Security Vulnerabilities in Modern Web Frameworks

Md Faiz[1], Udit Agrawal[2]. Prof. KM Ikra[3]
[1,2,3]*Department of Computer Science & Engineering, Galgotias University, Greater Noida, India*

***Abstract*—The way we develop web applications has changed dramatically since the introduction of front end frameworks like React and Angular, as well as back end frameworks such as Django, Laravel, and Express.js. With these frameworks, developers are able to use advanced routing systems, state management tools, and built in access control functionality to create applications quickly and consistently. In addition to providing these features and helping to standardize design patterns, these frameworks also allow developers to develop more efficiently. This means faster development cycles and less need for ongoing code maintenance. Even though the majority of web applications created with the above mentioned frameworks include basic inbuilt security functionalities, the applications may still be vulnerable to high levels of risk. These vulnerabilities are often due to factors such as poor coding practices, misconfigurations, and an excessive dependence on third party libraries and not because the frameworks themselves contain security vulnerabilities. In this research paper, we discuss the most common types of attack vectors (XSS, CSRF, SQL/NoSQL injection, SSTI, and RCE) and supply chain threats found in package managers. We conclude that most security breaches occur as the result of developer error and not due to design flaws within the frameworks.**

***Index Terms*—Web security, vulnerabilities, modern web frameworks, XSS, CSRF, API security, supply chain attacks**

I. INTRODUCTION

The World Wide Web has evolved over time from being sim- ply a medium for static information, to becoming a complex interactive platform that acts as an integral part of the modern global infrastructure. The World Wide Web is now critical to many of the day-to-day functions of finance, medicine, e-commerce, and public administration.

Because web-based applications are available around the clock, and they receive user-input from sources that are not authenticated, they present a much larger potential area of vulnerability compared to local software. It is well documented that the majority of all data breaches are caused by weak- nesses specific to the web, such as injection attacks, broken authentication, and vulnerabilities in application programming interfaces (APIs).

In order to deal with the complex nature of both coding and cybersecurity, developers are beginning to use modern frameworks such as Express.js, Next.js, etc. These modern frameworks help to create consistency by providing designers and developers with pre-defined design templates that are re- usable blocks. Additionally, the majority of modern frame-works include built-in security features that are specifically de- signed to prevent common coding vulnerabilities. Examples of features that provide developers with greater security resilience include secure middleware, automatic escaping to prevent injection attacks and built-in Token-based CSRF protection.

The number of security breaches that occur as a result of developing framework-based applications has been consistently increasing despite advances in technology. Cyberattacks demonstrate that implementing a new framework will not provide a complete safety net for developers who do not properly configure their settings, use safe programming conventions, or understand the security implications of their frameworks. Additionally, this risk can be further compounded if the developer improperly integrates external libraries and dependencies.

The increasing complexity of system architecture is one of the primary challenges facing modern-day software development. The introduction of microservices and various API technologies such as REST and GraphQL have changed the face of the internet, but while these innovations provide improved agility and performance for systems, they also add additional layers of complexity that make it difficult

for developers to create comprehensive security models. Furthermore, as a result of the massive number of third-party dependencies and the various package management tools available, the threat landscape continues to grow and provide additional opportunities for attackers.

In this work, we will examine how developers make choices resulting in an abundance of vulnerabilities in web applications despite having such advanced development tools and abstractions available today. By systematically analysing numerous different attack types and the responses developed by modern technology, we show that both the decisions made by developers as well as supply-chain vulnerabilities increase the overall risk. Furthermore, the goal of this study is to provide an overview of today's security challenges and recommend strategies to improve the safety and durability of web applications.

II. BACKGROUND AND MOTIVATION

When the first web platforms were created, they were mostly static content and had limited server-side logic. Because of this, security features were not typically considered at the time of design; however, as web technologies began to grow in capabilities, malicious actors began to discover and exploit the systemic weaknesses in web applications through multiple large-scale security breaches caused by the exploitation of SQL injection and XSS vulnerabilities. At this time, it became clear that the lack of input validation is no longer a minor design mistake, but rather a significant systemic risk.

