Probely

SECURITY REPORT

admin - https://project.wewebcloud.com/dev22-git/weadmin

Report generated on Nov. 22, 2022 at 16:11 UTC

Summary

This section contains the scan summary

TARGET https://project.wewebcloud.com/dev22-git/weadmin

Report generated on Nov. 22, 2022 at 16:11 UTC

STARTED	ENDED	DURATION	SCAN PROFILE
Nov. 22, 2022, 10:47 UTC	Nov. 22, 2022, 10:49 UTC	2 minutes	Lightning

NUMBER OF FINDINGS

	CURRENT SCAN	FROM LAST SCAN	PENDING FIX
HIGH	**0**	**= 0**	**0**
MEDIUM	**0**	**= 0**	**0**
LOW	**5**	**= 0**	**5**

TOP 5

HSTS header not enforced	**1**
Missing Content Security Policy header	**1**
Missing clickjacking protection	**1**
Referrer policy not defined	**1**
Browser content sniffing allowed	**1**

Settings

This section contains the summary of settings that were used during this scan

✓ SCAN PROFILE

Lightning

Security posture scan. Fast scan that checks vulnerabilities related to SSL/TLS, HTTP security headers, and cookies attributes.

Technical Summary

The following table summarizes the findings, ordered by their severity

#	SEVERITY	VULNERABILITY	STATE
3	LOW	**Referrer policy not defined** https://project.wewebcloud.com/dev22-git/weadmin/	NOT FIXED
2	LOW	**Missing Content Security Policy header** https://project.wewebcloud.com/dev22-git/weadmin/	NOT FIXED
4	LOW	**Missing clickjacking protection** https://project.wewebcloud.com/dev22-git/weadmin/	NOT FIXED
1	LOW	**HSTS header not enforced** https://project.wewebcloud.com/dev22-git/weadmin/	NOT FIXED
5	LOW	**Browser content sniffing allowed** https://project.wewebcloud.com/dev22-git/weadmin/	NOT FIXED

Exhaustive Test List

The following pages contain the list of vulnerabilities we tested in this scan, taking into consideration the chosen profile

- Cookie without HttpOnly flag
- Missing clickjacking protection
- SSL cookie without Secure flag
- Unencrypted communications
- HSTS header not enforced
- Certificate with insufficient key size or usage, or insecure signature algorithm
- Expired TLS certificate
- Insecure SSL protocol version 3 supported
- Deprecated TLS protocol version 1.0 supported
- Deprecated TLS protocol version 1.1 supported
- Secure TLS protocol version 1.2 not supported
- Weak cipher suites enabled
- Server Cipher Order not configured
- Untrusted TLS certificate
- Heartbleed
- Secure Renegotiation is not supported
- TLS Downgrade attack prevention not supported
- Certificate without revocation information
- HSTS header set in HTTP
- HSTS header with low duration and no subdomain protection
- HSTS header with low duration
- HSTS header does not protect subdomains
- Insecure SSL protocol version 2 supported
- Browser XSS protection disabled
- Browser content sniffing allowed
- Referrer policy not defined
- Insecure referrer policy
- Potential DoS on TLS Client Renegotiation
- TLS certificate about to expire
- Invalid referrer policy
- Missing Content Security Policy header
- Insecure Content Security Policy

Detailed Finding Descriptions

This section contains the findings in more detail, ordered by severity

# 3 Referrer policy not defined

LOW

CVSS SCORE
3.1 CVSS:3.0/AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:N/A:N

METHOD	PATH
GET	https://project.wewebcloud.com/dev22-git/weadmin/

DESCRIPTION

The application does not prevent browsers from sending sensitive information to third party sites in the **referer** header.

