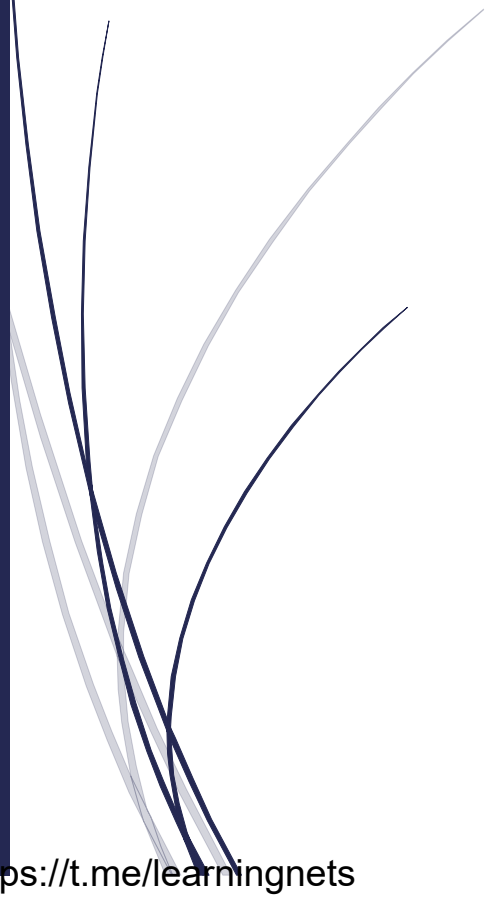
12/21/2020

OWASP ASVS 4.0 testing guide



BlazingWind

https://github.com/BlazingWind/OWASP-ASVS-4.0-testing-guide

https://t.me/learningnets

Contents

https://t.me/learningnets

ZAP set up for ASVS testing

Prerequisites

This guide assumes that you have OWASP ZAP installed and are able to access the graphical user interface (as opposed to using ZAP in headless mode).

Configuration

OWASP ZAP has by default enabled scripts and scan rules that should be disabled if you would like to receive a report which would include only alerts triggered by scripts testing OWASP ASVS controls. That includes action such as:

- Creating a context for your domain, so only the domains that are specified by you are scanned and no others.
- Disabling passive scan rules that are not part of the test
- Creating a policy for active scan scripts, in which only the scripts provided will be executed

Create a context for your domain

A context in ZAP defines how ZAP will work with certain websites. With specifying several contexts, you may separate issues that affect your website from issues that come from the third-party providers. This has to be look upon with a grain of salt - while the issue might have been found with a resource coming from a 3rd party provider, the issue may lay in how the resource is implemented on your website. When doing a scan of a website (especially using a spider or an active scan), configuring a context is a must - there you define:

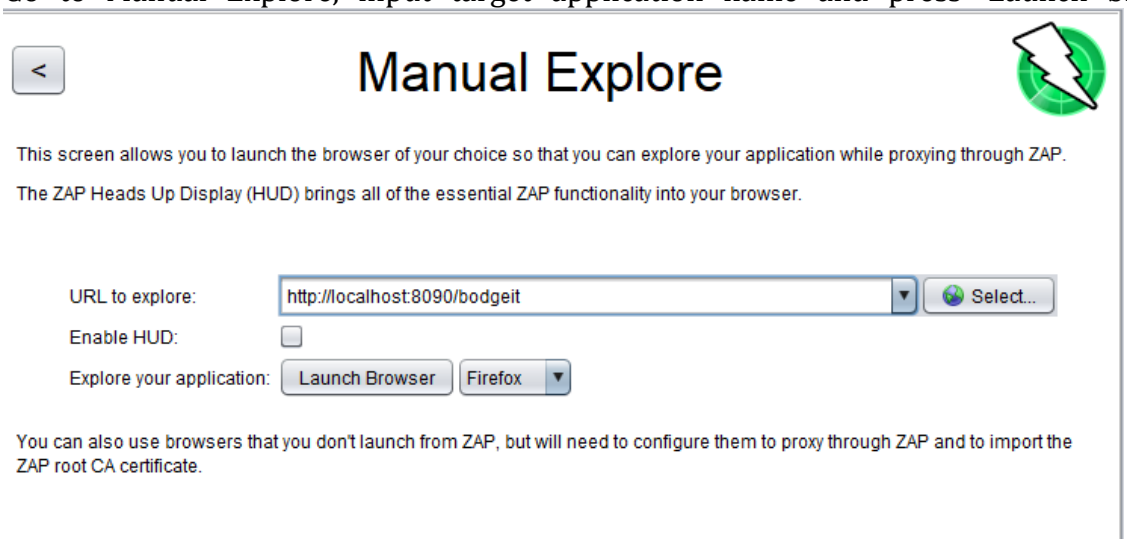
- scope - which websites to scan and which should explicitly be excluded
- authentication and users that should be used for authentication
- session management

