CRASHTEST **SECURITY**

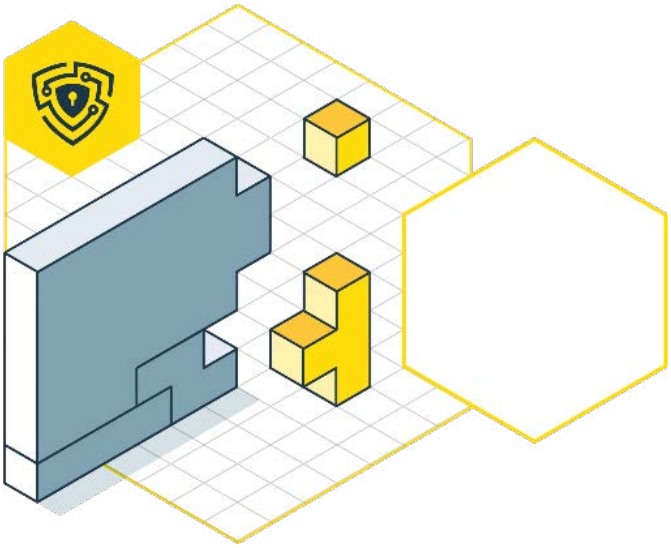
2022

GUIDE FOR PREVENTING JAVASCRIPT VULNERABILITIES

WHAT ARE THE STEPS TO KEEP YOUR WEB APP OR API SAFE FROM SUCH VULNERABILITY

GUIDE FOR THE JAVASCRIPT VULNERABILITY PREVENTION

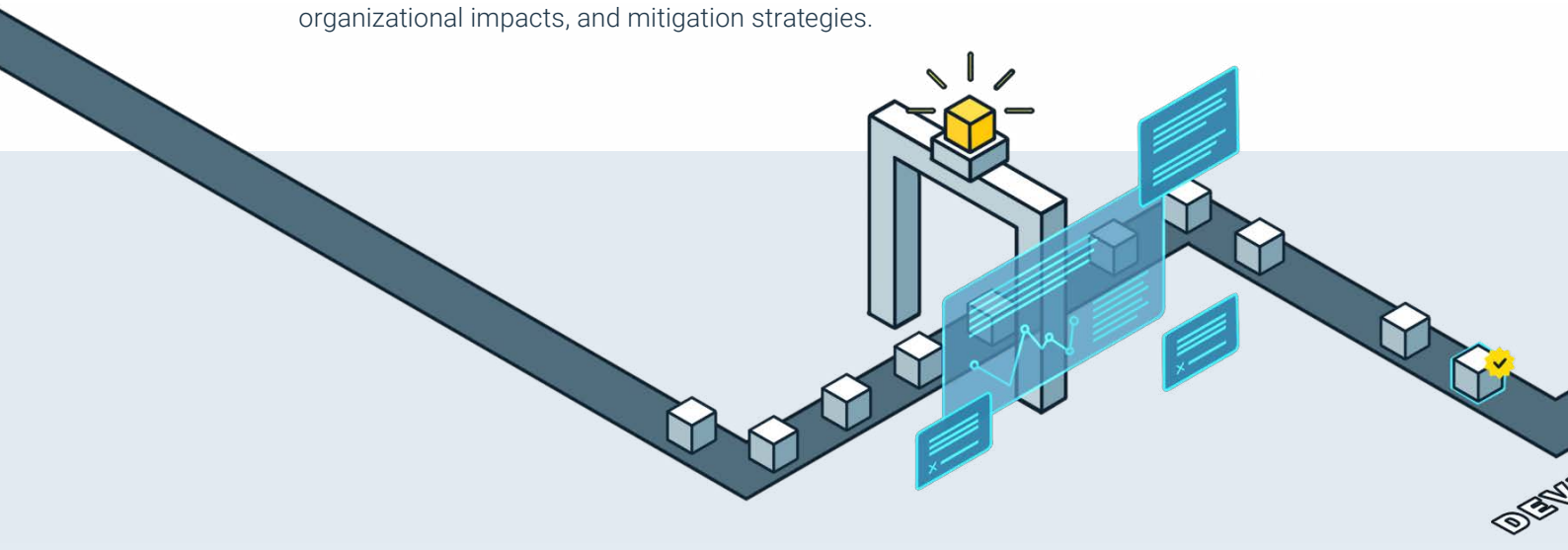
Table of Contents



INTRODUCTION TO THIS GUIDE

With the adoption of close to 98% of the world's websites, JavaScript is undoubtedly one of the popular programming languages used in modern web applications. By offering a wide range of libraries, open-source packages, and frameworks, JavaScript allows developers of any skill level to create complex websites with little overhead. While the feature-rich platform offers numerous benefits in developing dynamic websites and mobile applications, the platform is also infamous for being a prime target of attackers because of its inherent vulnerabilities.

This guide discusses common vulnerabilities in JavaScript applications, their types, organizational impacts, and mitigation strategies.



WHAT ARE JAVASCRIPT VULNERABILITIES?

As JavaScript interacts with a website's Document Object Model (DOM) to provide extended functionalities, threat actors typically exploit the security flaws of a DOM to inject malicious scripts into a web server and get those executed on client browsers. With sophisticated script execution, attackers can gain control of user accounts, compromise web systems or manipulate system/user data.

TYPES OF JAVASCRIPT VULNERABILITIES

Some inherent security flaws for Javascript applications include:

CROSS-SITE SCRIPTING VULNERABILITIES (XSS)

Cross-site scripting is an injection flaw that allows adversaries to feed corrupt client-side scripts into the web page, triggered for execution when users visit the page. Since such attacks can be orchestrated through multiple entry points where the application accepts user input, a cross-site scripting attack is usually complex to mitigate. In the absence of appropriate input validation, the end user's browser cannot verify the authenticity of a script and ends up executing it automatically. Since the browser assumes the script is trustworthy, it can access the user's session tokens, cookies, and other sensitive application data stored within the browser.

XSS-based common attacks are primarily categorized into:

- **Stored/Persistent XSS** - In such attacks, the compromised server permanently stores the script, which is then served to victims along with their requested information
- **Reflected XSS** - In these attacks, the malicious script is injected and immediately reflected off the server in a response that includes user inputs sent to the server. The malicious payload is delivered via an alternate route, such as malicious links in email messages and other websites. Since the untrusted script comes from a valid server, the browser assumes the script to be legitimate and eventually executes it.
- **DOM-based XSS** - This involves a malicious script delivered by modifying the DOM environment within the user's browser.

UNINTENDED SCRIPT EXECUTION

JavaScript embeds functions within HTML pages when interacting with a web page's DOM. Without JavaScript and HTML code encoding, attackers can access and leverage security loopholes to post unscrupulous content within these functions. As a result, every client machine that connects to the web page accesses and executes corrupt scripts.

SOURCE CODE VULNERABILITIES

These vulnerabilities are introduced when developers fail to follow secure coding practices. JavaScript relies on various open-source libraries, frameworks, and open-source packages that simplify product development. While such components reduce the development efforts, these packages are known to introduce a wide array of security vulnerabilities in the absence of appropriate validation. Using unsafe open-source packages before utilization gives rise to security inconsistencies that allow attackers to inject and execute malicious scripts.

EXPOSURE OF SESSION DATA

JavaScript applications that expose session tokens such as cookies and user session IDs, an attacker can impersonate a user and gain unauthorized access to data and resources. When session information is bundled together with the data accessed, an application may present its values to unintended entities. Some web applications also cache session data, tempting attackers to hijack sessions through compromised proxies and gateways.

CROSS-SITE REQUEST FORGERY (CSRF)

CSRF vulnerabilities allow attackers to trick legitimate users into executing malicious activities while logged in to a web application. This attack relies on the flaw where the browser automatically includes all cookies while processing requests. Since the user is already authenticated to the application, the website cannot differentiate malicious requests from legitimate ones. The attack typically begins with a social engineering technique that forces the authorized user to perform malicious acts on the hackers' behalf. The attacker uses various stealth tactics to transmit their requests, including hidden forms, specially-designed image tags, and AJAX requests.

RELYING ONLY ON CLIENT-SIDE VALIDATION

JavaScript applications typically employ client-side validation techniques to improve user experience. However, this form of validation is considered insufficient since it can be deactivated by the user, allow for workarounds, and fail if there is an error in the script. This permits attackers to fake data inputs, subsequently allowing unsafe, unsanitized data to reach the webserver.

SERVER-SIDE JAVASCRIPT INJECTION

Besides the importance of client-side validation, it is equally crucial to enforce a robust server-side validation for comprehensive protection. In instances where an application lacks sanitization and filtering mechanisms of user-controlled data, attackers can inject and execute arbitrary code on the webserver. These vulnerabilities often occur in functions that parse unvalidated user inputs, including scripts that the server can execute. The attackers typically deploy malicious code under the context of file system interaction and server configuration, which allows for partial or outright compromise of the host server.

CLIENT-SIDE LOGIC VULNERABILITIES

JavaScript engines enable fast and efficient client-side processing. At the same time, the feature acts conducive to rapid application development practices, and client-side processing, however, it grants the user complete control over the application's logical decision-making. Instances where sensitive operations are included on the client-side, a compromised user device enables adversaries to quickly access and control the web application's behavior through the browser.

JAVASCRIPT VULNERABILITIES - WHAT ARE THE IMPACTS?

Attackers primarily leverage JavaScript vulnerabilities to orchestrate client-side attacks. While JavaScript is a client-side application programming language, flaws in frameworks can also facilitate server-side attacks. Once a server is compromised, the attacker can inject arbitrary code into legitimate scripts, allowing them to access data on user behavior and browsing context. In websites that lack user input and output validation, the server may potentially execute these malicious scripts, with far-reaching impacts including:

- **Uncontrolled client application changes** - As JavaScript relies on multiple third-party frameworks and code libraries, malicious actors can push unwanted features within third-party code to expose application data and system configuration.
- **Exposure of sensitive information** - When an application invokes a third-party script, the browser connects directly with third-party servers. Requests to these servers typically contain sensitive information such as the referrer, previous session cookies, and the browser's original IP address. This typically grants the third-party website access to data on the application, organization, and users. Once attackers gain access to such information, they can further compromise the web server's sanity and integrity.
- **Execution of malicious code at the client-side** - When invoking and integrating packages into a website, developers rarely review the sanity of third-party code libraries. Once the client request reaches the webserver hosting the third-party code, the registered user's permissions and privileges are susceptible to malicious exploits. Executing unchecked third-party code also reduces the validity of any tests performed before entering production.

IDENTIFYINGJAVASCRIPT VULNERABILITIES WITH CRASHTEST SECURITY

The Crashtest Security Suite offers a wide range of API and web application vulnerability scanners to help reduce risk exposure. The platform provides an automatic scanner to identify and remediate inherent JavaScript attacks such as JavaScript injections, CSRF and XSS.

Crashtest Security blends into development pipelines through multiple seamless integrations to save time and money on extensive JavaScript penetration testing. The security suite allows cross-functional teams to perform security analysis within minutes, enabling a comprehensive white-box approach to pen-testing.

JAVASCRIPT VULNERABILITIES - PREVENTION TECHNIQUES

Some methods to reduce the attack surface for JavaScript applications include:

USER INPUT VALIDATION

Sanitizing the type and content of user-supplied data helps prevent the storage and execution of malformed scripts in downstream application components. User inputs should be validated when entering the data flow, ensuring the application does not accept malicious scripts that can generate malformed data. Aggregated data received from external entities should be subject to input validation, which helps prevent injection attacks, XSS, and other Javascript attack vectors. Input validation should also be performed at semantic and syntactic levels to ensure only admissible data structures, acceptable characters, and values are used within the browsing context.

Strategies to implement input validation include:

- **Data type validators bundled within development frameworks**
- **Schema-based validation**
- **Exception handling for type conversion**
- **Value range checks**
- **Regular expressions**

CSRF TOKENS

A server typically generates and stores a token linked to the user session. This token is included as a validation voucher when a client submits a request. The server then compares the token included in the user request with the one stored. If the values do not match, the web server rejects the request. This makes it difficult for hackers to construct a valid HTTP request that the server can execute. CSRF tokens are always generated with a Pseudo-Random number generator seeded with a timestamp, making it unpredictable with significant entropy. This facilitates additional security by restricting hackers from analyzing tokens based on existing samples they have obtained.

CONTENT SECURITY POLICIES (CSPS)

Most attackers use JavaScript source vulnerabilities to bypass the same-origin policy. A CSP provides an additional layer of protection against these attackers by providing directives that specify which data sources can interact with the web app. Such directives are specified in HTTP response headers that define permitted sources for web assets. Once a CSP header is specified, the browser restricts content execution from sources not included in the CSP whitelist. Some directives specified in CSPs include:

- **default-src:** the default directive that defines the fallback policy for most directives
- **script-src:** provides a whitelist of script sources
- **style-src:** defines the sources of CSS stylesheets
- connect-src: permitted sources for direct connections that use ***WebSocket***, ***EventSource*** and ***XMLHttpRequest*** objects
- **object-src:** controls the sources of plugins
- **img-src:** specifies sources of images
- **font-src:** specifies target sources for loading fonts

USER INPUT ENCODING/ESCAPING

JavaScript vulnerability attacks are typically orchestrated by supplying input with special characters executed by the web page's Javascript, CSS, or HTML. Encoding and escaping are common approaches to stopping this injection form by transforming user-supplied into a unified format. These transformations ensure multiple systems safely share and consume data while helping user inputs precisely interpreted into equivalent forms.

SUBRESOURCE INTEGRITY CHECKING

It is essential to check the integrity of the third-party and external scripts before they are fetched from their host servers for execution. Modern web browsers support Subresource Integrity (SRI) checking that verifies the validity of external scripts using a **cryptographic hash** for external JavaScript files. The hash value is generated using a command-line tool or specialized SRI hash generator, which is then added to HTML code by embedding it to the ***integrity*** attribute of the ***<link>*** or ***<script>*** element.

BEST PRACTICES IN PREVENTING JAVASCRIPT ATTACKS

Some recommended practices to avoid JavaScript vulnerability attacks include:

AVOID USING THE EVAL() FUNCTION

The ***eval()*** function is a global JavaScript function that evaluates an input string as JavaScript code before it is marked for execution. However, the function is considered unsafe for web applications since a malicious user can include an arbitrary script within input strings. Developers should avoid user input evaluation or parse JSON data using the ***eval()*** function as a recommended practice.

AVOID TRANSMITTING CSRF TOKENS IN SESSION

Transmitting CSRF tokens within cookies makes them susceptible to exposure since attackers can intercept and access session cookies. When using request submission forms, CSRF tokens are recommended to be included as hidden fields or headers in AJAX calls.

OBFUSCATE JAVASCRIPT CODE

Attackers can only abuse scripts if they understand the structure and logic. Therefore, a basic line of defense is to keep the details of code implementation unknown to unauthorized users. Code obfuscation involves modifying the code's executables to remain functional while hidden from attackers.

ENFORCE SAFE DOM MANIPULATION METHODS

Some JS methods, such as ***innerHTML***, do not limit values parsed onto them. This allows attackers to embed unsafe characters at the data input points. Developers are recommended to leverage mechanisms that can escape or encode any potentially malicious content, helping to protect the application against DOM-based XSS attacks.

CRASHTEST SECURITY'S VULNERABILITY SCANNER FOR SAFER WEB APPLICATIONS

As one of the critical approaches to mitigating modern application security risks, identifying and countering JavaScript vulnerabilities is often considered the first step. To support this, Crashtest Security Suite helps reduce security vulnerabilities on web applications, APIs, and JavaScript with its automated vulnerability scanner. The suite is built to scan a wide range of JavaScript attacks, including XSS, CSRF, and HTTP host header attacks, to eliminate blind spots that potentially allow for the execution of malicious scripts.

Crashtest Security benchmarks application security against the OWASP top 10 to enhance the security posture and add immediate value to development, security, and QA efforts. Run your first test with a free trial today and see how the platform can help mitigate security risks by administering robust security on your JS applications.

Start 2-Week Trial for Free

CRASHTEST **SECURITY**

WWW.CRASHTEST-SECURITY.COM