TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Maksim Semjonov 201757IVSB

Securing Web Applications Built with Express.js

Bachelor's thesis

Supervisor: Priidu Paomets
MSc
Co-supervisor: Toomas Lepikult
PhD

Tallinn 2024

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Maksim Semjonov 201757IVSB

Express.js baasil loodud veebirakenduste turvamine

bakalaurusetöö

Juhendaja:	Priidu Paomets
	MSc
Juhendaja:	Toomas Lepikult PhD

Tallinn 2024

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Maksim Semjonov

04.11.2023

Abstract

This thesis presents a comprehensive study on securing web applications developed using the Express.js framework within the Node.js environment. Main sources of the information are OWASP Top 10 list, along with official Node.js and Express.js documentation. The methodology combined a thorough literature review, practical evaluations of Express.js applications, and a systematic design process. This multifaceted approach ensured a deep understanding of Express.js and Node.js vulnerabilities and the development of effective mitigation strategies. The research identified key vulnerabilities in Express.js applications, categorized into three main areas, and presented practical mitigation strategies. The creation of a PDF guide made these findings accessible, providing a valuable tool for developers and IT professionals. This work significantly contributes to web application security, offering actionable insights that enhance the digital safety of businesses and individuals reliant on these technologies. The thesis thus plays a crucial role in strengthening the security of web applications in an increasingly digital-dependent world.

This thesis is written in English and is 26 pages long, including 6 chapters, 0 figures and 0 tables.

List of abbreviations and terms

API	Application Programming Interface
BOLA	Broken Object Level Authorization
BOPA	Broken Object Property Level Authorization
DIY	Do It Yourself
DoS	Denial of Service
DDoS	Distributed Denial of Service
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDOR	Insecure Direct Object Reference
I/O	Input-Output
MFA	Multi-Factor Authentication
NPM	Node Package Manager
OWASP	Open Web Application Security Project
TLS	Transport Layer Security
USPS	United States Postal Service
URL	Uniform Resource Locator

Table of contents

1 Introduction

In the digital epoch, the fabric of our online existence is increasingly dependent on web applications. These applications range from simple websites to complex cloud-based services, and are the major part of modern communication, entertainment, and financial solutions. Among the stack of technologies powering these applications, Express.js, a framework for Node.js, has emerged as a base in this digital world. However, as the reliance on these technologies grows, also grows the spectrum of cyber security threats they face. This thesis, situated within the study program of Cyber Security Engineering, seeks to address the value of enhanced security in web applications developed using Express.js.

1.1 Motivation

Author has gathered remarkable experience in developing of web applications, particularly with Express.js, has been a journey marked by both interest and caution. The ease and efficiency of Express.js are combined with an array of security vulnerabilities that threaten the integrity of web applications. This has not only raised authors professional curiosity but has also ignited a sense of responsibility. As cyber threats evolve in sophistication, the task of securing web applications becomes not just a technical challenge but a critical urgency. This thesis is driven by authors experiences in developing Express.js applications and the realization of the need for a comprehensive, practical approach to their security.

1.2 Problem Statement

The spread of web applications has been a defining trend in the modern web world. Express.js, known for its minimalism and flexibility within the Node.js ecosystem, has emerged as an efficient tool for many web developers. However, its widespread adoption has made it a huge target for cyber threats. This thesis aims to tackle the critical challenge of identifying and mitigating security vulnerabilities in web applications developed using Express.js, a task of priority importance in Cyber Security.

Real-world incidents vividly illustrate the severity of this issue. Consider the infamous Equifax data breach in 2017, which led to the exposure of sensitive data of over 147 million individuals. This breach was traced back to an unpatched vulnerability in a web application framework alike to Express.js. The incident not only showcased the consequences of security omissions but also underscored the necessity for stringent security protocols [1].

Another illustrative case is the 2018 Node.js event-stream incident, where a widely used *npm package* was compromised to steal cryptocurrency. This package, integral to many applications including those built with Express.js, was targeted due to lax security measures in managing third-party packages, a prevalent issue in the Node.js environment [2].

These examples highlight the complex nature of the challenge. Securing Express.js applications is not just about fortifying the framework itself; it also involves safeguarding the entire application stack, including external dependencies. The thesis will carefully study these challenges, focusing on common vulnerabilities. Additionally, it will explore critical yet often overlooked aspects like secure session management, data encryption, and secure configuration practices.

The implications of this problem are deep. In an era where data breaches can lead to substantial financial losses and irreparable reputational damage, securing web applications goes beyond a technical responsibility and becomes a business imperative. Through this thesis, author aim to improve the Cyber Security Engineering field with practical, actionable solutions to these pressing security challenges.

1.3 Contribution

This thesis aims to contribute to the field of Cyber Security, particularly in the context of securing Express.js web applications. The contributions of this research are following:

1. **Analysis of Security Vulnerabilities**: One of the key contributions is an analysis of the common security vulnerabilities associated with Express.js and the related to it Node.js environment. This analysis will not only identify the vulnerabilities but also explain why they occur and how they can be exploited.

2. **Development of Mitigation Strategies**: The research will go beyond merely identifying vulnerabilities. It will develop and present detailed strategies for mitigating these security risks. These strategies will be practical and actionable, tailored to be implemented in real-world Express.js applications.
3. **Creation of a Security Guide**: A significant contribution result of this thesis is a complete guide that provides step-by-step instructions on securing Node.js applications with Express.js framework. This guide is a valuable resource for developers, offering practical advice and best practices in a format that is easy to understand and apply.
4. **Bridging Theory and Practice**: By combining theoretical knowledge with practical application, this thesis will bridge the gap often found in cybersecurity studies. It will provide a balanced view that respects the complexities of the theoretical aspects of cybersecurity while also addressing the practical challenges faced by developers.
5. **Enhancing Cybersecurity Awareness**: This research will increase awareness about the importance of cybersecurity in web application development. This will highlight the need for developers to be proactive in implementing security measures as an integral part of the development process.
6. **Contribution to Academic and Professional Communities**: The findings and recommendations of this thesis will be useful not only to students and scholars in the field of cybersecurity engineering, but also to professional web developers and cybersecurity specialists. It will serve as a starting point for further research and discussion within the community.