Without a referrer policy, every time a user clicks a link that takes him to another origin (domain), the browser will add a **referer** header with the URL from which he is coming from. That URL may contain sensitive information, such as password recovery tokens or personal information, and it will be visible that other origin. For instance, if the user is at `example.com/password_recovery?unique_token=14f748d89d` and clicks a link to `example-analytics.com`, that origin will receive the complete password recovery URL in the headers and might be able to set the users password. The same happens for requests made automatically by the application, such as XHR ones.

Applications should set a secure referrer policy that prevents sensitive data from being sent to third party sites.

EVIDENCE

Response headers, missing the `Referrer-Policy` header:

```
Server: nginx
Date: Tue, 22 Nov 2022 10:47:08 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Set-Cookie: PHPSESSID=645eac7db61c3070be4e525118e88218; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
```

REQUEST

```
GET /dev22-git/weadmin/ HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; +https://probely.com/sos) ProbelyMRKT/0.1.0
Accept: */*
Accept-Encoding: gzip, deflate
Host: project.wewebcloud.com
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx
Date: Tue, 22 Nov 2022 10:47:08 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
```

```
Connection: keep-alive
Keep-Alive: timeout=20
Set-Cookie: PHPSESSID=645eac7db61c3070be4e525118e88218; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="th">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0,
maximum-scale=1">
        <meta name="robots" content="noindex, nofollow">
            <meta name="googlebot" content="noindex, nofollow">
                <link href="css/theme.css" rel="stylesheet"/>
                <link href="css/bootstrap.min.css" rel="stylesheet"/>
                <link href="css/bootstrap-theme.min.css" rel="stylesheet"/>
                <!-- <link href="css/font-awesome.min.css" rel="stylesheet"/>
-->
                <script
src="https://use.fontawesome.com/6f8306cef9.js"></script>
                <title>                         </title>
```

# 2 Missing Content Security Policy header

LOW

CVSS SCORE

3.7 CVSS:3.0/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N

METHOD	PATH
GET	https://project.wewebcloud.com/dev22-git/weadmin/

DESCRIPTION

The Content Security Policy (CSP) is an HTTP header through which site owners define a set of security rules that the browser must follow when rendering their site. The most common usage is to define a list of approved sources of content that the browser can lo ad. This can be used to effectively mitigate Cross-Site Scripting (XSS) and Clickjacking attacks.

CSP is flexible enough for you to define from where the browser can load JavaScript, Stylesheets, images, or fonts, among other op tions. It can also be used in report mode only, a recommended approach before deploying strict rules in a live environment. Howev er, please note that report mode does not protect you, it just logs policy violations.

EVIDENCE

Response headers, missing the `Content-Security-Policy` header:

```
Server: nginx
Date: Tue, 22 Nov 2022 10:47:13 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
```

REQUEST

```
GET /dev22-git/weadmin/ HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; +https://probely.com/sos) ProbelyMRKT/0.1.0
Accept: */*
Accept-Encoding: gzip, deflate
Host: project.wewebcloud.com
Cookie: PHPSESSID=645eac7db61c3070be4e525118e88218
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx
Date: Tue, 22 Nov 2022 10:47:13 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="th">
```

```
     <head>
          <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
          <meta name="viewport" content="width=device-width, initial-scale=1.0,
maximum-scale=1">
          <meta name="robots" content="noindex, nofollow">
               <meta name="googlebot" content="noindex, nofollow">
                    <link href="css/theme.css" rel="stylesheet"/>
                    <link href="css/bootstrap.min.css" rel="stylesheet"/>
                    <link href="css/bootstrap-theme.min.css" rel="stylesheet"/>
                    <!-- <link href="css/font-awesome.min.css" rel="stylesheet"/>
-->
                    <script
src="https://use.fontawesome.com/6f8306cef9.js"></script>
                    <title>                        </title>
                    <script language="JavaScript"  type="text/javascript"
src="js/jquery-1.9.0.js"></script>
```

# 4

Missing clickjacking protection

LOW

CVSS SCORE
6.5
CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:H/A:N

METHOD	PATH
GET	https://project.wewebcloud.com/dev22-git/weadmin/

DESCRIPTION

A **frameable response** occurs when one or multiple pages can be used on an iframe on any website. This allows the **clickjackin g** attack to be used.

