CRASHTEST **SECURITY**

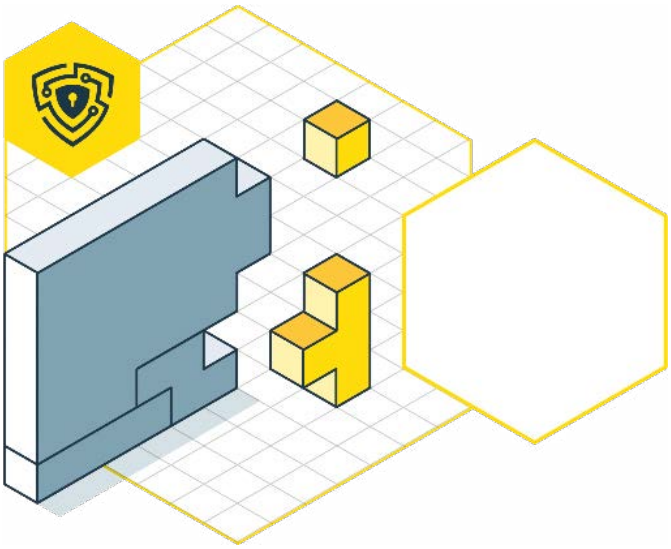
2022

GUIDE FOR PREVENTING CSRF ATTACKS

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

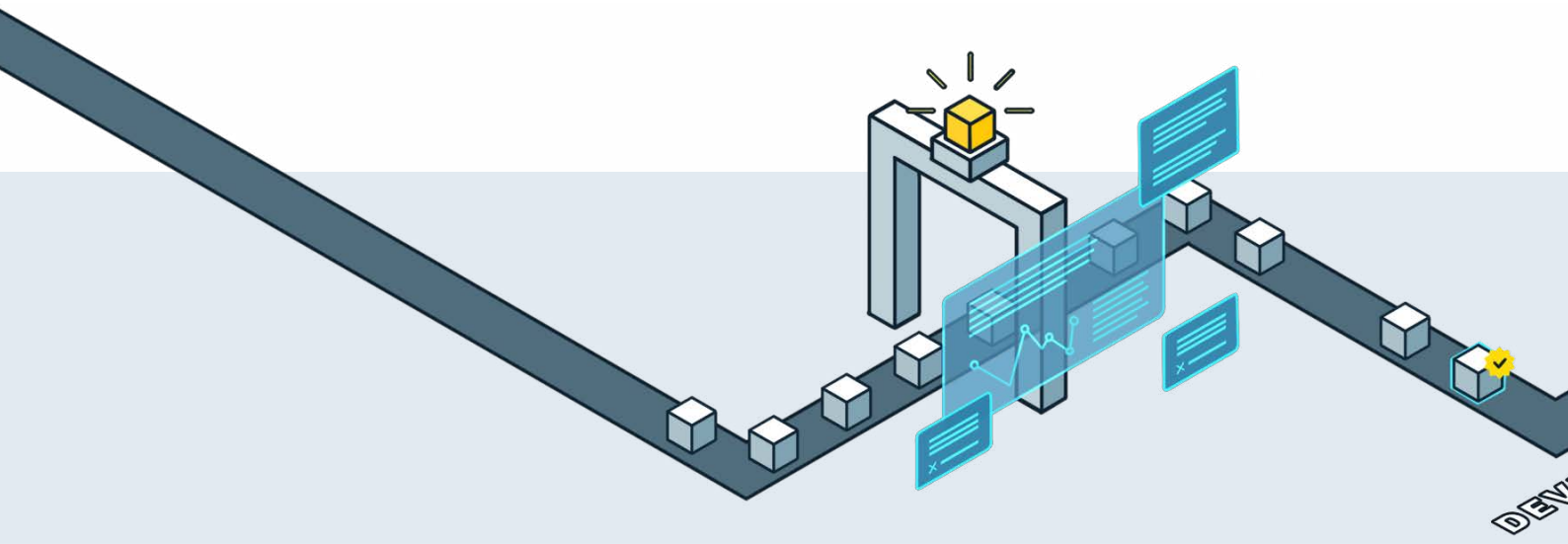
GUIDE FOR THE CSRF PREVENTION

Table of Contents



INTRODUCTION TO THIS GUIDE

Modern web applications rely on requests to retrieve and send resources the user wants to access. These requests often include credentials associated with the website, allowing the user to maintain a connection throughout the browsing session. **Cross-Site Request Forgery (CSRF)** is a commonly exploited web application vulnerability that modifies these requests and forces users into loading sensitive information from the web application. This guide discusses what a Cross-Site Request Forgery vulnerability is, how CSRF attacks are typically orchestrated, and best practices to prevent such attacks.



WHAT IS CSRF?

A cross-site request forgery is a common form of web-security attack. The threat actor forces authenticated users to send malicious requests to the website that would execute the hacker's intended actions. The attack, also known as session riding or a one-click attack, violates the same-origin policy by allowing the attacker partial or complete control of user sessions. Since the victim remains the recipient of the server's response, most CSRF attacks do not target data extraction; instead, they intend to interfere with the application's behaviour by targeting state-changing functionality.

A successful CSRF attack is dependent upon several factors, including:

- **Cookie-Based Session Handling** - If the application relies solely on cookies to validate the origin of requests, an attacker can orchestrate social engineering attacks to assume the user's identity and further exploit it to submit malicious requests.Installing malicious software on the user's machine
- **Predictable Request Parameters** - By conveniently speculating or obtaining values of the parameters used by a specific request, attackers can craft malicious requests to trigger unstable functionalities of an application.
- **Relevant Action** - A web application contains critical actions such as modifying privileged users' permissions or changing user-specific data like passwords or other account details. Exploiting such critical actions of a web application allows an attacker to induce sophisticated attacks that affect the entire application stack.

These vulnerabilities typically occur on application elements that accept unsanitized user input, included in the web server's dynamic response.

HOW CSRF EXPLOITS ARE DELIVERED

There are two primary approaches to encourage victims to submit the malicious requests:

As an HTML Payload

In this approach of request forgery attack, the hacker embeds the **unwanted action as HTML** in a website that they control and would wait for the user to visit their website. They also use **social engineering techniques** such as sending the link to users' emails, chats, social media messages, or forum comments on popular websites, thereby encouraging victims into clicking the malicious link leading to the website. Once the victim clicks on the link, the web server treats all requests from the attacker's website as legitimate requests as the session gets authenticated with the affected user's session cookie.

Using Vulnerable HTTP Methods

In the vulnerable HTTP method approach, the exploit is **self-contained** and can directly provide users with the malicious URL on the vulnerable website. The attacker employs the **GET** method and designs a **single URL with negative parameters** to transfer and execute malicious actions.

For an application on the vulnerable domain (***darwin.com***), a money transfer operation using GET requests for a user Adam would look similar to:

GET http://darwin.com/transfer.do?acct=ADAM&amount=100 HTTP/1.1

If an attacker, *Maria*, wants to trick the average user *Adam* into transferring $50,000 to her account, the beneficiary and amount parameters in the URL would be changed as below to form the malicious URL:

http://darwin.com/transfer.do?acct=MARIA&amount=50000

Social engineering techniques like unsolicited emails with the HTML content or as script/exploit URL on the bank's website can be further exploited to trick Adam into loading this URL. A sample exploit URL delivered as a hidden iframe would look similar to:

<img src="http://darwin.com/transfer.do?acct=MARIA&amount=50000" width="0" height="0" border="0">

CSRF exploits are also delivered using other HTTP methods such as **PUT, DELETE,** and **POST** requests.

WHAT IS THE SEVERITY LEVEL OF CSRF ATTACKS?

In the OWASP 2021 Top 10 list of vulnerabilities, CSRF is categorized under the **Broken Access Control** failures vulnerability, positioned at number one in the top ten list of security vulnerabilities. The impacts of a successful attack depend on the privileges of the victim user and assets exposed by the fake request. Few effects of CSRF attacks include:

- Complete account takeover
- Fraud/Unauthorized fund transfer
- Data breach
- Loss of income, trust, and reputation
- Complete compromise of the application's functionality
- Modification of account credentials

Prevalence of the CSRF vulnerability is considerably common in modern web applications. Attackers also often leverage the fact that most web applications use predictable parameters for actions. However, using appropriate code analysis and penetration testing techniques, the detectability of a CSRF vulnerability is relatively uncomplicated. Since such an identification technique requires **multi-stage delivery of payloads** and is well documented, a CSRF vulnerability's overall severity is typically regarded as of ***average exploitability***.

