Отчет по результатам сканирования безопасности

Проект: Тестовый проект 2

URL: http://testphp.vulnweb.com

Дата сканирования: 26.04.2025 20:37

Сводка результатов

Всего выявлено уязвимостей: 16

Критические	Высокие	Средние	Низкие	Информационные
2	2	11	1	0

Подробные результаты

1. SQL Injection in searchFor

Уровень серьезности: Критический

Статус: Обнаружена

Описание

SQL injection vulnerability found using BET technique at URL http://testphp.vulnweb.com/search.php?test=query

Решение

Implement parameterized queries, input validation and apply the principle of least privilege

Рекомендации

Verify the vulnerability manually and implement proper input validation

2. Content Security Policy (CSP) Header Not Set

Уровень серьезности: Средний

Статус: Обнаружена

Описание

Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.

Решение

Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header.

Рекомендации

См. решение и ссылки:

https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing_Content_Security_Policy

https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html

https://www.w3.org/TR/CSP/

https://w3c.github.io/webappsec-csp/

https://web.dev/articles/csp

https://caniuse.com/#feat=contentsecuritypolicy

https://content-security-policy.com/

3. Missing Anti-clickjacking Header

Уровень серьезности: Высокий

Статус: Обнаружена

Описание

The response does not protect against 'ClickJacking' attacks. It should include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options.

Решение

Modern Web browsers support the Content-Security-Policy and X-Frame-Options HTTP headers. Ensure one of them is set on all web pages returned by your site/app.

If you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. Alternatively consider implementing Content Security Policy's "frame-ancestors" directive.

Рекомендации

См. решение и ссылки:

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

4. Absence of Anti-CSRF Tokens

Уровень серьезности: Высокий

Статус: Обнаружена

Описание

No Anti-CSRF tokens were found in a HTML submission form.

A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf.

CSRF attacks are effective in a number of situations, including:

* The victim has an active session on the target site.

* The victim is authenticated via HTTP auth on the target site.

* The victim is on the same local network as the target site.

CSRF has primarily been used to perform an action against a target site using the victim's privileges, but recent techniques have been discovered to disclose information by gaining access to the response. The risk of information disclosure is dramatically increased when the target site is vulnerable to XSS, because XSS can be used as a platform for CSRF, allowing the attack to operate within the bounds of the same-origin policy.

Решение

Phase: Architecture and Design

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, use anti-CSRF packages such as the OWASP CSRFGuard.

Phase: Implementation

Ensure that your application is free of cross-site scripting issues, because most CSRF defenses can be bypassed using attacker-controlled script.

Phase: Architecture and Design

Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330).

Note that this can be bypassed using XSS.

Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation.

Note that this can be bypassed using XSS.

Use the ESAPI Session Management control.

This control includes a component for CSRF.

Do not use the GET method for any request that triggers a state change.

Phase: Implementation

Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons.

Рекомендации

См. решение и ссылки:

https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

https://cwe.mitre.org/data/definitions/352.html

5. Insufficient Site Isolation Against Spectre Vulnerability

Уровень серьезности: Средний

Статус: Обнаружена

Описание

Cross-Origin-Resource-Policy header is an opt-in header designed to counter side-channels attacks like Spectre. Resource should be specifically set as shareable amongst different origins.

Решение

Ensure that the application/web server sets the Cross-Origin-Resource-Policy header appropriately, and that it sets the Cross-Origin-Resource-Policy header to 'same-origin' for all web pages.

'same-site' is considered as less secured and should be avoided.

If resources must be shared, set the header to 'cross-origin'.

If possible, ensure that the end user uses a standards-compliant and modern web browser that supports the Cross-Origin-Resource-Policy header (https://caniuse.com/mdn-http_headers_cross-origin-resource-policy).

Рекомендации

См. решение и ссылки:

https://developer.mozilla.org/en-US/docs/Web/HTTP/Cross-Origin_Resource_Policy

6. Permissions Policy Header Not Set

Уровень серьезности: Средний

Статус: Обнаружена

Описание

Permissions Policy Header is an added layer of security that helps to restrict from unauthorized access or usage of browser/client features by web resources. This policy ensures the user privacy by limiting or specifying the features of the browsers can be used by the web resources. Permissions Policy provides a set of standard HTTP headers that allow website owners to limit which features of browsers can be used by the page such as camera, microphone, location, full screen etc.

Решение

Ensure that your web server, application server, load balancer, etc. is configured to set the Permissions-Policy header.

Рекомендации

См. решение и ссылки:

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Permissions-Policy

https://developer.chrome.com/blog/feature-policy/

https://scotthelme.co.uk/a-new-security-header-feature-policy/

https://w3c.github.io/webappsec-feature-policy/

https://www.smashingmagazine.com/2018/12/feature-policy/

7. Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)

Уровень серьезности: Средний

Статус: Обнаружена

Описание

The web/application server is leaking information via one or more "X-Powered-By" HTTP response headers. Access to such information may facilitate attackers identifying other frameworks/components your web application is reliant upon and the vulnerabilities such components may be subject to.

Решение

Ensure that your web server, application server, load balancer, etc. is configured to suppress "X-Powered-By" headers.

Рекомендации

См. решение и ссылки:

https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/01-Information_Gathering/08-Fingerprint_Web_Application_Framework

https://www.troyhunt.com/2012/02/shhh-dont-let-your-response-headers.html

8. Server Leaks Version Information via "Server" HTTP Response Header Field

Уровень серьезности: Средний

Статус: Обнаружена

Описание

The web/application server is leaking version information via the "Server" HTTP response header. Access to such information may facilitate attackers identifying other vulnerabilities your web/application server is subject to.

Решение

Ensure that your web server, application server, load balancer, etc. is configured to suppress the "Server" header or provide generic details.

Рекомендации

См. решение и ссылки:

https://httpd.apache.org/docs/current/mod/core.html#servertokens

https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff648552(v=pandp.10)

https://www.troyhunt.com/shhh-dont-let-your-response-headers/

9. X-Content-Type-Options Header Missing

Уровень серьезности: Средний

Статус: Обнаружена

Описание

The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.

Решение

Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages.

If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.

Рекомендации

См. решение и ссылки:

https://learn.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/compatibility/gg622941(v=vs.85)

https://owasp.org/www-community/Security_Headers

10. In Page Banner Information Leak

Уровень серьезности: Средний

Статус: Обнаружена

Описание

The server returned a version banner string in the response content. Such information leaks may allow attackers to further target specific issues impacting the product and version in use.

Решение

Configure the server to prevent such information leaks. For example:

Under Tomcat this is done via the "server" directive and implementation of custom error pages.

Under Apache this is done via the "ServerSignature" and "ServerTokens" directives.

Рекомендации

См. решение и ссылки:

https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/08-Testing_for_Error_Handling/

11. Charset Mismatch (Header Versus Meta Content-Type Charset)

Уровень серьезности: Средний

Статус: Обнаружена

Описание

This check identifies responses where the HTTP Content-Type header declares a charset different from the charset defined by the body of the HTML or XML. When there's a charset mismatch between the HTTP header and content body Web browsers can be forced into an undesirable content-sniffing mode to determine the content's correct character set.

An attacker could manipulate content on the page to be interpreted in an encoding of their choice. For example, if an attacker can control content at the beginning of the page, they could inject script using UTF-7 encoded text and manipulate some browsers into interpreting that text.

Решение

Force UTF-8 for all text content in both the HTTP header and meta tags in HTML or encoding declarations in XML.

Рекомендации

См. решение и ссылки:

https://code.google.com/p/browsersec/wiki/Part2#Character_set_handling_and_detection

12. Modern Web Application

Уровень серьезности: Средний

Статус: Обнаружена

Описание

The application appears to be a modern web application. If you need to explore it automatically then the Ajax Spider may well be more effective than the standard one.

