SECURITY OF FRONT-END APPLICATIONS: THREATS AND METHODS OF PROTECTION

Maslov Andrey

Bachelor, Tomsk Polytechnic University, Russia, Tomsk

Abstract. This article examines vulnerabilities in the client side of web applications, substantiates their causes and suggests methods of protection and prevention. With the development of digitalization, user data is becoming increasingly vulnerable to various types of attacks, which makes the topic of security especially relevant.

Keywords: Web, protect, threats, attack.

Introduction

In recent years, web applications have become increasingly complex and interactive, leading to an increase in potential entry points for attackers. The importance of ensuring the security of user data cannot be overestimated, as data leakage or compromise can lead to serious consequences for both users and companies.

So, the book The Web Application Hacker's Handbook [2] describes the exploitation of vulnerabilities that we will consider in this article.

Table 1.

The main threats

Type of threats	Short description
Cross-site Scripting (XSS)	vulnerability of web applications that allows the introduction malicious scripts in the user's browser
Cross-site Request Forgery (CSRF)	vulnerability in which an attacker forces a user to perform u on a trusted site on which he is logged in
Data leakage via JavaScript	vulnerability in which confidential information is inadverten through scripts
Attacks through library and framework vulnerabilities	attacks in which attackers exploit weaknesses in third-part frameworks used in an application to gain unauthorized acce system
Clickjacking	an attack in which attackers hide malicious interface elemen ones, forcing users to accidentally click on them and perform

Cross-site scripting (XSS)

Cross-site scripting (XSS) is a vulnerability in which attackers inject malicious code into the pages of a web application, allowing them to execute arbitrary scripts in the context of the user. This vulnerability can lead to identity theft, hijacking of sessions, redirection of users to phishing sites and other types of attacks. To protect against XSS, it is critically important to apply validation and

data entry screening, as well as use the Content Security Policy (CSP).

Validation of data entry is the first and most important step in protecting against XSS. It involves checking all incoming data to make sure that they match the expected values.

- *Server and client validation:* Always perform validation both on the client side (in the browser) and on the server side. Client validation helps prevent the user from sending incorrect data, but it cannot be relied on completely, since an attacker can bypass it by sending requests directly to the server. Server-side validation is necessary to ensure application-level security.
- *Using whitelists:* The use of whitelists is preferable before blacklists. The whitelist defines a set of acceptable values, while the blacklist tries to exclude malicious data, which may be less effective due to the possibility of missing some dangerous values.
- *Data Type Checking:* Make sure that the input data matches the expected types. For example, if the field should contain a number, make sure that the user entered the number, and not a line of text or HTML code.
- *Data Length limit:* Set limits on the length of input data to prevent buffer overflows and other potential attacks.

Data escaping is the process of replacing special characters (e.g. spaces, quotes) in data with their equivalents before displaying them in the browser.

Create and configure a content security policy for your web application. This can be done by adding the Content-Security-Policy header to the HTTP response. Content Security Policy (CSP) is a powerful mechanism that allows developers to control the resources that can be downloaded and executed on a web page. CSP helps prevent XSS by limiting the ability to execute unintended scripts.

Cross-site Request Forgery (CSRF)

CSRF (Cross-Site Request Forgery) is an attack in which an attacker fraudulently forces a user to perform an undesirable action on a site on which he is authenticated. This can lead to undesirable operations, such as changing the password, transferring money, or deleting the account. Various methods are used to protect against CSRF, such as using CSRF tokens, checking the referrer, and applying the SameSite attribute to cookies.

CSRF tokens are unique, random values that are generated by the server and included in forms and requests. They help to make sure that the request comes from the user, and not from an attacker.

- *Token generation:* During each user session, the server generates a unique CSRF token and stores it in the user's session. This token is also included in every form or request that requires authentication.
- *Token validation:* When the server receives a request, it verifies the presence and correctness of the CSRF token. If the token is missing or does not match the token stored in the session, the request is rejected.
- *Token Shielding:* To prevent XSS attacks, CSRF tokens must be shielded before being inserted into HTML. For example, using functions to escape HTML characters (&, <, >, ", ') when inserting the token into the form.

Referrer Verification - checks the Referer (or Origin) header in requests to verify that requests originate from trusted pages.

- Comparing the referrer with acceptable values: The server verifies that the Referer or Origin header matches the expected domain. If the header value does not match, the request is rejected.
- Problems with Referrer: It is important to keep in mind that some browsers and proxy servers may not send the Referrer header, so referrer verification should be used in combination with other security methods.

Applying the SameSite attribute to cookies: The SameSite attribute for cookies helps to limit the

transmission of cookies only within the same site, which prevents their use on other sites.

- Configuring the SameSite attribute: Set the SameSite attribute for all cookies used for authentication. The Strict value prohibits sending cookies for any cross-site request, and Las allows sending cookies for some secure requests (for example, GET requests).