HOW CRASHTEST SECURITY YHELPS TO IDENTIFY AND MITIGATE CSRF VULNERABILITIES

The Crashtest Security Suite delivers many features to help deal with CSRF vulnerabilities. These include:

AUTOMATED VULNERABILITY SCANNING

Crashtest Security ships with an **integrated CSRF scanning tool** that autonomously examines the application to uncover CSRF vulnerabilities within web applications and APIs. The automated CSRF scanner **embeds directly** into the development workflow, enabling development, QA, and security teams to get started with scanning in no time. The scanner **integrates with available chat notification** systems to provide real-time alerts in case of discovered vulnerabilities.

Crashtest Security **continuously benchmarks** the website's security posture against the **OWASP top ten** web application security risks, facilitating the elimination of known vulnerabilities that can be used alongside CSRF in a chain attack. The security suite implements a **trusted scanning process** with low false positives and negatives for enhanced scanning confidence.

CONTINUOUS PENETRATION TESTING

Crashtest Security offers **full-stack penetration testing** to map the actions of threat actors once they have discovered CSRF vulnerabilities. The platform follows a **black box testing pattern** to examine the application's security landscape the same way an external entity would. These tests **expose security misconfigurations** that can be used for CSRF attacks, **explore the path taken** by the attackers and **provide identification** and **mitigation** measures to prevent attacks.

ENFORCE PROPER OUTPUT ENCODING

Crashtest Security **automates vulnerability scanning** and **penetration testing** to eliminate the manual effort required in identifying and mitigating CSRF security gaps. Automation enables the development, security, and QA teams to build **rapid** and **high-quality** web applications, servers, and APIs with trusted security. Additionally, the platform outputs **actionable reports** that can be shared across development teams, clients, and executives for a secure, streamlined CI/CD workflow.

CSRF PREVENTION TECHNIQUES

Though prevention techniques may differ based on use cases, here are a few CSRF attack prevention approaches:

SYNCHRONIZER TOKEN PATTERN

The synchronizer token pattern utilizes a token, secret and unique ID to verify the origin of a request. Whenever the server receives a request from the user, it generates a session token to save session data within the authentication cookie. The server returns the session token within a hidden field in an HTML form that the user later uses as a part of their request. The web application compares the token saved in its storage with the request tokens of a user-submitted hidden form. A match indicates the request is from an authenticated user. As all synchronizer tokens are secret, unpredictable, and unique for each session, tokens prevent CSRF attacks as the attacker fails to orchestrate a valid request without a token.

DOUBLE-SUBMIT COOKIE TECHNIQUE/ STATELESS CSRF DEFENSE

On receiving a request, the server generates a unique cryptographically strong pseudo-random value, different from the session identifiers and stored as a cookie on the user's system. All subsequent user sign-ins to the site should include this token as a hidden value either within the request or as another request parameter. If the cookie value and request value don't match on the server side, the request is treated as illegitimate. As per the same-origin policy, this technique prevents attackers from modifying the cookie value or reading request data sent from the server.

USING STANDARD HEADERS TO VERIFY REQUEST ORIGIN

The technique involves using server-side validation by checking the request's header to ensure that the request's source origin matches the target origin. If the source and target origin do not match, the request violates the same-origin policy and is discarded. The source and target origins belong in the forbidden headers category that enables only browsers to set them and is restricted from being altered programmatically. A source origin is determined by checking the origin request header, which contains the origin URL's scheme, hostname,e, and porL. On the other hand, the target origin is determined in a number of ways:

- Setting the target origin value in a server configuration entry
- Using the x-forwarded-host header value
- Using the host header value

USING CUSTOM REQUEST HEADERS

Using CSRF tokens within custom request headers offers a more robust defense mechanism against CSRF attacks as it enforces the same-origin policy restriction. Modern browsers do not support custom headers to be transported with Cross-Domain requests by default. Custom headers can only be added in JavaScript or within the script's origin. This CSRF mitigation technique is stateless and requires no changes to the user experience, making it particularly useful for CSRF mitigation on a REST service.