The author's contribution to this thesis involved conducting the literature review, engaging in practical evaluations of web applications, synthesizing information from the OWASP Top 10 [3] and official documentation, designing the comprehensive PDF guide. The author's efforts in compiling, analysing, and presenting this information have been instrumental in creating a resource that not only addresses theoretical aspects of cybersecurity but also provides practical, actionable guidance for enhancing the security of Express.js applications.

In summary, this thesis will contribute to the field of Cyber Security Engineering by providing a detailed examination of security vulnerabilities in Express.js applications, proposing effective mitigation strategies, and offering a comprehensive guide for developers. These contributions are aimed at enhancing the security of web applications and raising awareness about the importance of cybersecurity in the digital age.

1.4 Target Audience

This thesis aims to reach a broad range of experts and individuals who work in the fields of online application development and cybersecurity, either directly or indirectly. The following groups are specifically targeted to benefit from the research:

1. **Web developers:** This thesis is primarily intended for the developers, particularly those who create web apps with Express.js and Node.js. Advice and techniques offered in the final sollution will be especially helpful to developers who want to improve the security of their apps.
2. **Cybersecurity Professionals**: Professionals in the field of cybersecurity will find this research beneficial in understanding the specific vulnerabilities associated with Express.js and Node.js. The mitigation strategies outlined can be integrated into broader cybersecurity practices and protocols.
3. **Students and Academics in Cyber Security Engineering**: This thesis is a helpful resource for students studying Cyber Security Engineering and related subjects to grasp the practical aspects of securing online applications. This research can be used as a teaching tool by academics to demonstrate the difficulties and solutions that arise in real-world application security.
4. **Decision-makers and IT managers:** Decision-makers and IT managers in companies that use online apps on a regular basis will get a thorough grasp of how crucial it is to put strong security measures in place.
5. **Technology Enthusiasts and Hobbyists:** The information in this thesis will be of interest to those with a great interest in online technologies and cybersecurity, as well as hobbyists and enthusiasts. It can be used as a reference to comprehend the intricacies of web application security.
6. **Policy Makers and Regulatory Bodies:** Although they are not the thesis' major target audience, policy makers and regulatory bodies that oversee establishing web application security standards might benefit from the research conducted for this thesis by better understanding the problems and possible solutions in this field.

In essence, this thesis is designed to be a comprehensive resource for anyone interested in or responsible for the security of web applications. It aims to disseminate knowledge that is both practical and theoretically sound, contributing to the overall enhancement of cybersecurity practices in the digital landscape.

1.5 Objectives and scope

The objectives and scope of this thesis are designed to provide a clear and focused exploration of securing web applications built with Express.js within the broader context of Cyber Security Engineering. The following outlines the primary of this research:

1. **Identify Key Security Vulnerabilities**: To systematically identify and catalog the primary security vulnerabilities that affect Express.js web applications, including but not limited to injection attacks, cross-site scripting, and session hijacking.
2. **Develop Mitigation Strategies**: To develop comprehensive strategies and best practices for mitigating identified vulnerabilities. This includes both preventive measures and reactive strategies to address security breaches.
3. **Create a Practical Security Guide**: To compile the findings and strategies into a user-friendly guide that serves as a practical resource for web developers. This guide will provide actionable steps for securing Express.js applications.
4. **Bridge Theoretical and Practical Aspects**: To integrate theoretical cybersecurity principles with practical application development scenarios, providing a holistic view of web application security.
5. **Raising Awareness**: To increase awareness among web developers, students, and other groups related to development and security fields about the importance of cybersecurity in web application development and the challenges associated with Express.js.

The scope of this thesis is limited to:

1. **Focus on Express.js and Node.js**: While the primary focus is on Express.js, the research will also cover relevant aspects of the Node.js environment.
2. **Practical Application-Oriented Approach**: The research will punctuate practical, application-oriented solutions and strategies, making it directly applicable to real-world development scenarios.

3. **Current and Relevant Technologies**: The study will concentrate on current and emerging technologies and practices in web application development and cybersecurity, ensuring the relevance and timeliness of the research.
4. **Inclusion of Case Studies and Examples**: To enhance understanding and applicability, the research will include relevant case studies and real-life examples where appropriate.
5. **Limitation to Web Application Security**: The research will be limited to the security aspects of web applications, specifically those built with Express.js, and will not delve into unrelated areas of cybersecurity or software development.

Through these objectives and scope, this thesis aims to make a substantial contribution to the field of Cyber Security Engineering, providing valuable insights and tools for securing Express.js web applications against a backdrop of evolving cyber threats.

2 Literature Review

This section of the thesis provides an overview of the existing literature related to securing web applications, particularly those built with Express.js and Node.js, and outlines the methodology adopted for this research.

2.1 Literature review

The literature review encompasses a range of sources, including academic journals, technical blogs, official documentation, and industry reports, to build a comprehensive understanding of the current state of web application security in the context of Express.js and Node.js. Key areas that the thesis will focus on are the following:

1. Security Vulnerabilities in Web Applications: This includes studies and reports on common security threats faced by web applications. Sources like the OWASP Top Ten provide a foundational understanding of these vulnerabilities [3].
2. Node.js and Express.js Specific Security Concerns: Sources that focus specifically on security in the Node.js environment and Express.js framework. This includes analysis of official documentation of Express.js and Node.js [4, 5].
3. Best Practices in Web Application Security: Articles that outline best practices in securing web applications. These sources provide guidelines and strategies for developing secure code, managing dependencies, and implementing effective security measures.
4. Case Studies and Real-World Incidents: Analysis of real-world security breaches and incidents involving Node.js and Express.js applications. These case studies offer valuable insights into how security vulnerabilities are exploited and the consequences of this.
5. Emerging Trends in Web Security: Literature on the latest trends and emerging threats in web security. This helps in understanding the evolving landscape of web application security and prepares the groundwork for future-proofing applications.