A properly configured context will make the spider and scan only target the websites in scope, without attacking ones that may belong to third party. If authentication and session management is configured, the spider and the scanner will be able to automatically authenticate if a session token has expired or the spider accidentally logged out - this will provide a better coverage of the tested site.

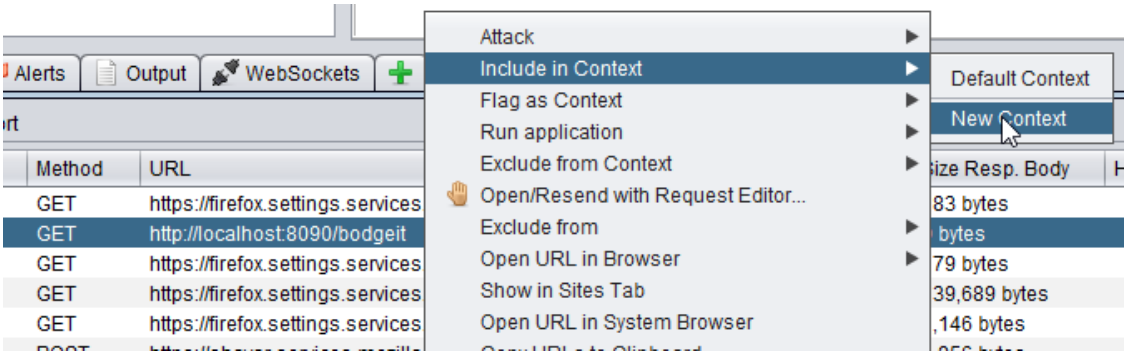
The easiest way to configure a context is after you have accessed the target domain with a built-in browser or preconfigured one.

