Acunetix by Invicti

Comprehensive Report

Acunetix Threat Level 3

One or more high-severity type vulnerabilities have been discovered by the scanner. A malicious user can exploit these vulnerabilities and compromise the backend database and/or deface your website.

Scan Detail

Target	https://localhost:8082/fa/v1/helloworld/restful
Scan Type	Full Scan
Start Time	Dec 31, 2021, 10:17:06 PM GMT+3
Scan Duration	2 minutes
Requests	2239
Average Response Time	10ms
Maximum Response Time	4102ms

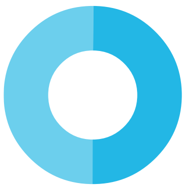
2	4	2	1
High	Medium	Low	Informational

Severity	Vulnerabilities	Instances
❗ High	2	2
❗ Medium	4	4
❕ Low	2	2
ⓘ Informational	1	1
Total	9	9

Informational

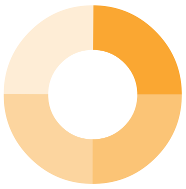
		Instances
🟩	Content Security Policy (CSP) not implemented	1

Low Severity



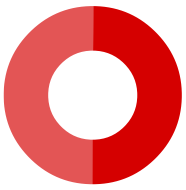
		Instances
🟦	Clickjacking: X-Frame-Options header	1
🟦	HTTP Strict Transport Security (HSTS) not imp…	1

Medium Severity



		Instances
🟧	TLS 1.1 enabled	1
🟧	TLS/SSL LOGJAM attack	1
🟧	TLS/SSL Sweet32 attack	1
🟧	Others	1

High Severity



		Instances
🟥	Cross site scripting	1
🟥	TLS 1.0 enabled	1

Impacts

SEVERITY	IMPACT	
High	1	Cross site scripting
High	1	TLS 1.0 enabled
Medium	1	TLS 1.1 enabled
Medium	1	TLS/SSL LOGJAM attack
Medium	1	TLS/SSL Sweet32 attack
Medium	1	TLS/SSL Weak Cipher Suites
Low	1	Clickjacking: X-Frame-Options header
Low	1	HTTP Strict Transport Security (HSTS) not implemented
Informational	1	Content Security Policy (CSP) not implemented

Cross site scripting

Cross-site Scripting (XSS) refers to client-side code injection attack wherein an attacker can execute malicious scripts into a legitimate website or web application. XSS occurs when a web application makes use of unvalidated or unencoded user input within the output it generates.

Impact

Malicious JavaScript has access to all the same objects as the rest of the web page, including access to cookies and local storage, which are often used to store session tokens. If an attacker can obtain a user's session cookie, they can then impersonate that user.

Furthermore, JavaScript can read and make arbitrary modifications to the contents of a page being displayed to a user. Therefore, XSS in conjunction with some clever social engineering opens up a lot of possibilities for an attacker.

---

https://localhost:8082/fa/

URI was set to **1<ScRiPt>DAw5(9196)</ScRiPt>**
The input is reflected inside a text element.

Request

```
GET /fa/v1/helloworld/"1<ScRiPt>DAw5(9196)</ScRiPt> HTTP/1.1
Referer: https://localhost:8082/fa/v1/helloworld/restful
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip,deflate,br
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/92.0.4512.0 Safari/537.36
Host: localhost:8082
Connection: Keep-alive
```

Recommendation

Apply context-dependent encoding and/or validation to user input rendered on a page

Description

In order for a Cross-site scripting (XSS) attack to take place, an attacker does not directly target a victim. Instead, an attacker exploits a vulnerability in a web application visited by a victim, where the web application is used to deliver the malicious JavaScript. The victim's browser is not able to distinguish between malicious and legitimate JavaScript, and therefore, executes the attacker's malicious payload.

Since cross-site scripting (XSS) is user input which is interpreted as code. In order to prevent XSS, secure input handling is necessary. The two fundamental methods of handling untrusted user input are

encoding and **validation**.

Encoding - Escapes user input so that browsers interpret it as **data**, not as code
Validation - Filters user input so that browsers interpret it as code without malicious commands

Encoding and validation are two different techniques to preventing cross-site scripting (XSS). Deciding which should be used highly depends on the **context** within which the untrusted user input is being inserted.