2.2 Express.js Framework

Node.js is a powerful and commonly used platform for building different kinds of applications. It is an open-source, cross-platform runtime environment that allows developers to write server-side code using JavaScript[6]. This section provides an overview of Node.js and provides the reasoning behind choosing Node.js as the programming language for web application development.

This section of the thesis provides an in-depth exploration of Node.js, detailing its architecture, functionality, and the role it plays in the development of web applications, particularly in relation to security considerations:

1. Event-Driven, Non-Blocking I/O Model: Node.js operates on an event-driven, non-blocking I/O model, making it perfect for data-intensive real-time applications that run across distributed devices. This approach allows applications to handle numerous simultaneous connections without blocking other connection, which is a significant advantage in application development [7].

2. Single Programming Language: Node.js allows developers to use JavaScript on both the client and server sides. This simplifies the end-product development, as it reduces the need to switch between different languages for front-end and back-end development [8].

3. NPM Ecosystem: Node.js comes with access to the Node Package Manager (NPM), which contains a huge amount of libraries and modules. This extensive ecosystem also allows developers to easily integrate various functionalities and tools into their applications, reducing the time spent on the development process [9].

4. Scalability: Node.js is designed to be scalable, which is essential for modern web applications that need to handle growth in both requests traffic and data volume efficiently [6].

The language for this thesis was selected based on a detailed analysis:

1. JavaScript's universalism: 2023 marks JavaScript’s eleventh year in a row as the most commonly used programming language, widely used for both client-side and

server-side development. This means that a vast majority of programmers and especially web developers are already familiar with it, reducing the learning curve for Node.js [9].

2. Real-Time Capabilities: Node.js event-driven possibility, coupled with Node.js's non-blocking I/O model, makes it an excellent choice for developing real-time web applications like chatting apps, entertainment platforms, and live streaming services. [7]

3. Community and Support: JavaScript, being widely adopted, has a strong community support and a wealth of resources. This community support is crucial for troubleshooting, learning, and staying updated with the latest trends and best practices. Based on authors personal experince, finding resolution to issues faced while programming was easily done using the stackoverflow portal.

4. Cross-Platform Development: JavaScript's compatibility with different platforms like Web, Android, IOS and its integration into Node.js allows for cross-platform application development, which is cost-effective and efficient. [10]

5. Security Considerations: While JavaScript and Node.js offer a great amount of options and benefits, they also present unique security challenges. The single-threaded nature of Node.js can lead to security vulnerabilities if not properly managed. Additionally, the extensive use of third-party modules from NPM can introduce security risks if these modules are not properly audited and configured. [12]

In conclusion, the choice of Node.js and JavaScript for web application development is driven by their efficiency, scalability, and the ease of using it for cross-platform development. However, this choice also necessitates a careful consideration of the associated security challenges, which this thesis aims to address in the context of Express.js web applications.

In addition to the features and benefits mentioned above, the popularity of Express.js is a significant factor in its selection as the framework of choice for web application development, as this helps developers to find solutions to their problems and expand with different modules:

1. Widespread Adoption: Express.js is one of the most widely adopted web frameworks within the Node.js ecosystem. Its popularity is evident from its extensive use in the industry, ranging from small-scale projects to large enterprise applications. According to npm Express has over 30.000.000 downloads per week. [13]

2. Community Support and Resources: The framework's popularity has led to a robust and active community. This community support translates into a wealth of shared knowledge, resources, and tools, making it easier for developers to find solutions to common problems, stay updated with best practices, and leverage the collective knowledge of experienced professionals. One of the examples is the stackoverflow programming portal.

3. Preferred Choice for Many Companies: Many well-known companies and startups choose Express.js for their web applications due to its efficiency and flexibility. This widespread industry adoption further validates its reliability and suitability for modern web development. [14]

4. Developed Ecosystem of Middleware and Plugins: The popularity of Express.js has contributed to a huge ecosystem of middleware and plugins. Developers have access to a rich array of modules that can be easily integrated into their Express.js applications, enhancing application functionality and reducing development time. [13]

5. Frequent Updates and Longevity: The framework's popularity ensures that it receives frequent updates and maintenance from contributors. This ongoing development is crucial for addressing emerging security vulnerabilities, adding new features, and ensuring the framework stays relevant and effective in the rapidly evolving field of web technology.

In summary, the popularity of Express.js is a key factor in its selection for web application development. It not only reflects the framework's capabilities and industry acceptance but also ensures a rich ecosystem of resources and community support, which are invaluable for both novice and experienced developers alike. This widespread adoption and support also underscore the importance of addressing its security aspects, a central theme of this thesis.

2.3 PDF Creation Using Figma

For the creation of the practical PDF guide in this thesis, Figma, a web-based graphic design tool, will be utilized. Figma's versatility in design and collaboration makes it an ideal choice for designing a visually appealing and informative PDF guide.

Based on the personal experience, Figma is a great online-tool for design tasks:

1. User-Friendly Interface: Figma offers an intuitive interface which simplifies the design process, making it easier to concentrate on the task more. Narrowing the amount of time needed to understand the tool.

2. Flexibility in Design: Figma allows for a high degree of customization and creativity in design. This flexibility is crucial for creating a guide that is not only informative but also engaging and easy to navigate.

3. Compatibility and Export Options: Figma supports exporting designs in various formats, including PDF. This feature ensures that the final product can be easily shared and accessed in the desired format.

