objectives

1. Find where the developers have mistakenly forgotten to add server-side protections within server/controllers/posts.js.
2. Fix the vulnerability to ensure unauthenticated users cannot 'heart' posts, and that authenticated users can only 'heart' a post once!

Instructions

1. Log out of the administrator account and log back with your regular user account.
2. Navigate to the front-page.
3. Click a blog post that you want to 'heart.'
4. Open BurpSuite and click 'intercept.'
5. Go back to the blog post and click the 'Heart' icon.
6. With the captured request, right-click the BurpSuite request and click 'Send to repeater.'
    a. Replay the request, and notice that the server does not check if the user is authenticated.
7. Continue sending the request using the repeater, and vote your selected post to the top!
8. Refresh the page in your browser and observe that multiple votes have been applied to the post. 

   10
9. Open your text editor (Atom).
10. Navigate to server/controllers/posts.js, locate the method that deals with the heart functionality and note down all the issues you see with it.

11. The application needs some design modifications to keep track of which users have already liked a post. Can you provide some high-level recommendations and/or pseudocode that the development team can use to fix the problems?
12. EXTRA CREDIT
    Can you find the following additional issues in the code base?
    a. Anonymous database connections
    b. Hard-coded secrets
    c. Mass assignment/parameter pollution
    d. Cross-site request forgery, etc.