https://t.me/learningnets

1. Go to Manual Explore, input target application name and press ‘Launch browser’.

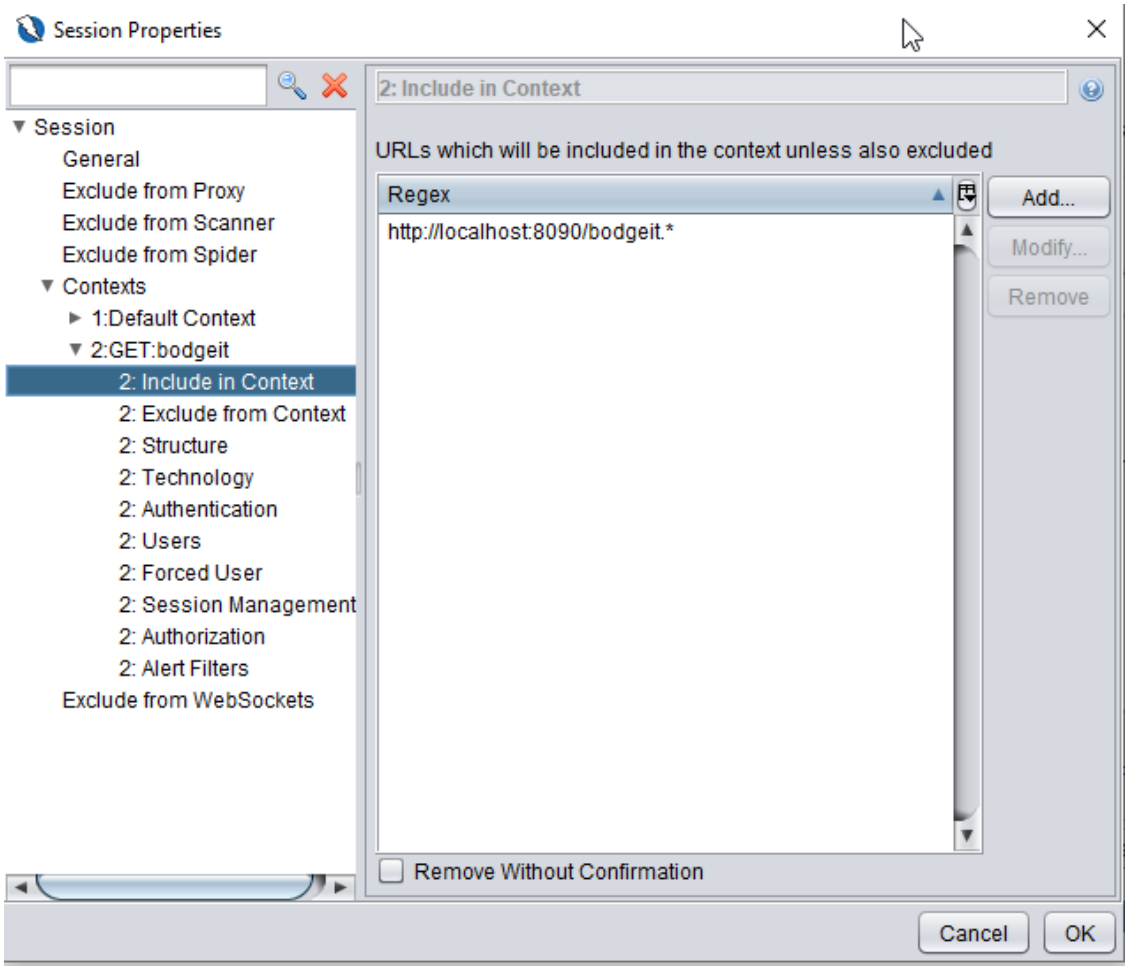


2. Wait for the browser and the application to load.
3. Left-click on the entry and choose Include in Context > New Context



4. A pop up box will appear where you can configure your new context. There has been added a wildcard to the address, which means that everything that comes after bodgeit will be in scope. If there are any other sites that you would like tested you may add them here.

https://t.me/learningnets

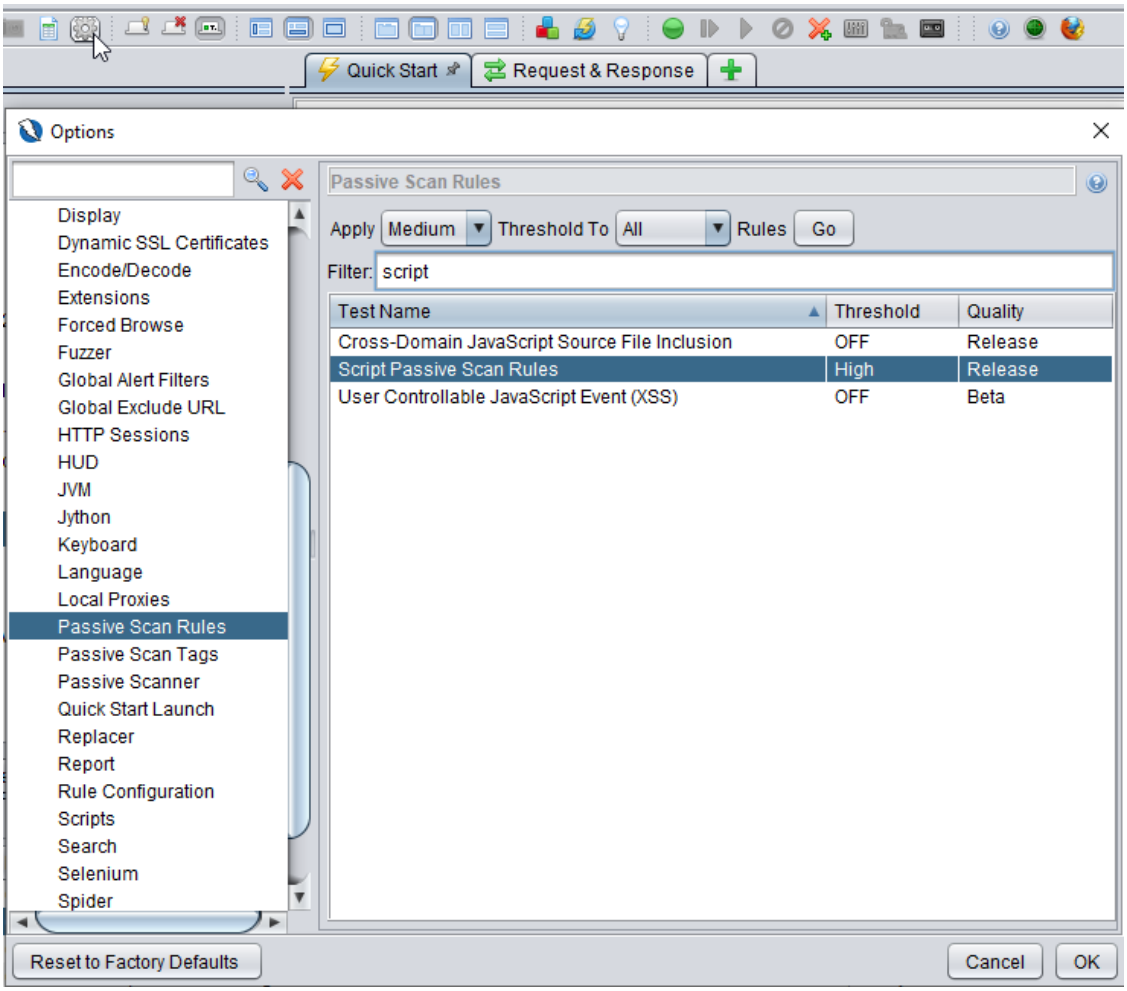


5. Exclude the subdirectories from target website or addresses that you wish not to be targeted.
6. If you wish to configure authentication and session management, you may do it here. Refer to videos on ZAP's website for configuring those: https://www.zaproxy.org/videos/ and articles such as: https://dzone.com/articles/scripting-authenticated-login-within-zap-vulnerabi They are a brilliant resource if you would like to get up to speed with using ZAP.
7. Now that you are done with configuring the context you can apply it in history tab by clicking the red target button. Remember you can save a context and import it at a later point, so you don't have to go through above steps again (unless there have been significant changes to the application you are testing).

Passive Scan rules

1. Click Options icon, go to Passive Scan Rules and choose 'Apply OFF Threshold to All'.

https://t.me/learningnets



2. Go to Passive Scanner and check 'Scan only messages in scope'

3. Click Marketplace icon, search for OWASP ASVS passive scan rules and install Jython extension. Install also RetireJS add-on. #If the scripts are made into an extension, then install it

4. Check in Options > Passive Scan rules. Search for “Script Passive Scan rules” and set Threshold is to ‘Medium’. Search for “Vulnerable JS library” and set Threshold to ‘Medium’.

5. Click the green plus next to ‘Sites’ tab > click ‘Scripts’ > choose ‘Passive Rules’ > enable all scripts that came from this repository (or only the ones that you would like to test) *If you have any scripts that reside on your computer, you can import them by using the button Import here.*

6. Go to Quick Start > Automated Scan and input the address of the domain you wish to scan.

7. Tick ‘Use traditional spider’ and click the ‘Attack’ button. If you wish for more coverage of your website, tick ‘Use ajax spider’ and choose one of the browsers you have installed.

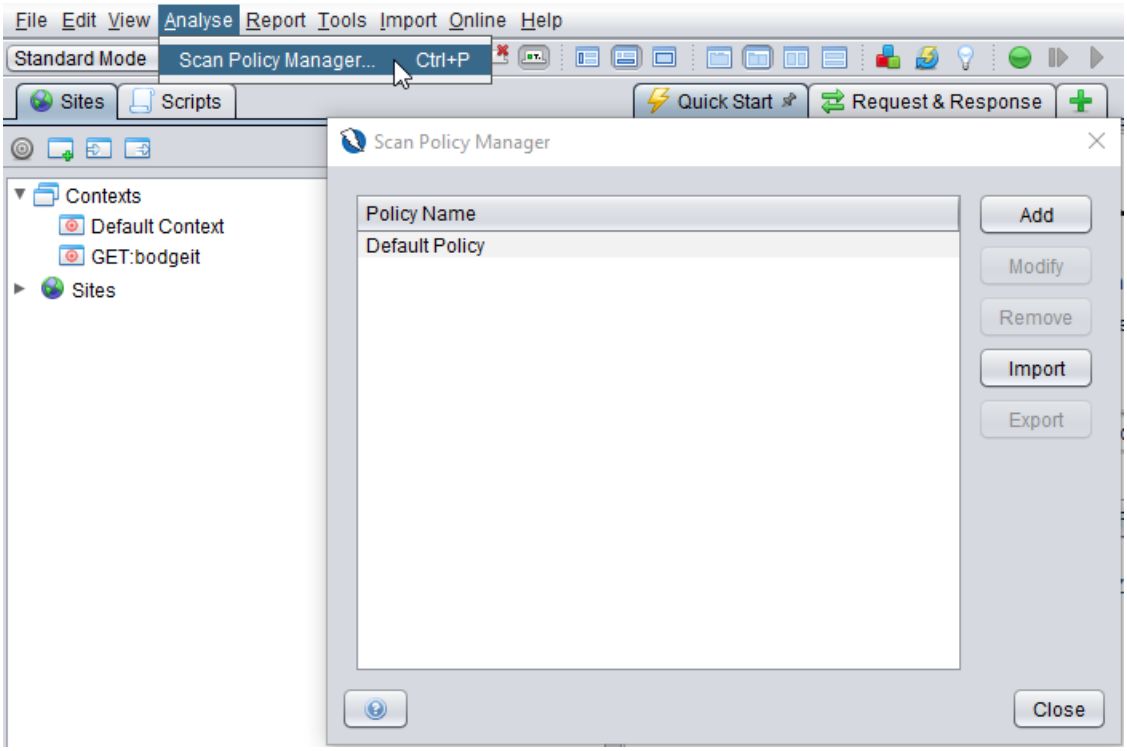
https://t.me/learningnets

8. The spider shouldn't take longer than a few minutes to run. If you have enabled the ajax spider it will take longer. When the spider(s) have finished:
9. In the lower right corner you will see several symbols, including an eye. Wait until the number goes down to zero - that means that passive scanning has finished. Depending on the size of your application, it may take a long time, so set it aside for an hour just in case.
10. Go to Report > Generate HTML report. If you prefer the folder in any other format, choose other options - Markdown, XML, JSON. If you have chosen HTML report, you will be presented with the report will all the controls.

Active Scan

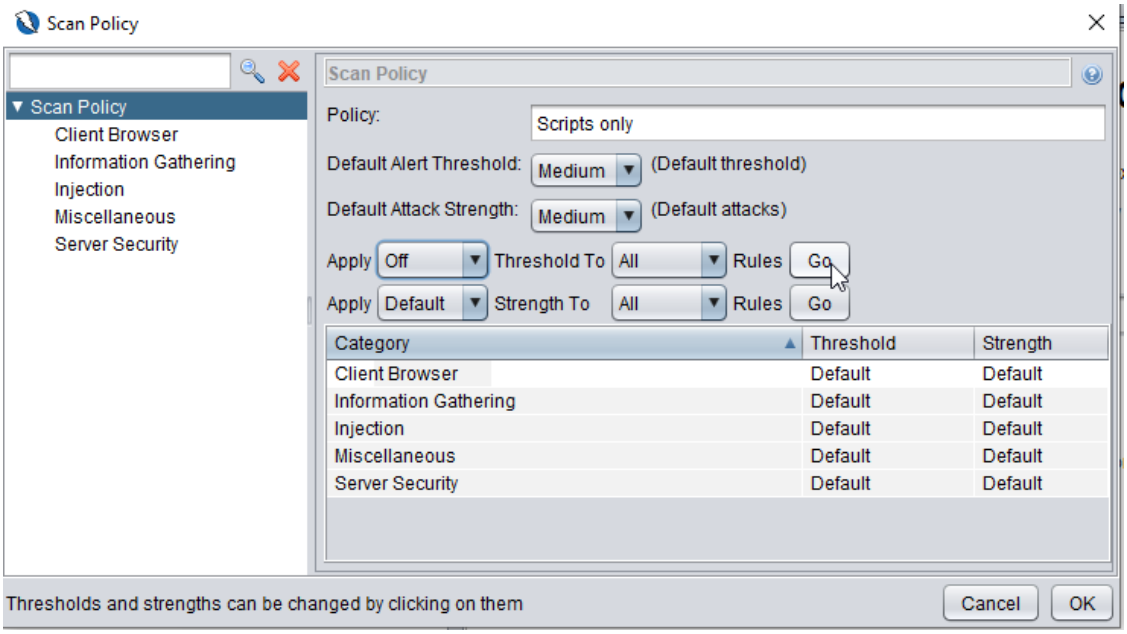
Policy

1. To scan an application using Active Scan scripts go to Scripts > Active Scan and enable all scripts that you wish to use.
2. Go to Analyse > Scan Policy Manager… You will see a pop up box. Click Add