Data leakage via JavaScript: Protection methods

Data leakage through JavaScript can become a serious security threat to web applications. Insufficient protection of data transmitted through JavaScript can lead to their compromise, which, in turn, can allow attackers to gain access to confidential information, intercept sessions, carry out phishing attacks and much more. To prevent such leaks, it is necessary to apply a comprehensive approach to data protection. Let's look at the main methods of protection: minimizing the use of sensitive data in JavaScript, data encryption and strict CORS policy.

Minimizing the use of sensitive data in JavaScript:

- Separation of data and logic: Store as little sensitive data as possible in JavaScript code. If possible, transfer the processing of sensitive data to the server side. This will reduce the likelihood of information leakage, since the server code is not directly accessible to users.
- Using tokens instead of data: Instead of transmitting sensitive data, use tokens that represent this data. Tokens can be short-lived and disposable, which reduces the risk of their compromise.
- Access Control: Restrict access to data in JavaScript using strict authentication and authorization mechanisms. Make sure that only authorized users can access sensitive information.

Data encryption:

- Transport encryption: Use HTTPS to encrypt data transmitted between the client and the server. This prevents hackers from intercepting data during transmission.
- Client-side encryption: In some cases, it makes sense to encrypt data directly in the user's browser before sending it to the server. Encryption standards such as AES can be used for this.
- Storing encrypted data: If you need to store data in the browser's local storage (for example, localStorage or sessionStorage), make sure that the data is encrypted before saving.

CORS:

- Configuring CORS on the server: Make sure that the server is configured to accept requests only from trusted domains. This can be done by configuring CORS (Cross-Origin Resource Sharing) headers.
- Restriction of methods and headers: Set restrictions on HTTP methods (GET, POST, etc.) and headers that can be used for cross-domain requests.

Attacks through vulnerabilities of libraries and frameworks: Methods of protection

Using outdated or vulnerable libraries and frameworks in web applications can open the way for various attacks, which poses a significant security threat. Vulnerabilities in third-party components can lead to data leakage, unauthorized access, arbitrary code execution, and other critical consequences. Let's look at the main methods of protection against such attacks: regular updates of dependencies and the use of vulnerability scanning tools.

Regular dependency updates: Make sure that the libraries and frameworks you use are always updated to the latest stable versions. Regular updates reduce the likelihood of exploiting vulnerabilities that have been fixed in newer versions.

Using vulnerability scanning tools: Use vulnerability scanning tools such as Snik, Dependabot, White Source or others during the development and build of the application. These tools analyze

dependencies and report vulnerabilities found.

Additional security measures: Use virtual environments or containers to isolate dependencies and prevent conflicts between them. It also helps to limit the impact of vulnerabilities in one component on the entire system. Provide regular training to developers on security issues and best practices for dependency management. This will help them keep up to date with new vulnerabilities and ways to prevent them.

Clickjacking

Clickjacking is an attack in which attackers inject malicious elements into the website interface, deceiving users and forcing them to perform undesirable actions. These actions may include clicking on hidden buttons, filling out forms, or other interactions that the user is not aware of. Methods such as the use of X-Frame-Options headers and Content Security Policy (CSP) are used to protect against clickjacking.

Using X-Frame-Options: The HTTP X-Frame-Options header allows developers to control how browsers should handle the inclusion of their pages in <iframe> tags. This helps prevent clickjacking attacks by preventing pages from loading on your site in frames of other sites.

Table 2.

X-Frame-Options value

X-Frame-Options value	Description
DENY	This parameter completely prevents the pa into the <iframe>
SAMEORIGIN	This option allows you to load a page in <ifr requested from the same doma
ALLOW-FROM <URI>	This option allows you to load the page in th from the specified URL. This option is less co supported by all browsers.

Conclusion: According to an article from OWASP [3], new attack methods appear every year, which requires constant improvement of protection mechanisms. Web security is an essential aspect of modern web application development, and understanding various types of attacks such as XSS, CSRF, data leakage through JavaScript, attacks through vulnerabilities in libraries and frameworks, as well as clickjacking, is the key to creating reliable and secure systems. A comprehensive approach to protecting web applications, including regular updates, the use of modern security tools and careful attention to every aspect of user interaction with the application, is the basis for creating a safe and reliable environment. It is important to remember that security is a process, not a one - time event, and continuous improvement and adaptation to new threats plays a crucial role in ensuring the protection of web applications.

References:

1. Hoffman, A. (2020). Web Application Security: Exploits and Countermeasures for Modern Web Applications.

2. Stuttard, D., & Pinto, M. (2011). The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws.

3. OWASP Foundation. (2021). OWASP Top Ten Web Application Security Risks - 2021. Retrieved from https://owasp.org/Top10/