Решение

This is an informational alert and so no changes are required.

Рекомендации

См. решение и ссылки:

13. Non-Storable Content

Уровень серьезности: Средний

Статус: Обнаружена

Описание

The response contents are not storable by caching components such as proxy servers. If the response does not contain sensitive, personal or user-specific information, it may benefit from being stored and cached, to improve performance.

Решение

The content may be marked as storable by ensuring that the following conditions are satisfied:

The request method must be understood by the cache and defined as being cacheable ("GET", "HEAD", and "POST" are currently defined as cacheable)

The response status code must be understood by the cache (one of the 1XX, 2XX, 3XX, 4XX, or 5XX response classes are generally understood)

The "no-store" cache directive must not appear in the request or response header fields

For caching by "shared" caches such as "proxy" caches, the "private" response directive must not appear in the response

For caching by "shared" caches such as "proxy" caches, the "Authorization" header field must not appear in the request, unless the response explicitly allows it (using one of the "must-revalidate", "public", or "s-maxage" Cache-Control response directives)

In addition to the conditions above, at least one of the following conditions must also be satisfied by the response:

It must contain an "Expires" header field

It must contain a "max-age" response directive

For "shared" caches such as "proxy" caches, it must contain a "s-maxage" response directive

It must contain a "Cache Control Extension" that allows it to be cached

It must have a status code that is defined as cacheable by default (200, 203, 204, 206, 300, 301, 404, 405, 410, 414, 501).

Рекомендации

См. решение и ссылки:

https://datatracker.ietf.org/doc/html/rfc7234

https://datatracker.ietf.org/doc/html/rfc7231

https://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html

14. Storable and Cacheable Content

Уровень серьезности: Низкий

Статус: Обнаружена

Описание

The response contents are storable by caching components such as proxy servers, and may be retrieved directly from the cache, rather than from the origin server by the caching servers, in response to similar requests from other users. If the response data is sensitive, personal or user-specific, this may result in sensitive information being leaked. In some cases, this may even result in a user gaining complete control of the session of another user, depending on the configuration of the caching components in use in their environment. This is primarily an issue where "shared" caching servers such as "proxy" caches are configured on the local network. This configuration is typically found in corporate or educational environments, for instance.

Решение

Validate that the response does not contain sensitive, personal or user-specific information. If it does, consider the use of the following HTTP response headers, to limit, or prevent the content being stored and retrieved from the cache by another user:

Cache-Control: no-cache, no-store, must-revalidate, private

Pragma: no-cache

Expires: 0

This configuration directs both HTTP 1.0 and HTTP 1.1 compliant caching servers to not store the response, and to not retrieve the response (without validation) from the cache, in response to a similar request.

Рекомендации

См. решение и ссылки:

https://datatracker.ietf.org/doc/html/rfc7234

https://datatracker.ietf.org/doc/html/rfc7231

https://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html

15. User Controllable HTML Element Attribute (Potential XSS)

Уровень серьезности: Критический

Статус: Обнаружена

Описание

This check looks at user-supplied input in query string parameters and POST data to identify where certain HTML attribute values might be controlled. This provides hot-spot detection for XSS (cross-site scripting) that will require further review by a security analyst to determine exploitability.

Решение

Validate all input and sanitize output it before writing to any HTML attributes.

Рекомендации

См. решение и ссылки:

https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

16. Authentication Request Identified

Уровень серьезности: Средний

Статус: Обнаружена

Описание

The given request has been identified as an authentication request. The 'Other Info' field contains a set of key=value lines which identify any relevant fields. If the request is in a context which has an Authentication Method set to "Auto-Detect" then this rule will change the authentication to match the request identified.

Решение

This is an informational alert rather than a vulnerability and so there is nothing to fix.

Рекомендации

См. решение и ссылки:

https://www.zaproxy.org/docs/desktop/addons/authentication-helper/auth-req-id/