Using Figma for the creation of the PDF guide offers a blend of design flexibility, ease of use, and collaborative features, resulting in a professional and effective educational resource.

3 Methodology

The methodology for this research is structured to ensure a comprehensive and systematic approach to understanding and addressing the security of web applications built with Express.js. Research approach for the thesis is following:

1. Literature Compilation: Gathering a wide range of relevant literature, including academic papers, technical reports, official documentation, and real-world case studies.
2. Comparative Analysis: Comparing and contrasting different sources to identify common themes, discrepancies, and gaps in the existing literature.
3. Practical Evaluation: Applying the theoretical knowledge gained from the literature to practical scenarios. This involves experimenting with Express.js applications to understand how vulnerabilities arise and can be mitigated.
4. Development of Best Practices Guide: Synthesizing the information gathered to develop a comprehensive guide on best practices for securing Express.js applications.

The "Data Collection and Analysis" phase of the thesis involved a blend of qualitative analysis, focusing on extracting key topics and insights from the literature on web application security, and quantitative analysis, employing statistical methods to evaluate data from case studies and security reports for identifying prevalent vulnerabilities and assessing their impact:

- Qualitative Analysis: Analyzing the content of the literature to extract key themes, patterns, and insights related to web application security.
- Quantitative Analysis: Where applicable, using statistical methods to analyze data from case studies or security reports to identify common vulnerabilities and their impact.

This methodology provides a balanced approach, combining theoretical knowledge with practical application, to comprehensively address the security of web applications built with Express.js.

4 Analysis of Common Vulnerabilities

4.1 Introduction to Analysis of Common Vulnerabilities

The section of this thesis is dedicated to a meticulous examination of the most prevalent and critical security vulnerabilities that affect web applications developed using Node.js and the Express.js framework. The selection of these vulnerabilities is grounded in a two-pronged approach: leveraging the insights from the "OWASP Top 10 API Security Risks – 2023" [3] and fundamental vulnerabilities from official documentation from Express.js and Node.js [4,5].

The Open Web Application Security Project (OWASP) Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications. However, not all vulnerabilities listed in the OWASP Top 10 are directly applicable to every web framework or environment. Therefore, this thesis selectively examines five key vulnerabilities from the OWASP Top 10 – 2023 list: Broken Object Level Authorization and Session Management, Broken Authentication, Broken Object Property Level Authorization, and Unrestricted Resource Consumption. These vulnerabilities were chosen based on their relevance to the Express.js framework and the frequency of their occurrence in real-world applications. The decision to focus on these particular vulnerabilities also stems from the observation that other vulnerabilities in the OWASP list are either outside the scope of this thesis or are similar to the selected ones but less common in the context of Express.js.

Approach to Analysis

The vulnerabilities selected for analysis in this thesis is not just enumeration of potential security flaws. Every vulnerability is selected to understand its root cause, how it manifests in a Node.js and Express.js environment, and its potential impact on an application. This analysis is supplemented with real-world examples, demonstrating how such vulnerabilities could be exploited. Furthermore, the thesis provides practical mitigation strategies, offering readers actionable guidance on how to secure their Node.js and Express.js applications against these identified risks.

By integrating the report of the OWASP Top 10 with the specific approach to the Node.js and Express.js vulnerabilities, this section aims to provide a comprehensive and structured view of the application security landscape. This approach ensures that the analysis is not only theoretically sound but also practically relevant to developers, security professionals, and anyone interested in the security aspects of modern web application development.

4.2 Broken Object Level Authorization and Session Management

Definition and Context: Broken Object Level Authorization (BOLA) is also known as Insecure Direct Object References (IDOR). This vulnerability can be exploited when an application provides access to an endpoint or resource based on user-supplied direct input. In the context of Node.js and Express.js applications, this often happens when API endpoints do not properly verify the user's authorization to access specific resources. [15]
Real-World Impact: BOLA can lead to unauthorized access and manipulation of data. For instance, if an API endpoint in an Express.js application retrieves user information based on an ID passed in the URL, an attacker could manipulate this ID to access other users' data. [16]
Mitigation Strategies:

1. Implement Strong Access Control: Ensure that access control checks are made for every API endpoint that requires this. Access control should check and approve that the user who made the request has the correct permissions to access or modify the requested resource. [15]
2. Use Indirect Object References: Direct object references can be guessed or bruteforced. This can lead to, for example, instead of using database IDs in URLs or API endpoints, use other identifiers that are not easily guessable. [15]

4.3 Broken Authentication

Definition and Context: Broken Authentication is a critical security vulnerability that occurs when the implementation of authentication mechanisms in a web application is flawed or inadequate. This vulnerability is particularly concerning in Node.js and Express.js applications, where custom authentication processes are often implemented. [17]
Real-World Impact: When authentication is not properly secured, attackers can exploit these weaknesses to impersonate legitimate users. This could lead to unauthorized access to sensitive data, account takeover, and even full system compromise. For example, an

Express.js application with weak password policies or improper session handling could allow attackers to guess or steal user credentials. [18]

Mitigation Strategies:

Strong Password Policies: Implement robust password policies that enforce the use of strong, complex passwords. This reduces the risk of brute force or dictionary attacks. [19]

Multi-Factor Authentication (MFA): Introduce multi-factor authentication to add an additional layer of security beyond just username and password. [19]

Secure Session Management: Ensure that session tokens are securely generated and managed. Implement measures like token invalidation upon logout and automatic expiration. [18]

Rate Limiting and Account Lockout: Implement rate limiting for login attempts and lockout mechanisms after a certain number of failed attempts to prevent brute force attacks. [19]

Encryption and Secure Transmission: Use HTTPS to encrypt data in transit and ensure that credentials are not exposed. Store passwords securely using strong hashing algorithms like bcrypt. [20]

Do not use default credentials: Using default credentials for the configuration or admin users can result in a successful bruteforce attack. [19]

Best Practices in Node.js and Express.js:

Utilize Trusted Authentication Libraries: Leverage well-established libraries like Passport.js for handling authentication in Express.js applications. These libraries are regularly updated and follow security best practices. [21]

Avoid DIY Authentication: Unless necessary, avoid building authentication mechanisms from scratch. It’s safer to use a ready and tested solution for this. [20]

Continuous Monitoring and Logging: Implement monitoring and logging mechanisms to detect and alert on unusual activities. This can be multiple failed login attempts from the same IP address. [22]

4.4 Broken Object Property Level Authorization

Definition and Context: Broken Object Property Level Authorization (BOPA) is a nuanced subset of the broader Broken Object Level Authorization (BOLA) vulnerability. It occurs specifically when an application fails to adequately protect the properties of an object that a user is authorized to access. In Node.js and Express.js applications, this often arises when APIs or functions expose more data than necessary or allow unauthorized modifications to certain properties of an object. [23]

Real-World Impact: The impact of BOPA can be significant. As an example, for Node.js application: an authenticated user might try to access their user profile object but should not be able to view or modify certain sensitive properties like other users' email addresses or passwords. If these properties are not properly protected, it could lead to data breaches and privacy violations. An example of this is a USPS attack, one of the largest data breaches in history that affected 60 million users. The breach was caused by a Broken Object Property Level Authorization attack, allowing anyone to access a USPS database. [24][25]

Mitigation Strategies:

Fine-Grained Access Control: Implement granular access controls that not only check if a user can access an object but also which specific properties they are allowed to view or modify. [26]

Data Sanitization: Ensure that any data sent to the client is appropriately sanitized and stripped of sensitive properties that the user should not access.

Use unpredictable values: Using random IDs minimizes the risk of bruteforce for the path properties, ensuring that only users that have the correct ID can access the resource [24]

Best Practices in Node.js and Express.js:

Use Middleware for Access Control: In Express.js, write custom middleware functions that handle access control checks for object properties based on the user's role and permissions. [27]

4.5 Unrestricted Resource Consumption

Definition and Context: Unrestricted Resource Consumption, is often referred to as a Denial of Service (DoS) vulnerability. It occurs when an application doesn't control the allocation and use of its resources correctly. In the context of Node.js and Express.js applications, this can happen when an application allows users to consume large amounts of server resources. This will lead to a higher server response time or complete unavailability. [28]

Real-World Impact: Resource Consumption vulnerability can be exploited with various methods. Few examples are sending numerous resource complicated requests to the server, uploading large sized files, or executing resource-consuming operations. For example, an attacker can send a suppressing number of requests to a Node.js server, causing it to throttle or crash the whole application, causing denial of service to all legitimate users. In April of 2022 distributed denial-of-service (DDoS) attack targeted a cryptocurrency platform with more than 15.3 million requests per second, as reported by the service provider Cloudflare. This attack was particularly powerful due to its use of HTTPS requests, which are more compute-intensive than HTTP requests, as every request must also be signed by the sender. The scale and method of this attack highlight the evolving nature of DDoS threats, emphasizing the need for robust security measures in digital platforms. [29, 30]

Mitigation Strategies:

Rate Limiting: Implement server-side rate limiting in the application to control the number of requests that a user can send in each time frame. This helps in mitigating brute-force attacks and reduces the risk of server overload. [31]

Input Validation: Ensure that the strict validation of user input is present, especially for resource consuming operations. Limiting the size of file uploads and the complexity of requests will also minimize risks. [32]

Efficient Code and Query Optimization: Write efficient code and make sure that database queries are optimized to minimize the server load. [33]

Use of Caching and Load Balancers: Implement caching solutions to reduce the load on the server and use load balancers to distribute traffic evenly across multiple servers if multi-server solution is used. [34]

Monitoring and Alerts: Set up monitoring tools to keep track of resource usage and performance metrics. Configure alerts for unusual spikes in resource consumption. [32]

4.6 Express Security Configuration

This chapter focuses on configuring Express.js to enhance the security of web applications. Express.js, known for its flexibility and minimalism, requires careful configuration to safeguard against common web vulnerabilities.

4.6.1 Implementing TLS/HTTPS for Secure Communication

Definition and Context: Transport Layer Security or TLS is a network protocol for encrypting requests and verifying server identity. If TLS is not present, data transmitted between client and server is susceptible to Man In The Middle Attack. [35]
Real-World Impact: Lack of TLS can lead to request or response breaches, compromising user information, such as login credentials or personal data. [35]
Mitigation Strategies: Implement TLS using Node.js's https module. Acquire SSL/TLS certificates from trusted authorities like Let's Encrypt. Redirect all HTTP traffic to HTTPS to ensure secure communication. [36]

4.6.2 Securing Static File Serving

Definition and Context: Static file serving in Express.js, if not properly configured, can expose sensitive files and directories to unauthorized users. [37]
Real-World Impact: This vulnerability can lead to unauthorized access to private files, potentially leaking confidential information. [38]
Mitigation Strategies: Use "*express.static*" middleware. Define accessible directories explicitly and implement path normalization to prevent directory traversal attacks. [39]

4.6.3 Setting Security Headers with Helmet

Definition and Context: HTTP security headers in Express.js are crucial for protecting against various web-based attacks. Helmet is a middleware that helps in setting these headers. [40]
Real-World Impact: Without proper security headers, applications are vulnerable to attacks like XSS and clickjacking, leading to data theft and site defacement. [41]
Mitigation Strategies: Integrate Helmet to automatically set secure HTTP headers. Customize headers like *X-Frame-Options*, and *X-Content-Type-Options* as per the application's needs.