The creation of web frameworks has allowed for a move away from fragmented scripts toward a more structured and maintainable approach to software development. With the increasing need for security, web frameworks started to include native protection against the most common types of attacks. Some examples of these improvements include ORM (Object- Relational Mapping) layers that make it possible to create database queries in a safe manner, template engines that automatically escape user input to prevent XSS attacks, and middleware architectures that give developers a point for implementing security logic.

Analysis of real security updates indicates that a majority of vulnerabilities are caused by a limited number of com- mon errors, specifically: missing validation, encoder errors, and logical mistakes in authentication. This implies that the persistence of security risks is not due to a shortage of defensive technology, but is instead a consequence of how these mechanisms are implemented in practice.

When secure configurations are provided by default, many engineers end up bypassing or working around those security measures due to specific performance goals or improved end user experience. For instance, when working with CSRF tokens a developer might turn them off for the sake of ease of use, and when using raw SQL instead of an ORM layer developers might skip to raw SQL from ORM for faster development. The use of unsafe rendering techniques to enable dynamic and complex content has also increased greatly among developers in recent years. All of these inten- tional bypasses of best practices greatly reduce the automatic protection afforded by the framework.

As we rely on third party libraries for the majority of mod- ern web development we have created a large bill of goods to buy into the risk of supply chain attacks. Developers utilizing libraries downloaded from public repositories to speed up their work are incidentally expanding their attack surface area with no intention of creating additional risk. A vulnerability in a very popular library could be exploited widely, and the risk of having malicious code injected into the build process dramatically increases the risk of compromising the complete system.

The increased use of API-driven development combined with modern frontend frameworks has shifted the security of applications away from a single server point. There is now an increasing need for the security of both the client-side logic and the cloud infrastructure in addition to protecting the backend. Therefore, Developers must manage bearer tokens and granular access control over the use of their application and encrypt everything.

There is a disparity between the potential of a framework for providing security and the actual security status of applications developed using that framework. By researching this disparity, developers can develop better approaches to security, make more informed decisions regarding the default security settings

for the next generation of frameworks, and be able to identify vulnerabilities and apply methods for reducing their impact with greater reliability and

accuracy than is currently possible.

III. THREAT MODEL

Structured threat models must be developed before analyz- ing vulnerabilities at the application level. In order to ensure the study reflects present-day threats, the frameworks used for this research are based on application-layer models that emulate actual exploits currently occurring in society.

This research describes an external hacking method where the attacker utilizes web-based protocols (HTTP/S) to connect with the application's servers and execute attacks. This method allows the attacker to create false requests, alter any type of information sent or received from the server (including headers), and use tools to automate their actions. The scope of this model is limited to remote network attacks, explicitly omitting threats that require physical hardware access or internal administrative credentials. Users are categorized within different groups due to the rate of access to the system. External and unauthenticated actors do not have a valid login name and target their attention to an interface that is vulnerable. Authenticated attackers have valid accounts but limited privileges and strive to obtain unau- thorized privilege escalation. Finally, internal threats include either leaked user credentials or authorized users that abuse their access and exploit application logic.

The adversary objectives that were examined within the context of this research include access to unauthorized data and bypassing authentication as well as taking advantage of functional logic errors. The threats that are also evaluated in the research are client side script execution (XSS) and remote code execution (RCE) on the server. The model also acknowl- edges what can be called an indirect type of attack vectors, in which the third-party dependencies or vulnerabilities of the application in the continuous integration and deployment pipelines are used to compromise the application.

New security issues are presented in modern framework de- signs because of their complexity. Frontend rendering frames may divulge sensitive logic to the client, supporting DOM- based exploits. Elsewhere, the threats of backend rendering are template injection and improper data processing. The defendability of API-driven systems is further compromised by the fact that it presents the attacker with many entry points that need to be defended at each point.

The analysis also addresses supply-chain vulnerabilities, where the security of third-party code or repositories is un- dermined. These attacks carry a high impact because they abuse the "trusted" nature of external packages, enabling a single malicious update to affect thousands of downstream applications at once.

