# CWE Detail – CWE-1385

## Description

The product uses a WebSocket, but it does not properly verify that the source of data or communication is valid.

## Extended Description

WebSockets provide a bi-directional low latency communication (near real-time) between a client and a server. WebSockets are different than HTTP in that the connections are long-lived, as the channel will remain open until the client or the server is ready to send the message, whereas in HTTP, once the response occurs (which typically happens immediately), the transaction completes. A WebSocket can leverage the existing HTTP protocol over ports 80 and 443, but it is not limited to HTTP. WebSockets can make cross-origin requests that are not restricted by browser-based protection mechanisms such as the Same Origin Policy (SOP) or Cross-Origin Resource Sharing (CORS). Without explicit origin validation, this makes CSRF attacks more powerful.

## Threat-Mapped Scoring

Score: 1.8

Priority: P4 - Informational (Low)

## Observed Examples (CVEs)

**•** CVE-2020-25095: web console for SIEM product does not check Origin header, allowing Cross Site WebSocket Hijacking (CSWH)

**•** CVE-2018-6651: Chain: gaming client attempts to validate the Origin header, but only uses a substring, allowing Cross-Site WebSocket hijacking by forcing requests from an origin whose hostname is a substring of the valid origin.

**•** CVE-2018-14730: WebSocket server does not check the origin of requests, allowing attackers to steal developer's code using a ws://127.0.0.1:3123/ connection.

**•** CVE-2018-14731: WebSocket server does not check the origin of requests, allowing attackers to steal developer's code using a ws://127.0.0.1/ connection to a randomized port number.

**•** CVE-2018-14732: WebSocket server does not check the origin of requests, allowing attackers to steal developer's code using a ws://127.0.0.1:8080/ connection.

## Modes of Introduction

**•** Architecture and Design: N/A

**•** Implementation: N/A

## Common Consequences

**•** Impact: Varies by Context, Gain Privileges or Assume Identity, Bypass Protection Mechanism, Read Application Data, Modify Application Data, DoS: Crash, Exit, or Restart — Notes: The consequences will vary depending on the nature of the functionality that is vulnerable to CSRF. An attacker could effectively perform any operations as the victim. If the victim is an administrator or privileged user, the consequences may include obtaining complete control over the web application - deleting or stealing data, uninstalling the product, or using it to launch other attacks against all of the product's users. Because the attacker has the identity of the victim, the scope of the CSRF is limited only by the victim's privileges.

## Potential Mitigations

**•** Implementation: Enable CORS-like access restrictions by verifying the 'Origin' header during the WebSocket handshake. (Effectiveness: N/A)

**•** Implementation: Use a randomized CSRF token to verify requests. (Effectiveness: N/A)

**•** Implementation: Use TLS to securely communicate using 'wss' (WebSocket Secure) instead of 'ws'. (Effectiveness: N/A)

**•** Architecture and Design: Require user authentication prior to the WebSocket connection being established. For example, the WS library in Node has a 'verifyClient' function. (Effectiveness: N/A)

**•** Implementation: Leverage rate limiting to prevent against DoS. Use of the leaky bucket algorithm can help with this. (Effectiveness: Defense in Depth)

**•** Implementation: Use a library that provides restriction of the payload size. For example, WS library for Node includes 'maxPayloadoption' that can be set. (Effectiveness: Defense in Depth)

**•** Implementation: Treat data/input as untrusted in both directions and apply the same data/input sanitization as XSS, SQLi, etc. (Effectiveness: N/A)

## Applicable Platforms

**•** None (Class: Not Language-Specific, Prevalence: Undetermined)