Clickjacking is when an attacker a hidden iframe with multiple transparent or opaque layers above it, to trick a user into clicking on a button or link on the iframe when they were intending to click on the the top level page. Thus, the attacker is "hijacking" clic ks meant for the top level page and routing them to the iframe.

Using a similar technique, keystrokes can also be hijacked. With a carefully crafted combination of stylesheets, iframes, and text bo xes, a user can be led to believe they are typing in the password to their email or bank account, but are instead typing into an inv isible frame controlled by the attacker.

EVIDENCE

Response headers, missing the `X-Frame-Options` header:

```
Server: nginx
Date: Tue, 22 Nov 2022 10:47:13 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
```

REQUEST

```
GET /dev22-git/weadmin/ HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; +https://probely.com/sos) ProbelyMRKT/0.1.0
Accept: */*
Accept-Encoding: gzip, deflate
Host: project.wewebcloud.com
Cookie: PHPSESSID=645eac7db61c3070be4e525118e88218
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx
Date: Tue, 22 Nov 2022 10:47:13 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
```

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="th">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0,
maximum-scale=1">
        <meta name="robots" content="noindex, nofollow">
            <meta name="googlebot" content="noindex, nofollow">
                <link href="css/theme.css" rel="stylesheet"/>
                <link href="css/bootstrap.min.css" rel="stylesheet"/>
                <link href="css/bootstrap-theme.min.css" rel="stylesheet"/>
                <!-- <link href="css/font-awesome.min.css" rel="stylesheet"/>
-->
                <script
src="https://use.fontawesome.com/6f8306cef9.js"></script>
                <title>                    </title>
                <script language="JavaScript"  type="text/javascript"
src="js/jquery-1.9.0.js"></script>
```

1

HSTS header not enforced

LOW

CVSS SCORE
7.4
CVSS:3.0/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N

METHOD	PATH
GET	https://project.wewebcloud.com/dev22-git/weadmin/

DESCRIPTION

The application does not force users to connect over an encrypted channel, i.e. over HTTPS. If the user types the site address in th e browser without starting with *https*, it will connect to it over an insecure channel, even if there is a redirect to HTTPS later. Even if the user types *https*, there may be links to the site in HTTP, forcing the user to navigate insecurely. An attacker that is able to in tercept traffic between the victim and the site or spoof the site's address can prevent the user from ever connecting to it over an encrypted channel. This way, the attacker is able to eavesdrop all communications between the victim and the server, including th e victim's credentials, session cookie and other sensitive information.

EVIDENCE

Response headers, missing the `Strict-Transport-Security` header:

```
HTTP/1.1 200 OK
Server: nginx
Date: Tue, 22 Nov 2022 10:47:13 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
```

REQUEST

```
GET /dev22-git/weadmin/ HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; +https://probely.com/sos) ProbelyMRKT/0.1.0
Accept: */*
Accept-Encoding: gzip, deflate
Host: project.wewebcloud.com
Cookie: PHPSESSID=645eac7db61c3070be4e525118e88218
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx
Date: Tue, 22 Nov 2022 10:47:13 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="th">
```