This threat model gives us a clear plan to study how modern web tools can be attacked. It shows that hackers have many different ways to break in, which proves that we need several layers of security to keep an application safe.

IV. CROSS-SITE SCRIPTING (XSS)

Cross-Site Scripting, or XSS, is a very common and risky security flaw in today's websites. It happens when a site takes information from a user and displays it on a page without cleaning it first. This mistake lets hackers sneak malicious 'code' into the page, which then runs inside the browser of anyone who visits. Once that code is running, the hacker can steal private login info or mess with the user's data.

There are three main types of XSS attacks. Reflected XSS happens when a hacker's harmful link sends a script to a website, which then bounces it right back to the user's screen. Stored XSS is more dangerous because the harmful code is saved permanently on the website like in a comment or a user profile so it can infect anyone who visits that page. DOM based XSS is a bit different; it attacks the way a website's own code handles data directly inside your browser without checking if that data is safe first.

XSS is well countermeasured through web frameworks. React, in particular, has an automated value encoding of values in JSX, but both Django and Laravel default encode HTML in their template languages. Angular also has a higher level of security with a contextual sanitization. However, developers usually compromise them to support the high-risk features like direct DOM overrides or uncooked HTML rendering to meet complex design requirements.

The majority of the problems with XSS are due to the mishandling of user input or lack of understanding of how the inbuilt protections of a framework are

implemented. Ap- plications are at risk unless they are encoded in a proper context aware fashion. XSS is one of the most serious threats to modern web development due to the level of its frequency and the harm it may cause.

V. CROSS-SITE REQUEST FORGERY (CSRF)

CSRF is a security flaw where an attacker hijacks the authority of a logged-in user to perform actions without their consent. By tricking the browser into sending a forged request to a vulnerable site, the attacker can execute sensitive operations using the victim's current session. Essentially, the application cannot distinguish between a legitimate user action and a malicious one sent by the attacker.

The threat of CSRF is most significant when applied to actions that modify a user's state, such as changing login credentials, initiating financial transactions, or altering profile settings. This vulnerability exists because web browsers are designed to automatically attach session cookies to every outbound request. Unless a secondary verification layer is present, the backend system remains unable to differentiate between an intentional user action and a fraudulent request generated by an attacker.

Two major concepts prevent CSRF: secret tokens or special headers. Larval and Django are frameworks that use a method of securing data by inserting a single-use CSRF token into forms, which a server needs to authenticate prior to a request being accepted. Other systems, such as Angular, simplify this through automatic additions of unique headers to API requests to make sure that the server knows that it is communicating with a legitimate application.

Despite these in-built defenses, the CSRF threats are still common when the developers disable the protective middle- ware, use defective token verification code, or adopt insecure API design. Moreover, the systems in which the authentication is made solely on the basis of tokens and the supporting of the strong CORS and CSRF are left vulnerable. To develop a strong defense, it is required that validation checks are applied uniformly to all endpoints with the ability to alter application state.

VI. SQL AND NOSQL INJECTION

Injection flaws arise when unverified user data is integrated into database commands without sufficient cleaning or the use of parameterized queries. Despite being one of the oldest and most documented threats in web security, SQL injection persists as a major risk, frequently exploited in contemporary software environments.

An SQL Injection can allow hackers to tamper with the logic of the database so they can by-pass the log-in process and obtain certain private information. In addition, hackers can also exploit them to perform high level admin functions. Object- Relational Mapping (ORM) solutions have significantly de- creased these risks because of their ability to effectively separate the application from the database; however, there are still instances when development engineers do not utilize the ORM to build queries or build query strings manually or create a dynamic command without validating the query against a database.

The increased usage of non-relational databases has created an additional type of attack method called NoSQL Injection. The vulnerabilities of NoSQL Injection create an opportunity for a hacker to mix unvalidated input from a user with the object they are querying, allowing them to manipulate the logic of the database to compromise it. An example of this is MongoDB, where a poorly constructed query may allow a user to bypass their identity verification and gain unauthorized entry into the database.