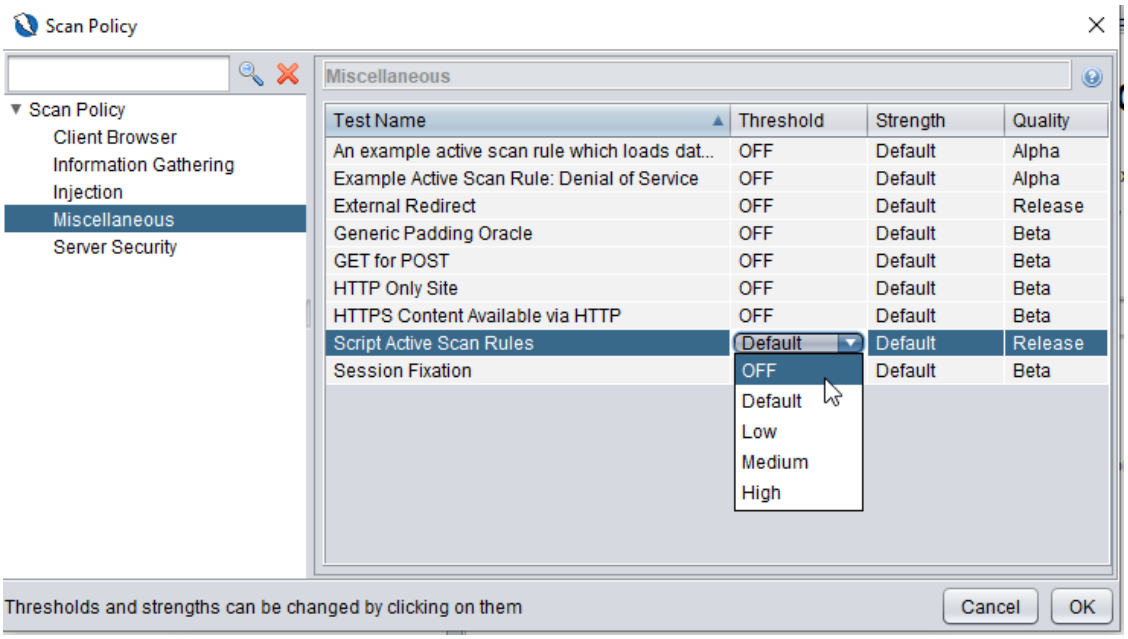


3. Another pop up box will appear. Name your policy "Scripts only" and choose "Apply 'Off' Threshold to 'All'" and press Go.

https://t.me/learningnets



4. Go to Miscellaneous and choose your preferred threshold - for example Medium. Then click OK.



Active Scanning

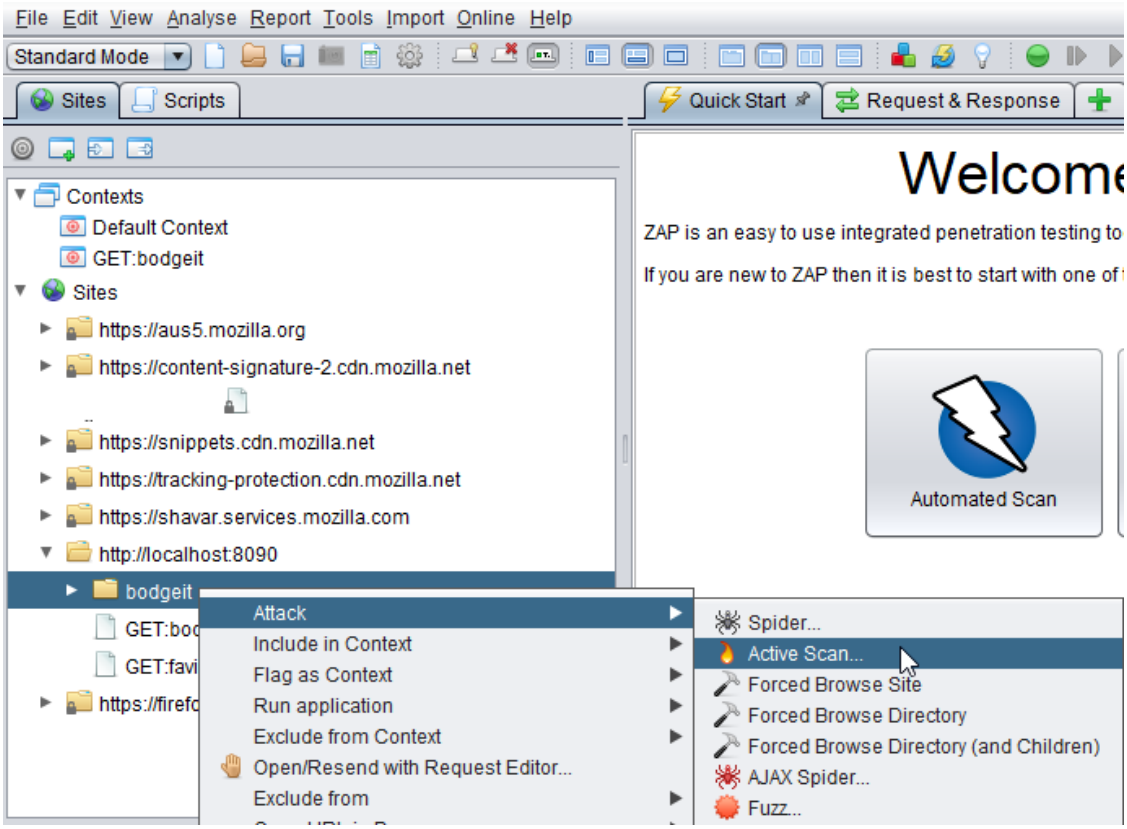
Place the scripts under C:\<your user>\ZAP and then import them by going to ZAP > click the green plus next to 'Sites' tab > click 'Scripts' > click the 'import' folder icon > choose active > choose the desired scripts to load.

