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

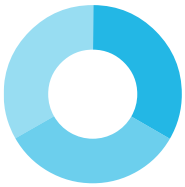
Target	https://localhost:7056
Scan Type	Full Scan
Start Time	Jun 1, 2022, 10:46:27 PM GMT-7
Scan Duration	15 minutes
Requests	37955
Average Response Time	1ms
Maximum Response Time	8374ms

Severity	Vulnerabilities	Instances
High	1	1
Medium	3	3
Low	3	3
Informational	1	1
Total	8	8

Informational

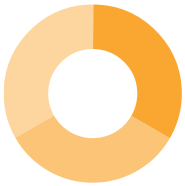
	Instances
Content Security Policy (CSP) not implemented	1

Low Severity



	Instances
Clickjacking: X-Frame-Options header	1
HTTP Strict Transport Security (HSTS) not imp…	1
Sensitive pages could be cached	1

Medium Severity



	Instances
TLS 1.1 enabled	1
TLS/SSL Sweet32 attack	1
TLS/SSL Weak Cipher Suites	1

High Severity

	Instances
TLS 1.0 enabled	1

Impacts

SEVERITY	IMPACT	
High	1	TLS 1.0 enabled
Medium	1	TLS 1.1 enabled
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
Low	1	Sensitive pages could be cached
Informational	1	Content Security Policy (CSP) not implemented

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:7056/ Confidence: 100%

The SSL server (port: 7056) 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:7056/ Confidence: 100%

The SSL server (port: 7056) 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 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:7056/

Cipher suites susceptible to Sweet32 attack (TLS1.0 on port 7056):

- TLS_RSA_WITH_3DES_EDE_CBC_SHA

Cipher suites susceptible to Sweet32 attack (TLS1.1 on port 7056):

- TLS_RSA_WITH_3DES_EDE_CBC_SHA

Cipher suites susceptible to Sweet32 attack (TLS1.2 on port 7056):

- 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:7056/

Weak TLS/SSL Cipher Suites: (offered via TLS1.0 on port 7056):

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

Weak TLS/SSL Cipher Suites: (offered via TLS1.1 on port 7056):

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

Weak TLS/SSL Cipher Suites: (offered via TLS1.2 on port 7056):

- 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

RFC 9155: Deprecating MD5 and SHA-1 Signature Hashes in TLS 1.2 and DTLS 1.2
https://datatracker.ietf.org/doc/html/rfc9155

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:7056/

Paths without secure XFO header:

- https://localhost:7056/swagger/index.html

Request

```
GET /swagger/index.html HTTP/1.1
Referer: https://localhost:7056/swagger/
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:7056
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:7056/

URLs where HSTS is not enabled:

- https://localhost:7056/swagger/index.html

Request

```
GET /swagger/index.html HTTP/1.1
Referer: https://localhost:7056/swagger/
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:7056
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

Sensitive pages could be cached

One or more pages contain possible sensitive information (e.g. a password parameter) and could be potentially cached. Even in secure SSL channels sensitive data could be stored by intermediary proxies and SSL terminators. To prevent this, a Cache-Control header should be specified.

Impact

Possible sensitive information disclosure.

---

https://localhost:7056/

List of pages that could be cached:

- https://localhost:7056/swagger/index.html?password=g00dPa$$w0rD&username=KfnqDuxw
- https://localhost:7056/swagger/v1/swagger.json?password=g00dPa$$w0rD&username=KfnqDuxw

Request

```
GET /swagger/index.html?password=g00dPa%24%24w0rD&username=KfnqDuxw HTTP/1.1
Referer: https://localhost:7056/swagger/index.html
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:7056
Connection: Keep-alive
```

Recommendation

Prevent caching by adding "Cache Control: No-store" and "Pragma: no-cache" to the HTTP response header.

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:7056/

Paths without CSP header:

- https://localhost:7056/swagger/index.html

Request

```
GET /swagger/index.html HTTP/1.1
Referer: https://localhost:7056/swagger/
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:7056
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:7056
  - _framework
    - aspnetcore-browser-refresh.js
    - blazor-hotreload
    - clear-browser-cache
  - _vs
    - browserLink
  - swagger
    - v1
      - Currencies
        - fiat
      - Notification
        - bitpay
          - Inputs
            - POST url, status, currency, invoiceTime, expirationTime, currentTime, id, transactionCurrency, exceptionStatus, paymentCodes.bch.biP72b, paymentCodes.bch.biP73, paymentCodes.btc.biP72b, paymentCodes.btc.biP73, paymentCodes.busd.eiP681b, paymentCodes.dai.eiP681b, paymentCodes.doge.biP72b, paymentCodes.doge.biP73, paymentCodes.eth.eiP681, paymentCodes.gusd.eiP681b, paymentCodes.pax.eiP681b, paymentCodes.usdc.eiP681b, paymentCodes.wbtc.eiP681b, paymentCodes.xrp.biP72b, paymentCodes.xrp.biP73, paymentCodes.xrp.riP681, paymentDisplayTotals.btc, paymentDisplayTotals.bch, paymentDisplayTotals.eth, paymentDisplayTotals.gusd, paymentDisplayTotals.pax, paymentDisplayTotals.busd, paymentDisplayTotals.usdc, paymentDisplayTotals.xrp, paymentDisplayTotals.doge, paymentDisplayTotals.dai, paymentDisplayTotals.wbtc
        - coinbase
          - Inputs
            - POST data.id, data.created_at, data.expires_at, data.addresses.bitcoin, data.addresses.ethereum, data.pricing.bitcoin.amount, data.pricing.bitcoin.currency, data.pricing.ethereum.amount, data.pricing.ethereum.currency, data.pricing.local.amount, data.pricing.local.currency, data.timeline.1.time, data.timeline.1.status, data.timeline.1.context
        - coinpayments
          - Inputs
            - POST txn_id, amount1, amount2, currency1, currency2, status, status_text
        - coinqvest
          - Inputs

POST eventType, data.checkout.id, data.checkout.state

Payment

Payment

Inputs

POST amount, fiatCurrency, cryptoCurrency, transactionReference

swagger.json

Inputs

GET password, username

swagger.yaml

index.html

#fragments

javascript:domxssExecutionSink(1,javascript:domxssExecutionSink(1,"'/"%3E%3Cxsstag%3E()hashxss")

Inputs

GET password, username

swagger-ui-bundle.js

swagger-ui-standalone-preset.js

swagger-ui.css