```
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0,
maximum-scale=1">
        <meta name="robots" content="noindex, nofollow">
            <meta name="googlebot" content="noindex, nofollow">
                <link href="css/theme.css" rel="stylesheet"/>
                <link href="css/bootstrap.min.css" rel="stylesheet"/>
                <link href="css/bootstrap-theme.min.css" rel="stylesheet"/>
                <!-- <link href="css/font-awesome.min.css" rel="stylesheet"/>
-->
                <script
src="https://use.fontawesome.com/6f8306cef9.js"></script>
                <title>                    </title>
                <script language="JavaScript"  type="text/javascript"
src="js/jquery-1.9.0.js"></script>
```

# 5 Browser content sniffing allowed

LOW

CVSS SCORE
4.7 CVSS:3.0/AV:N/AC:H/PR:N/UI:R/S:C/C:L/I:L/A:N

METHOD	PATH
GET	https://project.wewebcloud.com/dev22-git/weadmin/

DESCRIPTION

The application allows browsers to try to mime-sniff the content-type of the responses. This means the browser may try to guess t he content-type by looking at the response content, and render it in way it was not intended to. This behavior may lead to the exe cution of malicious code, for instance, to explore an XSS vulnerability.

Applications should disable this behavior, forcing browsers to honor the content-type specified in the response. Without a specific c ontent-type set browsers will default to render the content as text, turning XSS payloads innocuous.

Disabling mime-sniffing should be seen as an extra layer of defense against XSS, and not as replacement of the recommended XSS prevention techniques.

EVIDENCE

Response headers, missing the `X-Content-Type-Options` header:

```
Server: nginx
Date: Tue, 22 Nov 2022 10:47:08 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Set-Cookie: PHPSESSID=645eac7db61c3070be4e525118e88218; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
```

REQUEST

```
GET /dev22-git/weadmin/ HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; +https://probely.com/sos) ProbelyMRKT/0.1.0
Accept: */*
Accept-Encoding: gzip, deflate
Host: project.wewebcloud.com
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx
Date: Tue, 22 Nov 2022 10:47:08 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Set-Cookie: PHPSESSID=645eac7db61c3070be4e525118e88218; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="th">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0,
maximum-scale=1">
        <meta name="robots" content="noindex, nofollow">
            <meta name="googlebot" content="noindex, nofollow">
                <link href="css/theme.css" rel="stylesheet"/>
                <link href="css/bootstrap.min.css" rel="stylesheet"/>
                <link href="css/bootstrap-theme.min.css" rel="stylesheet"/>
                <!-- <link href="css/font-awesome.min.css" rel="stylesheet"/>
-->
                <script
src="https://use.fontawesome.com/6f8306cef9.js"></script>
                <title>                    </title>
```

Glossary

Term	Definition
Vulnerability	A type of security weakness that might occur in applications (e.g. Broken Authentication, Information Disclosure). Some vulnerabilities take their name not from the weakness itself, but from the attack that exploits it (e.g. SQL Injection, XSS, etc.).
Findings	An instance of a Vulnerability that was found in an application.

Severity Legend

To each finding is attributed a severity which sums up its overall risk

The severity is a compound metric that encompasses the likelihood of the finding being found and exploited by an attacker, the skill required to exploit it, and the impact of such exploitation. A finding that is easy to find, easy to exploit and the exploitation has high impact, will have a greater severity.

Different findings of the same type could have a different severity: we consider multiple factors to increase or decrease it, such as if the application has an authenticated area or not.

The following table describes the different severities:

Severity	Description	Examples
HIGH	These findings may have a direct impact in the application security, either clients or service owners, for instance by granting the attacker access to sensitive information.	SQL Injection OS Command Injection
MEDIUM	Medium findings usually don't have immediate impact alone, but combined with other findings may lead to a successful compromise of the application.	Cross-site Request Forgery Unencrypted Communications
LOW	Findings where either the exploit is not trivial, the impact is low, or the finding cannot be exploited by itself.	Directory Listing Clickjacking

Category Descriptions

The following pages contain descriptions of each vulnerability. For each vulnerability you will find a section explaining its impact, causes and prevention methods.
These descriptions are very generic, and whenever they are not enough to understand or fix a given finding, more information is provided for that finding in the Detailed Finding Descriptions section.

Referrer policy not defined

Description

The application does not prevent browsers from sending sensitive information to third party sites in the **referer** header.

Without a referrer policy, every time a user clicks a link that takes him to another origin (domain), the browser will add a **referer** header with the URL from which he is coming from. That URL may contain sensitive information, such as password recovery tokens or personal information, and it will be visible that other origin. For instance, if the user is at `example.com/password_recovery?unique_token=14f748d89d` and clicks a link to `example-analytics.com`, that origin will receive the complete password recovery URL in the headers and might be able to set the users password. The same happens for requests made automatically by the application, such as XHR ones.

Applications should set a secure referrer policy that prevents sensitive data from being sent to third party sites.

Fix

This problem can be fixed by sending the header **Referrer-Policy** with a secure value. There are different values available, but not all are considered secure. The following list explains each one, and it is ordered from the safest to the least safe:

- `no-referrer`: never send the header.
- `same-origin`: send the full URL to requests to the same origin (exact scheme + domain)
- `strict-origin`: send only the domain part of the URL, but sends nothing when downgrading to HTTP.
- `origin`: similar to **strict-origin** without downgrade restriction.
- `strict-origin-when-cross-origin`: send full URL within the same origin, but only the domain part when sending to another origin. It sends nothing when downgrading to HTTP.
- `origin-when-cross-origin`: similar to **strict-origin-when-cross-origin** without the downgrade restriction.

Insecure options: * `no-referrer-when-downgrade`: sends the full URL when the scheme does not change. It will send if both origins are, for instance, HTTP. * `unsafe-url`: always sent the full URL

A possible, safe option is `strict-origin`, so for nginx add the following line to your virtual host configuration file:

```
add_header Referrer-Policy "strict-origin" always;
```

It is normally easy to enable the header in the web server configuration file, but it can also be done at the application level.

Please note that the referrer header is written `referer`, with a single `r` but the referrer policy header is properly written, with `rr`: `Referrer-Policy`.

Missing Content Security Policy header

Description

The Content Security Policy (CSP) is an HTTP header through which site owners define a set of security rules that the browser must follow when rendering their site. The most common usage is to define a list of approved sources of content that the browser can load. This can be used to effectively mitigate Cross-Site Scripting (XSS) and Clickjacking attacks.

CSP is flexible enough for you to define from where the browser can load JavaScript, Stylesheets, images, or fonts, among other options. It can also be used in report mode only, a recommended approach before deploying strict rules in a live environment. However, please note that report mode does not protect you, it just logs policy violations.

Fix

You can define a Content Security Policy by setting a header in your application. The header can look like this:

```
Content-Security-Policy: frame-ancestors 'none'; default-src 'self', script-src '*://*.example.com:*'
```

In this example, the **frame-ancestors** directive set to **'none'** indicates that the page cannot be placed inside a frame, not even by itself. The **default-src** defines the loading policy for all resources, in this case, they can be loaded from the current origin (protocol + domain + port). The example sets a more specific policy for scripts, through the **script-src**, restricting script loading to any subdomain of example.com.

The policy can be with different directives, and there are other less strict options for the directives above.

Missing clickjacking protection

Description

A **frameable response** occurs when one or multiple pages can be used on an iframe on any website. This allows the **clickjacking** attack to be used.

Clickjacking is when an attacker a hidden iframe with multiple transparent or opaque layers above it, to trick a user into clicking on a button or link on the iframe when they were intending to click on the the top level page. Thus, the attacker is "hijacking" clicks meant for the top level page and routing them to the iframe.

Using a similar technique, keystrokes can also be hijacked. With a carefully crafted combination of stylesheets, iframes, and text boxes, a user can be led to believe they are typing in the password to their email or bank account, but are instead typing into an invisible frame controlled by the attacker.

Fix

The recommended way to prevent clickjacking is to send a header that instructs the browser to not allow arbitrary framing, typically from other domains.

The current recommendation is to use the Content-Security-Policy HTTP header (CSP) with a **frame-ancestors** directive. This header obsoletes the X-Frame-Options HTTP header.

To use CSP you need the following header:

```
Content-Security-Policy: frame-ancestors 'none'
```

The header might contain more directives, and there are other less strict options for the **frame-ancestors** directive.

If you want to use X-Frame-Options, send the proper HTTP header, with one of the following directives:

```
X-Frame-Options: DENY
X-Frame-Options: SAMEORIGIN
```

A third directive, **ALLOW-FROM** is no longer supported by modern browsers.

If you specify **DENY**, all attempts to load the page in a frame will fail. **SAMEORIGIN** will allow the page to be loaded in the site including it in a frame is the same as the one serving the page.

The most common option is **DENY** when there is no need to load your pages on some other site.

To configure nginx to send the Content-Security-Policy header, add this either to your http server, or location configuration:

```
add_header Content-Security-Policy "frame-ancestors 'none'";
```

To configure nginx to send the X-Frame-Options header, add this either to your http server, or location configuration:

```
add_header X-Frame-Options DENY;
```

HSTS header not enforced

Description

The application does not force users to connect over an encrypted channel, i.e. over HTTPS. If the user types the site address in the browser without starting with *https*, it will connect to it over an insecure channel, even if there is a redirect to HTTPS later. Even if the user types *https*, there may be links to the site in HTTP, forcing the user to navigate insecurely. An attacker that is able to intercept traffic between the victim and the site or spoof the site's address can prevent the user from ever connecting to it over an encrypted channel. This way, the attacker is able to eavesdrop all communications between the victim and the server, including the victim's credentials, session cookie and other sensitive information.

Fix

The application should instruct web browsers to only access the application using HTTPS. To do this, enable HTTP Strict Transport Security (HSTS).

You can do so by sending the `Strict-Transport-Security` header so that a browser that supports HSTS will always enforce a secure connection to your site on all subsequent requests. This will prevent some passive and active attacks such as *sslstrip*.

An HSTS enabled server includes the following header in an HTTPS reply: `Strict-Transport-Security: max-age=15768000;includeSubdomains`

When the browser sees this, it will remember, for the given number of seconds, that the current domain should only be contacted over HTTPS. In the future, if the user types *http://* or omits the scheme, HTTPS is the default. In this example, which includes the option includeSubdomains, all requests to URLs in the current domain and subdomains will be redirected to HTTPS. When you set `includeSubdomains` make sure you can serve all requests over HTTPS! It is, however, important that you add the option `includeSubdomains` whenever is possible.

Instead of changing your application, you can have the web server doing this for you.

You should add the following line to your NGINX host configuration: `add_header Strict-Transport-Security max-age=15768000;includeSubdomains` Note that because HSTS is a "trust on first use" (TOFU) protocol, a user who has never accessed the application will never have seen the HSTS header, and will therefore still be vulnerable to SSL stripping attacks. To mitigate this risk, you can optionally add the 'preload' flag to the HSTS header, and submit the domain for review by browser vendors.

Browser content sniffing allowed

Description

The application allows browsers to try to mime-sniff the content-type of the responses. This means the browser may try to guess the content-type by looking at the response content, and render it in way it was not intended to. This behavior may lead to the execution of malicious code, for instance, to explore an XSS vulnerability.

Applications should disable this behavior, forcing browsers to honor the content-type specified in the response. Without a specific content-type set browsers will default to render the content as text, turning XSS payloads innocuous.

Disabling mime-sniffing should be seen as an extra layer of defense against XSS, and not as replacement of the recommended XSS prevention techniques.

Fix

This problem can be fixed by sending the header **X-Content-Type-Options** with value **nosniff**, to force browsers to disable the content-type guessing (the sniffing).

The header should look this:

```
X-Content-Type-Options: nosniff
```

For nginx add the following line to your virtual host configuration file:

```
add_header X-Content-Type-Options "nosniff" always;
```

It is normally easy to enable the header in the web server configuration file, but it can also be done at application level.