4.6.4 Cookie Security

Definition and Context: Cookies often store sensitive session data. Unsecured cookies can lead to data interception and manipulation. [42]
Real-World Impact: By compromising an authenitcation cookie, an attacker can gain access to user data and accounts. [43]

Mitigation Strategies: Implement *HttpOnly*, *Secure*, and *SameSite* flags for cookies. Use signed cookies to verify their integrity. Default cookies settings should not be used. [4]

4.6.5 Error Handling to Prevent Information Leakage

Definition and Context: In Express.js, improperly handled errors can inadvertently reveal sensitive information about the application's internals. [44]
Real-World Impact: Information leakage through error messages can provide attackers with insights into potential vulnerabilities within the application. [45]
Mitigation Strategies: Develop comprehensive error handling mechanisms. Customize error responses to avoid sending stack traces or internal error details to clients. [45]

4.7 Utilizing npm audit for Enhanced Security in Express.js Applications

Definition and Context

- What is *npm audit*? *npm audit* is a command-line utility accessible by default via the *npm* (Node Package Manager). This tool automatically reviews the project's dependencies to identify any known vulnerabilities in the packages that are used in the application. [46]
- Importance in Express.js: Given the extensive use of third-party packages in Express.js and Node.js applications, *npm audit* becomes a valuable tool for developers to keep their applications secure from known vulnerabilities in these dependencies. [47]

Real-World Impact

- Vulnerability Exposure: Neglecting to regularly audit and update dependencies can leave an application exposed to security breaches. Vulnerabilities in packages can be exploited by attackers to gain unauthorized access, inject malicious code, or disrupt service.
- Case Studies: Real-world incidents, such as the event-stream incident, highlight the importance of regular dependency audits. In this case, a widely used package was compromised, affecting numerous applications. In 2018 a widely used *npm package* was compromised to steal cryptocurrency. This *npm package* is essential for numerous applications including those developed with Express.js, was

compromised stealing account details and private keys from accounts having a balance of more than 100 Bitcoin or 1000 Bitcoin Cash. [48]

Mitigation Strategies

- Regular Audits: Integrate *npm audit* into the development workflow. Run it regularly, ideally as part of continuous integration processes, to ensure timely detection of vulnerabilities. [46]
- Understanding Audit Reports: Learn to interpret the audit report generated by *npm audit*. It categorizes vulnerabilities by severity (low, moderate, high, critical) and provides information on the affected packages and available fixes. [46]
- Updating Dependencies: Follow the recommendations provided by *npm audit* to update or replace vulnerable packages. Use commands like *npm update* and *npm audit fix* to automate the resolution of these vulnerabilities. [46]

4.8 Avoiding Default Credentials and Utilizing Environmental Variables

Avoiding Default Credentials: Using default credentials in development environments, can pose significant security risks. These credentials are easily guessable and widely known, making them a prime target for attackers. [49]

Using Environmental Variables: Storing sensitive credentials like API keys or database passwords in environmental variables is a recommended practice. This approach enhances security by keeping sensitive data out of the codebase. [50]

Mitigation Strategies

Implementing Strong, Unique Credentials: Always replace default credentials with strong, unique passwords and keys. This should be a standard practice in both development and production environments.

Secure Storage and Access: Utilize environmental variables for storing sensitive data. Ensure that these variables are securely configured and accessible only by the necessary parts of the application.

Best Practices in Node.js and Express.js

In addition to the above strategies, it's crucial to adhere to best practices specific to Node.js and Express.js development:

Environmental Variable Management: Use tools like *dotenv* for managing environmental variables in Node.js applications. This helps in keeping configuration separate from code and makes it easier to manage different settings for development, testing, and production environments. [51]

Avoid Hardcoding Sensitive Data: Never hardcode sensitive information like API keys or database credentials directly in the source code.

5 PDF Guide Creation Using Figma

Initial Planning and Structure

The process began with a clear vision of the document's structure. The vulnerabilities were categorized into three main sections: Secure Baseline Configuration of Express.js, Best Security Practices in Node.js, and Secure API Logic Implementation. This categorization provided a logical flow and made the document more user-friendly.

Content Development and Organization

For each vulnerability, a three-question format was adopted: 'What?' to describe the vulnerability, 'Why?' to explain the associated risks, and 'How?' to outline mitigation strategies. This approach ensured that each section was informative and practical. The content was meticulously gathered and organized to ensure clarity and coherence.

Designing in Figma

Using Figma, the layout was designed to be intuitive and engaging. The design emphasized readability and ease of navigation, with a focus on a clean and professional aesthetic.

Incorporating Design Elements:

- Yellow Rounded Forms: These were used to highlight important notices and clarifications, ensuring they stood out for quick reference.

- Good and Bad Examples: To aid in understanding, some vulnerabilities were accompanied by side-by-side comparisons of good and bad practices. This visual representation helped in illustrating the practical implications of each vulnerability.

- Code Examples: Real-world code examples were integrated into the guide. These examples provided practical insights into how the vulnerabilities could be addressed in actual development scenarios.

Linking to Resources

The document included hyperlinks to official resources and guides. These links offered readers the opportunity to delve deeper into topics and access additional information.

Finalizing and Exporting the Document

Once the content and design were finalized, the document was exported from Figma as a PDF. Special attention was given to maintaining the integrity of the design elements and ensuring that all links were functional. The final PDF was reviewed for quality assurance, ensuring that it met the intended educational and practical purposes.

Outcome

The resulting PDF guide serves as an easy-to-follow resource for developers and IT professionals. The structure and design if the document make it easy to read. Practical examples and additional resources help to understand how issues can be solved to enhance the security of web applications built with Express.js and Node.js.

6 Summary

This thesis embarked on a comprehensive journey to explore and address the critical aspects of securing web applications developed using the Express.js framework, within the context of Node.js environments. The primary focus was on identifying common vulnerabilities, understanding their implications, and providing practical mitigation strategies.

Key Highlights of this thesis are the following:

- Express.js and Node.js Overview: The thesis began with a detailed exploration of Node.js and Express.js, highlighting their popularity, flexibility, and the security challenges inherent in their use. The choice of these technologies was justified based on their widespread adoption and relevance in modern web development.

- Vulnerability Analysis: A significant portion of the thesis was dedicated to analyzing common vulnerabilities. This analysis was guided by the OWASP Top 10 API Security Risks – 2023, along with specific vulnerabilities pertinent to Node.js and Express.js. The vulnerabilities were categorized into three main areas: Secure Baseline Configuration of Express.js, Best Security Practices in Node.js, and Secure API Logic Implementation.

- Practical Mitigation Strategies: For each identified vulnerability, the thesis provided a detailed 'What?', 'Why?', and 'How?' approach. This structure helped in understanding the nature of each vulnerability, its potential risks, and the most effective strategies for mitigation. Real-world examples, code snippets, and comparative analyses of good and bad practices enriched this section.

- Express.js Security Configurations: The thesis delved into specific configurations and practices for securing Express.js applications. This included using TLS, managing static file serving, implementing security headers, handling cookies securely, and managing dependencies.

- PDF Guide Creation: A unique aspect of this thesis was the creation of a comprehensive PDF guide using Figma. This guide encapsulated all the findings and recommendations of the thesis in a visually engaging and accessible format, complete with interactive elements, visual aids, and practical examples.

- Contribution and Audience: The thesis contributed significantly to the field of cybersecurity, particularly in the context of Express.js applications. It serves as a valuable resource for developers, cybersecurity professionals, and students in the field of Cyber Security Engineering.

Conclusion:

The thesis successfully achieved its objectives by providing a thorough understanding of the security landscape surrounding Express.js and Node.js applications. The research and methodologies employed were robust, ensuring that the findings were relevant, practical, and could be readily applied in real-world scenarios. The creation of the PDF guide further extended the usability of this research, making it a practical tool for ongoing reference and application.

In conclusion, this thesis stands as a testament to the importance of cybersecurity in web application development and provides a solid foundation for developers and professionals seeking to enhance the security of their Express.js applications.

References

[1] Electronic Privacy Data Center, “Equifax Data Breach”, [Online]. Available: https://archive.epic.org/privacy/data-breach/equifax. [Accessed 20 November 2023].

[2] NPM Blog, “Details about the event-stream incident”, 27 November 2018. [Online]. Available: https://blog.npmjs.org/post/180565383195/details-about-the-event-stream-incident. [Accessed 20 November 2023].

[3] OWASP, “OWASP Top 10 API Security Risks – 2023”, [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0x11-t10. [Acessed 15 October]

[4] Express, “Production Best Practices: Security”, [Online]. Available: https://expressjs.com/en/advanced/best-practice-security.html. [Acessed 15 October]

[5] Node.js, “Node.js Security Best Practices”, [Online]. Available: https://nodejs.org/en/guides/security. [Acessed 15 October]

[6] Node.js oficial website, “About Node.js”, [Online]. Available: https://nodejs.org/en/about. [Accessed 20 November 2023]

[7] Node.js oficial website, “Learn Node.js”, [Online]. Available: https://nodejs.org/en/learn. [Accessed 20 November 2023]

[8] EPAM Blog, “Top 5 NodeJS Pros and Cons: What They Mean for Your Business”, 11 November 2023. [Online]. Available: https://anywhere.epam.com/business/node-js-pros-and-cons. [Accessed 20 November 2023]

[9] NPM, “About npm”, 11 November 2023. [Online]. Available: https://www.npmjs.com/about. [Accessed 20 November 2023]

[10] Stackoverflow, “2023 Developer Survey”, [Online]. Available: https://survey.stackoverflow.co/2023/#most-popular-technologies-language. [Accessed 20 November 2023]

[11] Convective, “Javascript cross-platform”, [Online]. Available: https://www.convective.com/javascript-cross-platform. [Accessed 20 November 2023]

[12] Snyk, “Top 10 Node.js Security Best Practices”, [Online]. Available: https://snyk.io/learn/nodejs-security-best-practice. [Accessed 20 November 2023]

[13] NPM, “express”, [Online]. Available: https://www.npmjs.com/package/express. [Accessed 20 November 2023]

[14] Express.js website, “Companies using Express in production” , [Online]. Available: https://expressjs.com/en/resources/companies-using-express.html. [Accessed 20 November 2023]

[15] OWASP, “API1:2023 Broken Object Level Authorization”, [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization. [Acessed 10 November]

[16] Imperva, “Broken Object Level Authorization”, [Online]. Available: https://www.imperva.com/learn/application-security/broken-object-level-authorization-bola. [Accessed 8 November]

[17] OWASP, “API2:2023 Broken Authentication”, [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0xa2-broken-authentication. [Acessed 10 November]

[18] Contrast Security, “Broken Authentication”, [Online]. Available: https://www.contrastsecurity.com/glossary/broken-authentication. [Accessed 8 November]

[19] OWASP, “A2:2017-Broken Authentication”, [Online]. Available: https://owasp.org/www-project-top-ten/2017/A2_2017-Broken_Authentication. [Acessed 10 November]

[20] Stack Hawk, “NodeJS Broken Authentication Guide: Examples and Prevention”, [Online]. Available: https://www.stackhawk.com/blog/nodejs-broken-authentication-guide-examples-and-prevention. [Accessed 8 November]

[21] Passport.js, “Overview”, [Online]. Available: https://www.passportjs.org/concepts/authentication. [Accessed 8 November]

[22] Appsignal, “Best Practices for Logging in Node.js”, [Online]. Available: https://blog.appsignal.com/2021/09/01/best-practices-for-logging-in-nodejs.html. [Accessed 8 November]