Injection risks are typically caused by either a lack of appropriately sanitized data, a poorly written query and/or a lack of focus on security practices in the software development process. In order to mitigate these threats, it is essential for organizations to have a defence strategy comprised of parameterized queries, rigorous input filtering and adherence to the principle of least privilege (PoLP) with regards to the database.

VII. SERVER-SIDE TEMPLATE INJECTION (SSTI)

When an application incorporates user-generated content into its Server-Side Template, then utilizes a template engine to interpret that Server-Side Template, it creates Server-Side Template Injection, or SSTI. Today, as many web applications are primarily created through modern template engines, SSTI Vulnerabilities present a serious threat to the security of these programs. If an attacker were to exploit an

SSTI Vulnerability,
there is an opportunity to modify the internal process of the application's server or to execute an unauthorized command on that remote server.
Modern template engines are extremely powerful. They permit the processing of expressions, loops, and function calls, all of which can result in a security hole if an attacker success- fully hijacks that process through an unverified user submitted payload. In some situations, a successful SSTI attack could circumvent all security measures, allowing an attacker full Remote Code Execution (RCE) on the host operating system. Although most web application frameworks promote secure practices for rendering templates, SSTI vulnerabilities pre- dominantly arise through the developer's use of dynamically compiled templates, as opposed to following standard best practices for rendering. To eliminate these vulnerabilities, de- velopers must keep their presentation layer distinct from their application logic. The only way to ensure security in these scenarios is to prevent the dynamic evaluation of templates and ensure that user-supplied data is treated solely as static
data and not executable code.

VIII. AUTHENTICATION AND AUTHORIZATION VULNERABILITIES

Web application security, specifically the areas of identity verification and access management, presents significant secu- rity risks. Systems that have a design flaw in the authentication process, are a target for an attacker to take over the identity of an authorized user. On the other hand, when a system does not enforce the boundary restrictions, users are able to access sensitive data or perform administrative functions without having any authorization.
The majority of systemic problems in the authentication process are due to insufficient requirements for password complexity and encryption of stored credentials, as well as breakdowns in session and token security. In contrast, the majority of authorization issues are attributed to a lack of consistent access verification, inconsistency with role-based permission assignments, and unintended exposure of sensitive internal object identifiers.
Although modern development frameworks provide pre- built authentication modules and access management products, many developers elect to create their own custom solutions, which create the potential for they to bypass established security protocols. Additionally, when authorization logic is distributed across an overly complex application architecture it becomes increasingly more difficult to manage and review, which increases the likelihood of important vulnerabilities being overlooked.
In order to establish effective identity and access control, there are three fundamental items needed for security. The first item is a centralized location for managing authoriza- tion policy. The second item is deploying hardened session protocols. The third item is applying advanced techniques for securing credentials. As a complement, organizations that desire a secure environment must display a commitment to continual improvement of their security posture.

IX. API SECURITY CHALLENGES

APIs are an essential part of modern web design by linking together clients, mobile applications, and servers. However, as essential as they may be to how we create web applications, they create a number of unique security challenges that often go undetected or are ignored completely by developers.
One of the ways that APIs are vulnerable is through bad implementation practices. For example, APIs may allow for unauthorized access to stored data or sensitive data, which could put both users and businesses at risk. Other common security threats for APIs are over-sharing of information (i.e. data that should have been kept private) and a lack of proper request throttling (i.e. blocking certain requests that exceed specified limits). All of these threats stem from weakly validating user input and having bad permission controls on. As a result, developers may inadvertently expose user records to the internet.
While web frameworks (e.g., Django, Ruby on Rails) pro- vide developers with the necessary routing/URL registration and middleware support to develop APIs, most developers are not able to secure their APIs by default rather than developing custom security measures for each individual endpoint.
To create a robust security posture for your API, you must actively adopt industry best practices, establish a "least privilege" philosophy, create an effective identity verification process, and ensure

proper logging of audit trails.

X. SUPPLY CHAIN AND DEPENDENCY ATTACKS

”While modern web development relies on external libraries and plugins to speed up production, these dependencies create major security gaps. Supply chain exploits abuse the trust placed in third-party code to launch wide-reaching attacks against numerous applications simultaneously.”

