

**Dynamic Vulnerability Assessment**

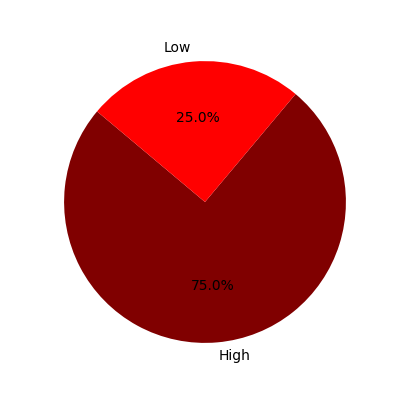
**one final test**

**Requested by:** one final test

## Version Information

|  |  |  |
| --- | --- | --- |
| Date | Application Version | Reviewer |
| 02-May-2025 | Initial Draft | one final test |
| 02-May-2025 | Peer Review |  |
| 02-May-2025 | Approved |  |

**Vulnerability Severity Distribution**



## Summary Table

|  |  |  |  |
| --- | --- | --- | --- |
| Sl. No. | Security Observation | Risk Rating | Page No. |
| **High Severity** | | | |
| 1 | Insecure Transport: Weak SSL Ciphers | High | 4 |
| 2 | Insecure Transport: Weak SSL Ciphers | High | 5 |
| 3 | HTML5: Overly Permissive CORS Policy | High | 6 |
| **Low Severity** | | | |
| 4 | Misconfigured Content-Security-Policy Header | Low | 7 |

## URLs and Scope

URLs: one final test

Scope: one final test

## Vulnerability Details

### 1. Insecure Transport: Weak SSL Ciphers

**Severity: High**

CVSS Score: 8.4

CVSS Vector: CVSS:3.1/AV:A/AC:L/PR:L/UI:R/S:C/C:H/I:H/A:H

#### Description

Improper The Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols provide a  
mechanism to help protect authenticity, confidentiality and integrity of the data transmitted between a  
client and web server. The strength of this protection mechanism is determined by the authentication,  
encryption and hashing algorithms. These are collectively known as a cipher suite chosen for the  
transmission of sensitive information over the TLS/SSL channel. Most web servers support a range of such  
cipher suites of varying strengths. Using a weak cipher or an encryption key of insufficient length, for  
example, could enable an attacker to defeat the protection mechanism and steal or modify sensitive  
information. If misconfigured, a web server could be manipulated into choosing weak cipher suites. A  
weak encryption scheme can be subjected to brute force attacks that have a reasonable chance of  
succeeding using current methods and resources. An attacker could possibly execute a man in the middle  
attack which would allow them to intercept, monitor and tamper with sensitive data. Each weak cipher  
was enumerated by establishing an SSL connection with the target host and specifying the cipher to test  
in the Client Hello message of the SSL handshake.

#### Evidence

Step 1: one final test



#### Recommendation

It is recommended not to use RC4, CBC,SHA, SHA1, MD5 etc ciphers

#### Reference

https://www.acunetix.com/blog/articles/tls-ssl-cipher-hardening/  
http://zero.webappsecurity.com

### 2. Insecure Transport: Weak SSL Ciphers

**Severity: High**

CVSS Score: 7

CVSS Vector: CVSS:3.1/AV:P/AC:H/PR:L/UI:N/S:C/C:H/I:H/A:H

#### Description

Improper The Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols provide a  
mechanism to help protect authenticity, confidentiality and integrity of the data transmitted between a  
client and web server. The strength of this protection mechanism is determined by the authentication,  
encryption and hashing algorithms. These are collectively known as a cipher suite chosen for the  
transmission of sensitive information over the TLS/SSL channel. Most web servers support a range of such  
cipher suites of varying strengths. Using a weak cipher or an encryption key of insufficient length, for  
example, could enable an attacker to defeat the protection mechanism and steal or modify sensitive  
information. If misconfigured, a web server could be manipulated into choosing weak cipher suites. A  
weak encryption scheme can be subjected to brute force attacks that have a reasonable chance of  
succeeding using current methods and resources. An attacker could possibly execute a man in the middle  
attack which would allow them to intercept, monitor and tamper with sensitive data. Each weak cipher  
was enumerated by establishing an SSL connection with the target host and specifying the cipher to test  
in the Client Hello message of the SSL handshake.