[23] OWASP, “API3:2023 Broken Object Property Level Authorization”, [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0xa3-broken-object-property-level-authorization. [Acessed 10 November]

[24] APISEC, “What is Broken Object Level Authorization (BOLA) and How to Fix It”, [Online]. Available: https://www.apisec.ai/blog/broken-object-level-authorization. [Acessed 10 November]

[25] The Verge, “USPS took a year to fix a vulnerability that exposed all 60 million users’ data”, [Online]. Available: https://www.theverge.com/2018/11/22/18107945/usps-postal-service-data-vulnerability-security-patch-60-million-users. [Acessed 15 November]

[26] Wallarm, “Broken Object Level Authorization (BOLA)”, [Online]. Available https://www.wallarm.com/what/broken-object-level-authorization. [Acessed 10 November]

[27] Pieces, “Building a Role-Based Access System in Node.js”, [Online]. Available https://code.pieces.app/blog/role-based-access-systems-in-nodejs. [Acessed 10 November]

[28] OWASP, “API4:2023 Unrestricted Resource Consumption”, [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0xa4-unrestricted-resource-consumption. [Acessed 10 November]

[29] Paloato, “What is a denial of service attack (DoS) ?”, [Online]. Available: https://www.paloaltonetworks.com/cyberpedia/what-is-a-denial-of-service-attack-dos. [Acessed 10 November]

[30] Arstechnica, “One of the most powerful DDoSes ever targets cryptocurrency platform”, [Online]. Available: https://arstechnica.com/information-technology/2022/04/one-of-the-most-powerful-ddoses-ever-targets-cryptocurrency-platform. [Acessed 10 November]

[31] Reflectoring, “How to Implement API Rate Limiting in a Node.js Express Application”, [Online]. Available: https://reflectoring.io/tutorial-nodejs-rate-limiter. [Accessed 3 November]

[32] Thesmartscanner, “How to Secure your NodeJs Express Javascript Application - part 2”, [Online]. Available: https://www.thesmartscanner.com/blog/how-to-secure-your-nodejs-express-javascript-application-part-2. [Accessed 11 November]

[33] Medium, “Node.js on Nitro: Unleashing Blazing Performance and Invincible Security”, [Online]. Available: https://smit90.medium.com/node-js-on-nitro-unleashing-blazing-performance-and-invincible-security-%EF%B8%8F-55b7d0b519bd. [Accessed 11 November]

[34] Medium, “Node.js on Nitro: Unleashing Blazing Performance and Invincible Security”, [Online]. Available: https://medium.com/@vishwasacharya/scaling-node-js-applications-for-high-traffic-best-practices-da96b030d745. [Accessed 11 November]

[35] Cloudflare, “What is TLS (Transport Layer Security)?”, [Online]. Available: https://www.cloudflare.com/learning/ssl/transport-layer-security-tls. [Accessed 11 November]

[36] Adamtheautomator, “How to Create an HTTPS NodeJS Web Service with Express”, [Online]. Available: https://adamtheautomator.com/https-nodejs. [Accessed 11 November]

[37] Snyk, “Directory Traversal”, [Online]. Available: https://security.snyk.io/vuln/npm:server-static:20180226. [Accessed 11 November]

[38] OWASP, “Path Traversal”, [Online]. Available: https://owasp.org/www-community/attacks/Path_Traversal. [Acessed 10 November]

[39] Express.js, “Serving static files in Express”, [Online]. Available: https://expressjs.com/en/starter/static-files.html. [Acessed 10 November]

[40] Helmet.js, “Get started”, [Online]. Available: https://helmetjs.github.io. [Acessed 10 November]

[41] LogRocket, “Using Helmet in Node.js to secure your application”, [Online]. Available: https://blog.logrocket.com/using-helmet-node-js-secure-application. [Acessed 10 November]

[42] MDN Web Docs, “Using HTTP cookies”, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies. [Acessed 10 November]

[43] Malcare, “Cookie Stealing in WordPress: Understanding the Risks and Consequences”, [Online]. Available: https://www.malcare.com/blog/cookie-stealing/. [Acessed 10 November]

[44] OWASP, “Improper Error Handling”, [Online]. Available: https://owasp.org/www-community/Improper_Error_Handling. [Acessed 10 November]

[45] Sematext, “Node.js Error Handling Made Easy: Best Practices On Just About Everything You Need to Know”, [Online]. Available: https://sematext.com/blog/node-js-error-handling. [Acessed 10 November]

[46] NPM Docs, “npm-audit”, [Online]. Available: https://docs.npmjs.com/cli/v9/commands/npm-audit. [Acessed 10 November]

[47] NPM Docs, “Auditing package dependencies for security vulnerabilities”, [Online]. Available: https://docs.npmjs.com/auditing-package-dependencies-for-security-vulnerabilities. [Acessed 10 November]

[48] NPM Blog, “Details about the event-stream incident”, [Online]. Available: https://blog.npmjs.org/post/180565383195/details-about-the-event-stream-incident. [Acessed 10 November]

[49] CWE, “CWE-1392: Use of Default Credentials”, [Online]. Available: https://cwe.mitre.org/data/definitions/1392.html. [Acessed 18 November]

[50] Progress Telerik, “Beginner’s Guide to Environment Variables”, [Online]. Available: https://www.telerik.com/blogs/beginners-guide-environment-variables. [Acessed 18 November]

[51] Node Package Manager, “dotenv”, [Online]. Available: https://www.npmjs.com/package/dotenv. [Acessed 18 November]

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Maksim Semjonov

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Securing Web Applications Built with Express.js”, supervised by Priidu Paomets
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

04.12.2023

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – PDF Guide

The PDF guide that was created as an end product of this thesis is available at https://drive.google.com/file/d/1FykAOOIgt3Dgfxoc8uKVf8FdjsuonHrB/view?usp=sharing