Attackers can compromise the software ecosystem by incorporating malicious scripts into common, popular libraries, by exploiting legacy dependencies with known vulnerabilities, or by compromising the integrity of build-process tools. Such interventions are unusually dangerous, since they bypass traditional defensive perimeters and at the same time they risk an enormous number of downstream uses.

The traditional security assessments are prone to fail in detecting risks that are related to third-party dependencies. Detection of these threats requires a thorough management plan that involves a continuous audit of the libraries used to write the code, version pinning to avoid automatic updates, and security-oriented build pipelines. Moreover, the organizations should also be actively monitoring the vulnerability disclosures to be able to respond swiftly to the recently found vulnerabilities in their software supply chain.

XI. MITIGATION STRATEGIES

All web applications must implement a layered defensive depth strategy during the entire development lifecycle to mitigate security risks. Since default framework settings do not adequately secure applications, developers should integrate security throughout the build process.

Developers should implement rigorous coding standards and conduct both automated scans and manual testing to ensure secure coding practices. Hardening the application configura- tion is equally important as is leveraging specialized tools for identifying and remediating vulnerabilities within third-party libraries.

“Developers must cultivate security literacy in order to reduce the risk of vulnerability due to human error. This human-centric approach must be complemented by continu- ous supervision of the system, thorough audit trails, defined policies for incident response and recovery to provide the necessary resiliency for an application against attack.”

XII. LIMITATIONS AND FUTURE WORK

The current research article does provide insight into trends in web security, but there are some existing constraints to consider when interpreting the research results, as well as providing guidance for future research.

The first limitation of this research is it was done from mainly a qualitative perspective. This means that this research involved reviewing existing literature, past reports of security concerns, and security measures offered by the various frame- works being analyzed instead of measuring how often people’s use of a given framework has resulted in a loss or theft of data or a breach of their system through its various vulnerabilities on a larger scale.

The speed of change in the development of web technolo- gies will significantly affect the outcome of all the findings presented in this research as these frameworks continue to be updated and improved with additional security measures over time, thus all vulnerabilities that are currently exploited, will be mitigated. Additionally, the emergence of new types of exploits and methods for exploiting known vulnerabilities will continue to change the landscape and create difficulties in measuring and comparing how secure a given framework is today versus how it will be in the future.

The next steps of work can also be aimed at the stabilization of new architectural trends, such as microservices, serverless computing, and API-first designs. With the increasing distri- bution of web applications, research is required to know how the mechanisms of security in frameworks can be extended at the boundaries of services.

Lastly, future studies ought to consider developer-oriented methods of security, including better security documentation, better framework defaults, and even embedding security edu- cation into development tools. The human aspect of security is also an issue that is vital to mitigate the vulnerability of contemporary web frameworks.

XIII. CONCLUSION

The modern web frameworks have revolutionized the web application building by having powerful abstraction, general- ized structure and inbuilt security systems. React, Angular, Django, Larval, Express.js and Next.js frameworks make the enigma of development significantly simpler and promote best practices. However, as it has been demonstrated in this paper,
the adoption of the modern frameworks cannot guarantee the security of web applications.

The paper has addressed a wide range of security vulner- abilities of the current web structures including Cross-site Scripting (XSS), cross-site Request Forgery (CSRF), SQL and NoSQL Injection, server-side template injection (SSTI), authentication and authorization vulnerability, insecure API, and supply chain attacks. In the analysis, most of the weak- nesses are not founded on the shortcomings of frameworks, but instead, the misuse by programmers, insecure environments and the abuse of the built-in structure features.

A larger attack surface has been brought by the increased complexity of modern web applications and infrastructure, both based on client-side rendering, API-driven designs, mi- croservices, and large dependency networks. Frameworks are not as self-defending as the developers wish them to be in spite of the fact that they provide a basic degree of security primitives. The dependency-based supply chain risks also make the security situation more complicated by offering the weaknesses under the control of the application developers.