#### Evidence

Step 1: one final test



#### Recommendation

It is recommended not to use RC4, CBC,SHA, SHA1, MD5 etc ciphers

#### Reference

https://www.acunetix.com/blog/articles/tls-ssl-cipher-hardening/  
http://zero.webappsecurity.com

### 3. HTML5: Overly Permissive CORS Policy

**Severity: High**

CVSS Score: 7

CVSS Vector: CVSS:3.1/AV:L/AC:H/PR:L/UI:N/S:U/C:H/I:H/A:H

#### Description

Cross-Origin Resource Sharing, commonly referred to as CORS, is a technology that allows a domain to  
define a policy for its resources to be accessed by a web page hosted on a different domain using cross  
domain XML HTTP Requests (XHR). Historically, the browser restricts cross domain XHR requests to abide  
by the same origin policy. These restrictions are managed by access policies typically included in  
specialized response headers, such as: Access-Control-Allow-Origin, Access Control-Allow-Headers,  
Access-Control-Allow-Methods. A domain includes a list of domains that can make cross domain requests  
to shared resources in Access Control-Allow-Origin header. This header can have either list of domains or  
a wildcard character (“\*”) to allow all access. Having a wildcard is considered overly permissive policy.

#### Evidence

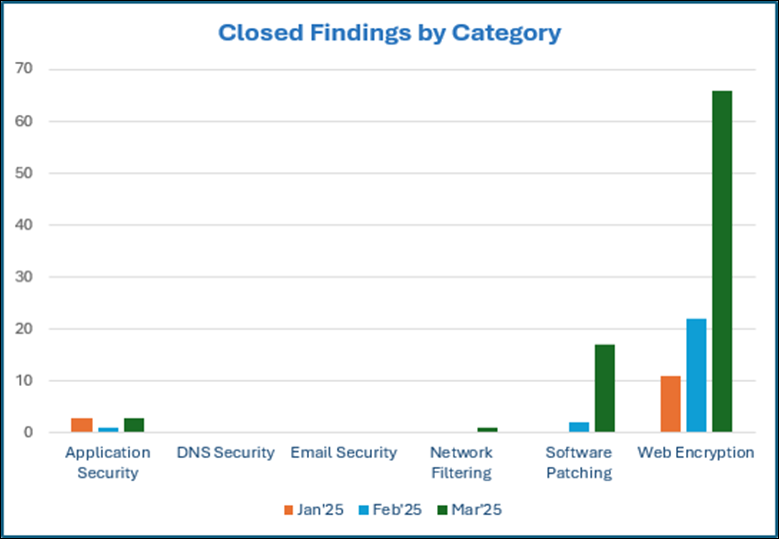
Step 1: one final test

[Image not found: ]

Step 2: one final test

[Image not found: ]

Step 3: one final test



#### Recommendation

Review the Cross-Origin-Resource-Sharing policy and consider restricting access to only trusted domains.  
Never use wildcard open-access permissions (e.g. “\*”) in the Access-Control-Allow-Origin header.  
Additionally, do not automatically include Access-Control-Allow-Origin headers in the response unless the  
request is cross-domain. Alternatively, implement a whitelist of known domains that can access this  
domain and only include domains that tried to access the resource. Otherwise, reject the request and  
reply with only host domain not exposing all allowed domains. Please note that this needs to be  
implemented throughout the application as a best practice.

#### Reference

https://en.wikipedia.org/wiki/Cross-origin\_resource\_sharing

### 4. Misconfigured Content-Security-Policy Header

**Severity: Low**

CVSS Score: 3.6

CVSS Vector: CVSS:3.1/AV:P/AC:L/PR:H/UI:R/S:U/C:L/I:L/A:L

#### Description

It is observed the CSP header value is not implemented properly in the response. Configuring Content  
Security Policy involves adding the proper CSP HTTP header to a web page and giving it values to control  
resources the user agent is allowed to load for that page. For example, a page that uploads and displays  
images could allow images from anywhere but restrict a form action to a specific endpoint. A properly  
designed Content Security Policy helps protect a page against a cross site scripting attack.

#### Evidence

Step 1: one final test



Step 2: one final test



#### Recommendation

Configuring Content Security Policy involves adding the proper CSP HTTP header to a web page and giving  
it values to control resources the user agent is allowed to load for that page. For example, a page that  
uploads and displays images could allow images from anywhere but restrict a form action to a specific  
endpoint. A properly designed Content Security Policy helps protect a page against a cross site scripting  
attack.  
It is suggested to implement CSP as per the application requirements. For example, content security policy  
can be designed as mentioned below: “Content-Security-Policy: default-src 'self’ https://Appdomain;  
child-src 'none'; object-src 'none

#### Reference

https://owasp.org/www-community/controls/Content\_Security\_Policy  
https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP