DD2525 – Lab Assignment 2
Web Application Security

Deadline: April 29, 2025

1 Introduction

In this assignment we provide a lite forum application, which is a fork of the *forumerly* project available at https://github.com/jayvolr/forumerly. The forum enables registered users to interact on various topics of interest like technology, entertainment, sports, as well as general discussion on the web site itself. Besides, *forumerly* offers a parking service that allows logged in users to park/unpark their vehicles. By providing a valid license plate and parking location, *forumerly* tracks every user's vehicles and parking locations. We have modified the forum application by inserting a few vulnerabilities which you are expected to find, exploit, and fix.

Goal of the assignment The goal of this assignment is to detect, exploit, and fix vulnerabilities in *forumerly*. To do so, you may need to read articles and blog posts that describe these vulnerabilities and the corresponding fixes. When implementing a fix, you should also discuss how existing web defenses (such as those presented in Lecture 4 and 5) can help protect against the attacks that you found.

2 Overview of the web application

The web application contains 9 fixed topics which allow registered users to engage in conversations either by starting a new thread or by replying to existing threads. A user needs to be logged in to create a new thread or reply to existing threads, however any user can read existing threads.

forumerly allows registered users to edit their profiles, e.g., by uploading a profile picture or adding information about themselves. However, there exists only one admin user and it is not possible to create an admin profile directly from the *forumerly* interface.

The project is built with Express.js (on top of Node.js) and uses as third-party services MongoDB, Redis, and our Authentication Service (we are reluctant to trust another service for authentication). The third-party services are delivered as a container images. Figure 1 provides a high level overview of *forumerly*'s architecture.

forumerly is a web application that listens on port 3000. You can open `http://localhost:3000` in a browser after setting up the server (explained below) and explore it. The source code of the application is located in the folder `forumerly` provided with the assignment. It has 3 internal folders: `public` contains static resources like CSS, images, and client-side JavaScript. All files in this folder are available for download by the browser and may be included in HTML pages. `routes` contains server-side JavaScript files that

handle client requests. This is the core business logic of the application. You can read more about this in the Express.js docs on routing. `views` contains templates for Web pages. At runtime, the template engine replaces variables from a template file with actual values, and transforms the template into an HTML file sent to the client. You can see what forms are rendered by the template code even if you do not have the necessary user permissions to open these forms in the browser. The root folder `forumerly` contains some utility files and an entry point of the Node.js application `app.js`.

If you are curious to see what is stored in your MongoDB database you can use MongoDB Compass tool `https://www.mongodb.com/products/compass`.

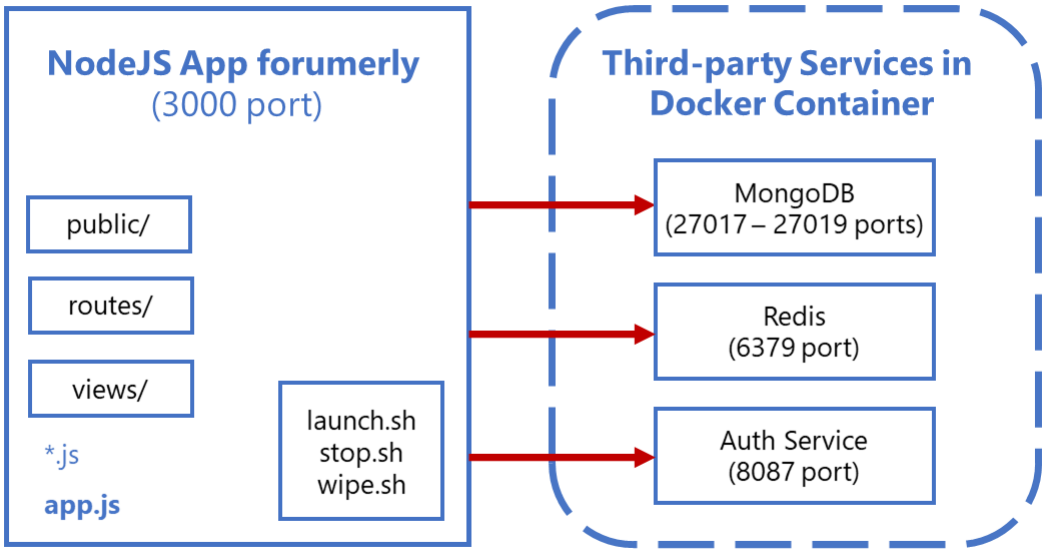

Figure 1: The *forumerly* project architecture.

The assignment's zip archive with source code also contains the folder `tools`. The files from this folder are needed only for subsection 5.4.

3 A primer on the Node.js engine

Designed with scalability in mind, Node.js (and JavaScript in general) is an asynchronous event-driven framework. When an application wants to perform a potentially blocking action, such as opening a file or writing to the network, it registers a callback function, triggers the relevant action, and terminates. When the action completes (e.g., the file was opened), the callback event is called by the event loop.

The event-driven model is extremely scalable, as threads never "sit and wait", but it introduces problems when a function takes long time to complete. Since the Node.js server runs only one thread per core (by default), such a long-running function would take up the entire core. Node.js is particularly susceptible to vulnerabilities as you will notice in this lab. You can read more on the Node.js website.

4 Getting Started

Download the zip file from the course web page in Canvas. Once you have downloaded the project, please check the requirements to run the application.

4.1 Setup

1. To run the forum application you need the Node.js engine; follow the instructions at `https://nodejs.org/en/` to install it in your computer. Note that you may already have Node.js installed from Lab assignment 1.

2. Since the web application uses third-party services, you need to install Docker. Please follow the instructions at `https://docs.docker.com/install/` to install Docker. Note that you may already have Docker installed from Lab assignment 1.

3. Once Docker is installed, run the bash script `launch.sh` present in the zip folder. This script launches the services needed to run the application: MongoDB with the database fixtures, Redis for user sessions, and the Authentication Service.

4. Node.js applications usually use several third-party packages; to install all of them, run the command `npm clean-install` in the *forumerly* project root.

In the folder *forumerly* you will find two more scripts, `wipe.sh` and `stop.sh`. `wipe.sh` will stop and remove the services from the Docker engine. Therefore, you will lose all the information in the *forumerly* database if you added any data to it. The `stop.sh` command will stop the services without deleting the containers. These scripts are useful if you want to clean up the database in case it is corrupted. In this case you should run `wipe.sh` followed by `launch.sh`, i.e., deleting all the services and then creating them again.

4.2 Launch application

Now you are ready to run the forum application. Go to the root folder of the project and type `node app.js` – this command will launch the application server in port 3000. Now, open your browser and type `http://localhost:3000`, and get ready to break and fix the application.

You can use your favorite IDE or a text editor to explore the code. We recommend to install and use Visual Studio Code. Visual Studio Code has a built-in debugger for Node.js allowing you to see how your requests are processed in real time. This can be useful when preparing exploits and figuring out how some features work.

4.3 Troubleshooting

- **The application is not running in the browser:** Check that port 3000 is free, this is the port that *forumerly* uses. Take a look at these links to check busy ports in Windows and Linux.

- **The application runs, but I cannot login or signup:** Check port 6379 is free (this port is used by the Redis service), port 8087 is free (this port is used by the authentication service), and ports 27017 - 27019 are free (these ports are used by MongoDB).

- **Docker services do not work in my Windows machine** Check that your Docker environment is set to linux.
- **The ports are freed but services are not running**. Run `wipe.sh` followed by `launch.sh`.
- **Other issues?** Drop us a line on Canvas discussion forum and we will help you.

5 Vulnerabilities

This section describes the classes of vulnerabilities that you are expected to find, exploit, and fix in *forumerly*. section 8 contains reference material that you may need to read to complete the tasks.

5.1 Regular Expression Denial of Service (ReDoS)

Regular expressions are widely used in today's web applications. For instance, the backend framework of a web application server may use regular expressions to route and catch URL parameters in regular GET requests. Regular expression Denial of Service (ReDoS) is a class of algorithmic complexity attacks where matching a regular expression against an attacker-provided input takes an unexpectedly long time.

The project contains vulnerable application logic that is subject to ReDoS attacks. Find a ReDoS attack in the project and answer the following questions in the final report:

- Exploit: What is the actual attacker's input and why does it cause a ReDoS attack?
- Mitigation: How can you prevent the exploitation of the discovered ReDoS and why does it work?

5.2 Web Side Channels

A side channel is a method to obtain information about a system or its data by analyzing unintended or auxiliary signals, for example the storage side channels.

Unfortunately, *forumerly* contains several side channels, posing a significant security risk. The side-channels in *forumerly* are of the same type and are related to its registration and parking features. Your goal is to identify storage side channels for these two features in the code, implement their exploitation, and propose effective mitigation strategies. Note that the information gathered from one side channel may be needed to exploit the second side channel. You get 2 points for only exploiting the first side channel.

The final report must address the following questions:

- What information in *forumerly* can be used as a side channel? Provide a step-by-step demonstration of the information that can be inferred and how it can be achieved.
- How can the side channel be exploited? Provide an implementation of how to exploit each of the found side channels. We provide some ideas for how to use Python for scraping and querying *forumerly* in the webscraping folder.
- How can the side channel be fixed? Propose effective fixes to the current *forumerly* implementation to mitigate the side-channel attacks. You don't have to necessarily implement them. Note that the communication between *forumerly* and third-party services cannot be changed.

5.3 Cross Site Scripting (XSS)

Cross Site Scripting (XSS) attacks occur when an application takes untrusted data and sends it to a web browser without proper validation or sanitization, thus allowing attacker's malicious code to be executed in a victim's browser.

This project may contain several XSS attacks. You need to detect and exploit at least one. We recommend to explore the *forumerly* application and test it with input data that contains HTML tags and XSS vectors, identify uses of these tags, and examine how this input data is validated on the server-side. The documentation of Express.js and mustache.js can be helpful to figure out when an application escapes HTML in variables.

When you find a way to inject JavaScript code into a Web page, you may notice that the browser blocks the execution of the attacker's JavaScript code. You can explore the logs in the browser console and read about Content Security Policy (CSP). You should identify weaknesses in configured CSP policy in order to find at least one bypass and demonstrate the XSS attack. We recommend starting with HackTricks - Content Security Policy (CSP) Bypass and checking the described unsafe CSP rules in the *forumerly* application. One of the bypasses requires finding Unrestricted File Upload, examine the application for usages of write-to-file functions and file upload APIs. We also recommend reading the paper Code-reuse attacks for the web: Breaking cross-site scripting mitigations via script gadgets to get ideas for other possible bypasses.

You should implement a fix for the detected XSS to sanitize the untrusted data in a secure way and set up CSP properly to avoid exploitation of other XSS in *forumerly*. We grade the detection and fix of XSS as 2 points. One bypass of CSP and proper CSP setup for the fix is worth 3 points. You also can take 1 bonus point if you demonstrate two CSP bypasses.

The final report must address the following questions:

- How did you exploit the XSS? What requests did you need to send to the server?
- Does the server-side code contain any input data validation against XSS attack?
- What is the Content Security Policy (CSP)? How can it be bypassed?
- What impact does this vulnerability have? What data can an attacker steal? If you aren't sure, test it by the detected XSS.
- How did you fix this vulnerability? Can you set up a web application to prevent the execution of any malicious script? How would you do it?

5.4 Remote Code Execution (RCE)

Remote Code Execution (RCE) is one of the most dangerous types of cyber attacks as it allows to remotely run malicious code within the target system on the local network or over the Internet. RCE can exploit various kinds of weaknesses in the code: buffer overflows, use-after-free, insecure deserialization, business logic, and software design flaws. *forumerly* contains a prototype pollution vulnerability that leads to RCE. Your goal is to detect and exploit the RCE, thus getting a remote shell on the web server. Moreover, you should implement and describe a fix for this vulnerability.

We recommend reading of the paper Silent Spring: Prototype Pollution Leads to Remote Code Execution in Node.js to study how prototype pollution can be exploited in Node.js applications. After that, you need to examine the source code of *forumerly* to find the prototype pollution pattern and exploit it to achieve RCE. The following remarks and questions will help you focus on the right steps for finding the vulnerability:

- Review the code of *forumerly* for prototype pollution patterns. You can try searching by computed property assignments like `object[exp1] = exp2`. If *object* refers to `Object.prototype` and an attacker controls values of *exp1* and *exp2*, then you found a good candidate for a possible vulnerability.
- What request should an attacker send to trigger an execution of the prototype pollution pattern?
- Should an attacker be authorized in the system to execute code subject to prototype pollution?
- How can you verify that you achieve prototype pollution exploitation, e.g. by using a debugger with VSCode or similar? Can you achieve Denial of Service by prototype pollution?
- Which code of *forumerly* can be affected by prototype pollution and lead to RCE? Try to find function calls that builds and executes a JavaScript function at runtime. These calls can be either `eval()` or `new Function()`. If an argument of the call is data from a property read, this is a good candidate for a gadget.
- What request should an attacker send to trigger a gadget execution? What property should be polluted to achieve the gadget execution and pass an attacker controlled value to `eval()` or `new Function()`?

The next step is to generate a Reverse Shell payload in order to run it on the server and get full remote control of the server. The following recommendations can be helpful.

- Look at Reverse Shell Cheatsheet or read the blog post Exploiting Node.js deserialization bug for Remote Code Execution for information on Node.js reverse shell. We highly recommend using the script in `tools/nodejsshell.py` to generate the JavaScript code for Reverse Shell. This is a customized version of the script described in the blog post above.
- Run `nc -l <ip-address> <port>` on your computer to listen a port for incoming connections from Reverse Shell. If you use Windows, you can install Nmap and use `ncat -l <ip-address> <port>` instead of `nc`.

The final step is a combination of all parts of the exploit and delivery of the payload to the server. You need to prepare a web request that contains Reverse Shell code as a value of the property to pollute, trigger the prototype pollution vulnerability by this request, then trigger the gadget by another request and get remote access on the server. If you need to upload a file with your payload via HTML form, one of the ways is to reconstruct the HTML form that sends the same request to the server (read about HTML forms) or use `curl --form <name=content>` command.

For this task, you can get 3 points if you find prototype pollution, exploit it to achieve Denial of Service, and implement a fix. You can get 2 additional points if you get RCE on the server, as described in the task description.

6 Additional Vulnerabilities

The original version of *forumerly* turns out to contain other vulnerabilities. We did not fix them, hence you can try to detect and describe these vulnerabilities in the report. However, we do not accept XSS vulnerabilities for bonus points (sorry!), therefore you should try to find other kinds of vulnerabilities. You need to show a demo for each extra vulnerability and explain how to mitigate it.

Read about other popular client-side vulnerabilities, e.g. Path Traversal, and try to use this knowledge for penetration testing of *forumerly*. We do not provide any hints for these vulnerabilities and it is as close to the wild as it gets. Act like a real hacker and get 1 bonus point for each vulnerability found.

7 Requirements and Grading

7.1 Report

A zip file to be submitted via Canvas that should include the following:

1. Source code of your solutions.
2. A report describing how you approached the problem and the design of mitigation strategies. The report should contain, for each reported vulnerability, how you exploited it, and what mitigation strategy you implemented. Finally, the report should contain answers to questions inlined in specific tasks.
3. Clear statement of contributions of each group member.

7.2 Grading (max 22 points)

- Each task in section 5 is worth **5 points**.
- To pass the lab you should solve at least 2 tasks from section 5 and get 10 points.
- You can get **1 extra point** for demonstration of **two** different CSP bypass techniques for subsection 5.3.
- You can get **1 point** for every additional vulnerability beyond XSS in section 6.

Observe that the points above assume that the report is satisfactory and every group member is able to explain the solution and answer questions during the live presentation of the lab. Failure to do so may result in deduction of points. Moreover, any change to the third-party services is **NOT** accepted as a solution.

8 Reference material

- Best security practices in NodeJS
- Web side-channels (highlight on the storage side-channel)
- Freezing the Web: A Study of ReDoS Vulnerabilities in JavaScript-based Web Servers
- HashWick vulnerability
- HackTricks - Content Security Policy (CSP) Bypass
- Code-reuse attacks for the web: Breaking cross-site scripting mitigations via script gadgets
- Silent Spring: Prototype Pollution Leads to Remote Code Execution in Node.js