This paper indicates the importance of the holistic approach to web applications security. In order to develop securely, it is necessary to combine good framework characteristics, secure coding under control, automated security testing, dependency management that is careful and constant monitoring. Security is an issue that should be integrated in the software develop- ment lifecycle rather than being a post hoc.

In conclusion, it is found that modern web frameworks are deemed a necessary beginning point in the development of secure web applications but cannot be termed as a solution. The security concerns of the modern web development can be worked out through the regularity of research, the creation of improved tools, and the increased attention to the awareness of the developers. By employing multi-layered approach to security, organizations can go far, in reducing the risk of vulnerability and developing strong-headed web applications that can stand the resilience of the emerging cyber threats.

REFERENCES

[1] J. Fonseca, N. Seixas, M. Vieira, and H. Madeira, "Analysis of Field Data on Web Security Vulnerabilities," *IEEE Transactions on Depend-able and Secure Computing*, vol. 11, no. 2, pp. 89–100, 2014.

[2] J. Fonseca, M. Vieira, and H. Madeira, "Evaluation of Web Security Mechanisms Using Vulnerability and Attack Injection," *IEEE Transac- tions on Dependable and Secure Computing*, vol. 11, no. 5, pp. 440–453, 2014.

[3] D. R. Sahu and D. S. Tomar, "Analysis of Web Application Code Vulnerabilities using Secure Coding Standards," *Arabian Journal for Science and Engineering*, vol. 41, no. 11, pp. 4335–4353, 2016.

[4] D. H. Curie, J. Jaison, J. Yadav, and J. R. Fiona, "Analysis on Web Frameworks," *Journal of Physics: Conference Series*, vol. 1362, no. 1, Art. no. 012114, 2019.

[5] OWASP Foundation, "OWASP Top Ten Web Application Security Risks," 2023. [Online]. Available: https://owasp.org

[6] OWASP Foundation, "OWASP API Security Top 10," 2023. [Online]. Available: https://owasp.org/www-project-api-security/

[7] M. Dalton, C. Kozyrakis, and N. Zeldovich, "Nemesis: Preventing Authentication and Access Control Vulnerabilities in Web Applications," in *Proc. USENIX Security Symposium*, 2010.

[8] A. Razzaq, K. Latif, H. F. Ahmad, A. Hur, Z. Anwar, and P. C. Bloodsworth, "Semantic Security Against Web Application Attacks," *Information Sciences*, vol. 254, pp. 179–194, 2014.

[9] S. Bandhakavi, N. Tiku, W. Pittman, S. T. King, P. Madhusudan, and M. Winslett, "Vetting Browser Extensions for Security Vulnerabilities with VEX," *Communications of the ACM*, vol. 54, no. 9, pp. 91–99, 2011.

[10] G. Wassermann and Z. Su, "An Analysis Framework for Security in Web Applications," in *Proc. ACM SIGSOFT FSE*, 2004, pp. 70–78.

[11] D. Stuttard and M. Pinto, *The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws*, 2nd ed. Indianapolis, IN, USA: Wiley, 2011.
[12] National Institute of Standards and Technology (NIST), "Guide to Secure Web Applications," NIST Special Publication 800-53, 2020.
[13] MITRE, "Common Weakness Enumeration (CWE)," [Online]. Avail- able: https://cwe.mitre.org
[14] MITRE, "Common Vulnerabilities and Exposures (CVE)," [Online]. Available: https://cve.mitre.org
[15] Snyk Ltd., "State of Open Source Security," 2023. [Online]. Available: https://snyk.io
[16] Gartner, "Application Security Testing Market Guide," 2022.
[17] D. Fett, R. Ku¨sters, and G. Schmitz, "The Web SSO Standard OpenID Connect: In-Depth Formal Security Analysis," in *Proc. IEEE CSF*, 2014.
[18] D. Mellado, E. Ferna´ndez-Medina, and M. Piattini, "A Common Criteria Based Security Requirements Engineering Process for the Development of Secure Information Systems," *Computer Standards & Interfaces*, vol. 29, no. 2, pp. 244–253, 2007.