The following are two examples of the most common cross-site scripting (XSS) contexts.

```
<!-- HTML element -->
<div>userInput</div>

<!-- HTML attribute -->
<input value="userInput">
```

The method for preventing cross-site (XSS) scripting in the two examples above is different. In the first example, where user input is inserted in an HTML element, HTML encoding is the correct way to prevent XSS. However, in the second example, where user input is inserted in an HTML attribute, validation (in this case, filtering out `'` and `"`)is the appropriate prevention method.

```
<!-- Application code -->
<input value="userInput">

<!-- Malicious string -->
"><script>...</script><input value="

<!-- Resulting code -->
<input value=""><script>...</script><input value="">
```

In **most** of the time, encoding should be performed whenever user input is included in a page, however, as with the above example, in some cases, encoding has to be replaced by or complemented with validation.

It's important to remember that secure input handling has to take into account which context of a page the user input is inserted into.

References

Cross-site Scripting (XSS) Attack - Acunetix
https://www.acunetix.com/websitesecurity/cross-site-scripting/

Types of XSS - Acunetix
https://www.acunetix.com/websitesecurity/xss/

XSS Filter Evasion Cheat Sheet
https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

Excess XSS, a comprehensive tutorial on cross-site scripting
https://excess-xss.com/

Cross site scripting
https://en.wikipedia.org/wiki/Cross-site_scripting

TLS 1.0 enabled

The web server supports encryption through TLS 1.0, which was formally deprecated in March 2021 as a result of inherent security issues. In addition, TLS 1.0 is not considered to be "strong cryptography" as defined and required by the PCI Data Security Standard 3.2(.1) when used to protect sensitive information transferred to or from web sites. According to PCI, "30 June 2018 is the deadline for disabling SSL/early TLS and implementing a more secure encryption protocol – TLS 1.1 or higher (TLS v1.2 is strongly encouraged) in order to meet the PCI Data Security Standard (PCI DSS) for safeguarding payment data.

Impact

An attacker may be able to exploit this problem to conduct man-in-the-middle attacks and decrypt communications between the affected service and clients.

---

https://localhost:8082/ Confidence: 100%

The SSL server (port: 8082) encrypts traffic using TLSv1.0.

Recommendation

It is recommended to disable TLS 1.0 and replace it with TLS 1.2 or higher.

References

RFC 8996: Deprecating TLS 1.0 and TLS 1.1
https://tools.ietf.org/html/rfc8996

Are You Ready for 30 June 2018? Saying Goodbye to SSL/early TLS
https://blog.pcisecuritystandards.org/are-you-ready-for-30-june-2018-sayin-goodbye-to-ssl-early-tls

PCI 3.1 and TLS 1.2 (Cloudflare Support)
https://support.cloudflare.com/hc/en-us/articles/205043158-PCI-3-1-and-TLS-1-2

TLS 1.1 enabled

The web server supports encryption through TLS 1.1, which was formally deprecated in March 2021 as a result of inherent security issues. When aiming for Payment Card Industry (PCI) Data Security Standard (DSS) compliance, it is recommended to use TLS 1.2 or higher instead. According to PCI, "30 June 2018 is the deadline for disabling SSL/early TLS and implementing a more secure encryption protocol – TLS 1.1 or higher (TLS v1.2 is strongly encouraged) in order to meet the PCI Data Security Standard (PCI DSS) for safeguarding payment data.

Impact

An attacker may be able to exploit this problem to conduct man-in-the-middle attacks and decrypt communications between the affected service and clients.

---

https://localhost:8082/ Confidence: 100%

The SSL server (port: 8082) encrypts traffic using TLSv1.1.

Recommendation

It is recommended to disable TLS 1.1 and replace it with TLS 1.2 or higher.

References

RFC 8996: Deprecating TLS 1.0 and TLS 1.1
https://tools.ietf.org/html/rfc8996

Are You Ready for 30 June 2018? Saying Goodbye to SSL/early TLS
https://blog.pcisecuritystandards.org/are-you-ready-for-30-june-2018-sayin-goodbye-to-ssl-early-tls

PCI 3.1 and TLS 1.2 (Cloudflare Support)
https://support.cloudflare.com/hc/en-us/articles/205043158-PCI-3-1-and-TLS-1-2

TLS/SSL LOGJAM attack

The LOGJAM attack is a SSL/TLS vulnerability that allows attackers to intercept HTTPS connections between vulnerable clients and servers and force them to use 'export-grade' cryptography, which can then be decrypted or altered. This vulnerability alert is issued when a web site is found to support DH(E) export cipher suites, or non-export DHE cipher suites using either DH primes smaller than 1024 bits, or commonly used DH standard primes up to 1024 bits.

Impact

An attacker may intercept HTTPS connections between vulnerable clients and servers.

---

https://localhost:8082/

Weak DH Key Parameters (p < 1024 bits, or <= 1024 bits for common primes):

- TLS1.0, TLS_DHE_RSA_WITH_AES_128_CBC_SHA: 1024 bits (common prime)
- TLS1.0, TLS_DHE_RSA_WITH_AES_256_CBC_SHA: 1024 bits (common prime)
- TLS1.0, TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA: 1024 bits (common prime)
- TLS1.1, TLS_DHE_RSA_WITH_AES_128_CBC_SHA: 1024 bits (common prime)
- TLS1.1, TLS_DHE_RSA_WITH_AES_256_CBC_SHA: 1024 bits (common prime)
- TLS1.1, TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA: 1024 bits (common prime)
- TLS1.2, TLS_DHE_RSA_WITH_AES_128_CBC_SHA: 1024 bits (common prime)
- TLS1.2, TLS_DHE_RSA_WITH_AES_128_CBC_SHA256: 1024 bits (common prime)
- TLS1.2, TLS_DHE_RSA_WITH_AES_128_GCM_SHA256: 1024 bits (common prime)
- TLS1.2, TLS_DHE_RSA_WITH_AES_256_CBC_SHA: 1024 bits (common prime)
- TLS1.2, TLS_DHE_RSA_WITH_AES_256_CBC_SHA256: 1024 bits (common prime)
- TLS1.2, TLS_DHE_RSA_WITH_AES_256_GCM_SHA384: 1024 bits (common prime)
- TLS1.2, TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA: 1024 bits (common prime)

Recommendation

Reconfigure the affected SSL/TLS server to disable support for any DHE_EXPORT suites, for DH primes smaller than 1024 bits, and for DH standard primes up to 1024 bits. Refer to the "Guide to Deploying Diffie-Hellman for TLS" for further guidance on how to configure affected systems accordingly.

References

Weak Diffie-Hellman and the Logjam Attack
https://weakdh.org/

Guide to Deploying Diffie-Hellman for TLS
https://weakdh.org/sysadmin.html

TLS/SSL Sweet32 attack

The Sweet32 attack is a SSL/TLS vulnerability that allows attackers to compromise HTTPS connections using 64-bit block ciphers.

Impact

An attacker may intercept HTTPS connections between vulnerable clients and servers.

https://localhost:8082/

Cipher suites susceptible to Sweet32 attack (TLS1.0 on port 8082):

- TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_3DES_EDE_CBC_SHA

Cipher suites susceptible to Sweet32 attack (TLS1.1 on port 8082):

- TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_3DES_EDE_CBC_SHA

Cipher suites susceptible to Sweet32 attack (TLS1.2 on port 8082):

- TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_3DES_EDE_CBC_SHA

Recommendation

Reconfigure the affected SSL/TLS server to disable support for obsolete 64-bit block ciphers.

References

Sweet32: Birthday attacks on 64-bit block ciphers in TLS and OpenVPN
https://sweet32.info/

TLS/SSL Weak Cipher Suites

The remote host supports TLS/SSL cipher suites with weak or insecure properties.

Impact

---

https://localhost:8082/

Weak TLS/SSL Cipher Suites: (offered via TLS1.0 on port 8082):

- TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (Medium strength encryption algorithm (3DES).)
- TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (Medium strength encryption algorithm (3DES).)
- TLS_RSA_WITH_3DES_EDE_CBC_SHA (Medium strength encryption algorithm (3DES).)

Weak TLS/SSL Cipher Suites: (offered via TLS1.1 on port 8082):

- TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (Medium strength encryption algorithm (3DES).)
- TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (Medium strength encryption algorithm (3DES).)
- TLS_RSA_WITH_3DES_EDE_CBC_SHA (Medium strength encryption algorithm (3DES).)

Weak TLS/SSL Cipher Suites: (offered via TLS1.2 on port 8082):

- TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (Medium strength encryption algorithm (3DES).)
- TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (Medium strength encryption algorithm (3DES).)
- TLS_RSA_WITH_3DES_EDE_CBC_SHA (Medium strength encryption algorithm (3DES).)

Recommendation

Reconfigure the affected application to avoid use of weak cipher suites.

References

OWASP: TLS Cipher String Cheat Sheet
https://cheatsheetseries.owasp.org/cheatsheets/TLS_Cipher_String_Cheat_Sheet.html

OWASP: Transport Layer Protection Cheat Sheet
https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html

Mozilla: TLS Cipher Suite Recommendations
https://wiki.mozilla.org/Security/Server_Side_TLS

SSLlabs: SSL and TLS Deployment Best Practices
https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices

Clickjacking: X-Frame-Options header

Clickjacking (User Interface redress attack, UI redress attack, UI redressing) is a malicious technique of tricking a Web user into clicking on something different from what the user perceives they are clicking on, thus potentially revealing confidential information or taking control of their computer while clicking on seemingly innocuous web pages.

The server did not return an **X-Frame-Options** header with the value DENY or SAMEORIGIN, which means that this website could be at risk of a clickjacking attack. The X-Frame-Options HTTP response header can be used to indicate whether or not a browser should be allowed to render a page inside a frame or iframe. Sites can use this to avoid clickjacking attacks, by ensuring that their content is not embedded into untrusted sites.

Impact

The impact depends on the affected web application.

---

https://localhost:8082/

Paths without secure XFO header:

- https://localhost:8082/fa/v1/helloworld/restful

Request

```
GET /fa/v1/helloworld/restful HTTP/1.1
Referer: https://localhost:8082/fa/v1/helloworld/restful
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip,deflate,br
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/92.0.4512.0 Safari/537.36
Host: localhost:8082
Connection: Keep-alive
```

Recommendation

Configure your web server to include an X-Frame-Options header and a CSP header with frame-ancestors directive. Consult Web references for more information about the possible values for this header.

References

The X-Frame-Options response header
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options

Clickjacking
https://en.wikipedia.org/wiki/Clickjacking

OWASP Clickjacking
https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html

Frame Buster Buster
https://stackoverflow.com/questions/958997/frame-buster-buster-buster-code-needed

HTTP Strict Transport Security (HSTS) not implemented

HTTP Strict Transport Security (HSTS) tells a browser that a web site is only accessable using HTTPS. It was detected that your web application doesn't implement HTTP Strict Transport Security (HSTS) as the Strict Transport Security header is missing from the response.

Impact

HSTS can be used to prevent and/or mitigate some types of man-in-the-middle (MitM) attacks

https://localhost:8082/

URLs where HSTS is not enabled:

- https://localhost:8082/fa/v1/helloworld/restful

Request

```
GET /fa/v1/helloworld/restful HTTP/1.1
Referer: https://localhost:8082/fa/v1/helloworld/restful
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip,deflate,br
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/92.0.4512.0 Safari/537.36
Host: localhost:8082
Connection: Keep-alive
```

Recommendation

It's recommended to implement HTTP Strict Transport Security (HSTS) into your web application. Consult web references for more information

References

hstspreload.org
https://hstspreload.org/

Strict-Transport-Security
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security

Content Security Policy (CSP) not implemented

Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks.

Content Security Policy (CSP) can be implemented by adding a **Content-Security-Policy** header. The value of this header is a string containing the policy directives describing your Content Security Policy. To implement CSP, you should define lists of allowed origins for the all of the types of resources that your site utilizes. For example, if you have a simple site that needs to load scripts, stylesheets, and images hosted locally, as well as from the jQuery library from their CDN, the CSP header could look like the following:

```
Content-Security-Policy:
default-src 'self';
script-src 'self' https://code.jquery.com;
```

It was detected that your web application doesn't implement Content Security Policy (CSP) as the CSP header is missing from the response. It's recommended to implement Content Security Policy (CSP) into your web application.

Impact

CSP can be used to prevent and/or mitigate attacks that involve content/code injection, such as cross-site scripting/XSS attacks, attacks that require embedding a malicious resource, attacks that involve malicious use of iframes, such as clickjacking attacks, and others.

---

https://localhost:8082/

Paths without CSP header:

- https://localhost:8082/fa/v1/helloworld/restful

Request

```
GET /fa/v1/helloworld/restful HTTP/1.1
Referer: https://localhost:8082/fa/v1/helloworld/restful
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip,deflate,br
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/92.0.4512.0 Safari/537.36
Host: localhost:8082
Connection: Keep-alive
```

Recommendation

It's recommended to implement Content Security Policy (CSP) into your web application. Configuring Content Security Policy involves adding the **Content-Security-Policy** HTTP header to a web page and giving it values to control resources the user agent is allowed to load for that page.

References

Content Security Policy (CSP)
https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

Implementing Content Security Policy
https://hacks.mozilla.org/2016/02/implementing-content-security-policy/

Coverage

- https://localhost:8082
  - fa
    - v1
      - helloworld
        - restful