UI-BASED MITIGATION OPTIONS

Few techniques require users to authenticate themselves to prevent CSRF attempts during the active session. Some user-interaction based CSRF attack mitigation techniques include:

- One-time passwords
- CAPTCHA-based authorization
- Multi-factor authentication with device-based access
- Re-authentication mechanisms

SAMESITE COOKIE ATTRIBUTE

This approach uses a cookie as an attribute in the Set-Cookie response header that allows developers to define whether a cookie is set for the same-origin or cross-origin requests. These attributes can be set to three values:

- **Lax** - Lax session cookies include cookies for cross-site requests only if the request uses the GET method or the user navigated from the origin site using a link.
- **Strict** - Strict cookies are only set for a first-party context (same origin) requests but aren't initiated for cross-origin requests.
- **None** - Cookies are sent in requests initiated by both first-party and third-party contexts.

BEST PRACTICES FOR PREVENTING CSRF ATTACKS

Almost all modern programming languages/frameworks include out-of-the-box defenses for preventing Cross-Site Request Forgery attacks. This section explores CSRF protection best practices for major web development platforms.

CSRF PREVENTION IN PHP

Using synchronizing tokens in PHP applications is a common approach to prevent CSRF attacks. To use the synchronizing token, initially a one-time token is created that is added to the $_SESSION variable

$_SESSION[‚token‘] = md5(uniqid(mt_rand(), true));

This token is later inserted as a hidden value into the HTML form as shown below:

<input type=“hidden“ name=“token“ value=“<?php echo $_SESSION[‚token‘] ?? ‚‘ ?>“>

Once the user submits the form, it is checked to see if the token exists in the INPUT_POST. The application then compares this value with the current session token $_SESSION[‘token’], as shown:

```
<?php

$token = filter_input(INPUT_POST, ‚token‘, FILTER_SANITIZE_STRING);

if (!$token || $token !== $_SESSION[‚token‘]) {
   // return 405 http status code
   header($_SERVER[‚SERVER_PROTOCOL‘] . , 405 Method Not Allowed‘);
   exit;
} else {
   // process the form
}
```

If the token does not match or fails to exist, the PHP application returns a 405 HTTP error and exits.

CSRF PREVENTION IN WORDPRESS

WordPress integrates with several anti-CSRF plugins to strengthen CSRF prevention. These include:

Comment form CSRF Protection - The plugin adds secure tokens to comment forms and runs a validation check before accepting comments for the site, ensuring input forms are secure.

Anti CSRF plugin - This plugin offers robust protection from hacking attacks by enforcing a hidden value that is sent with a user’s cookies and requests while establishing a secure connection.

WP CSRF Protector - This plugin protects the WordPress administrator’s panel from CSRF attacks by automatically calling methods to attach a nonce to the HTML output.

CSRF PREVENTION IN PYTHON

Python does not ship with in-built protection against CSRF attacks. However, the platform supports the use of extensions to mitigate such vulnerabilities. The **Flask framework**, for instance, includes a **WTF** extension to enable CSRF protection.

The snippet below shows the script used to protect a Flask application from CSRF:

```
From flask_wtf.csrf import CSRFProtect
csrf= CSRFProtect(app)
```

The command enables global protection for Flask application endpoints using the default Flask app's **SECRET_KEY** to sign the CSRF token securely.

START AUTOMATED TESTING AND SCANNING TODAY

The Crashtest Security Suite implements vulnerability scanners to automate the testing of web applications, APIs, and JavaScript testing to help spot common web application security risks. The suite offers rapid security assessment that enables C-level management, security, and DevOps teams to swiftly assess and mitigate potential security exploits.

Contact us here to know more and start a free,14-day trial of the Crashtest Security platform.

Start 2-Week Trial for Free

CRASHTEST **SECURITY**

WWW.CRASHTEST-SECURITY.COM