It is preferable that you first run a spider before using active scanning for better coverage.

1. Next to Sites tab you will see a Green Plus sign. Press it and choose Scripts. You will see a new tab with example scripts you can use.

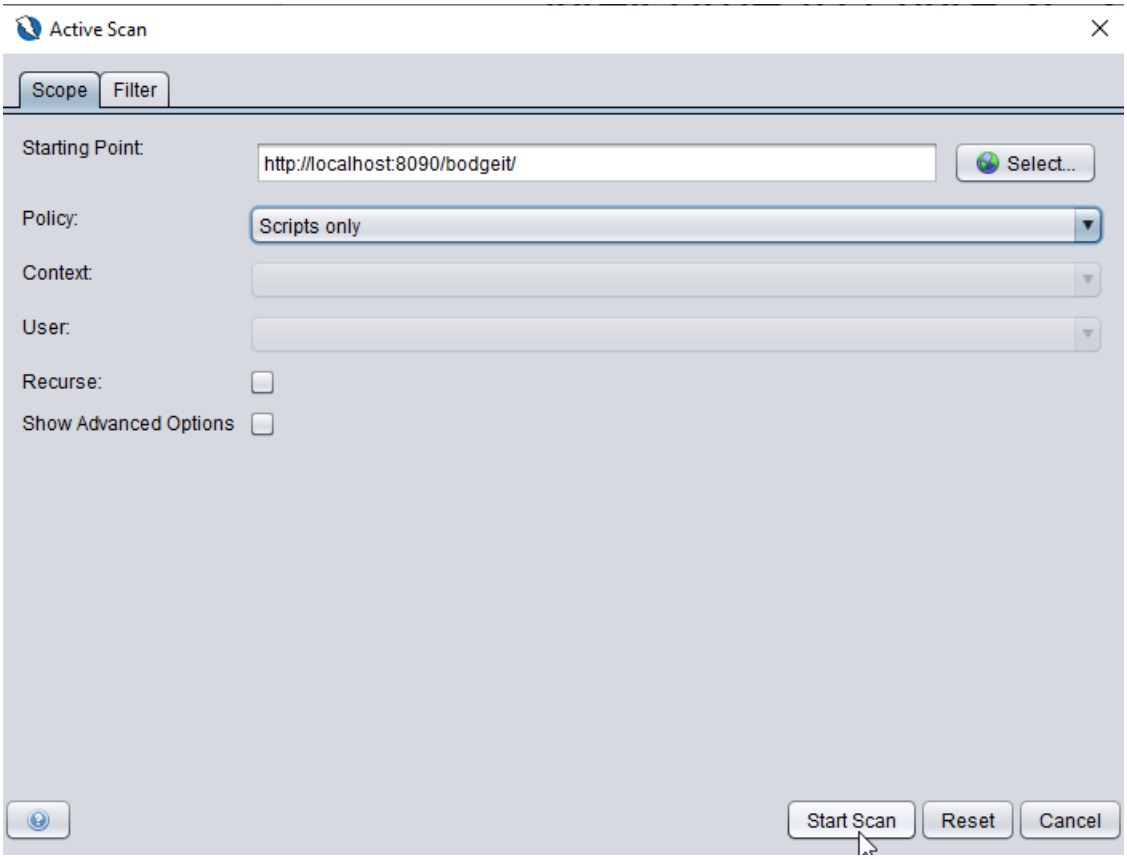
https://t.me/learningnets

2. Go to Active Scan and enable the scripts you wish to scan with.
3. Go back to Sites tab and choose the domain you wish to scan. Left click, choose Attack > Active Scan…



4. A pop up box will appear. Choose script policy “Scripts only” that you configured earlier. If you want to scan the node you have selected in the sites tree, then uncheck “Recurse” option. If you wish the active scan to be executed on all subnodes (subdirectories), leave the option checked. Press Start Scan and let it run for several minutes.

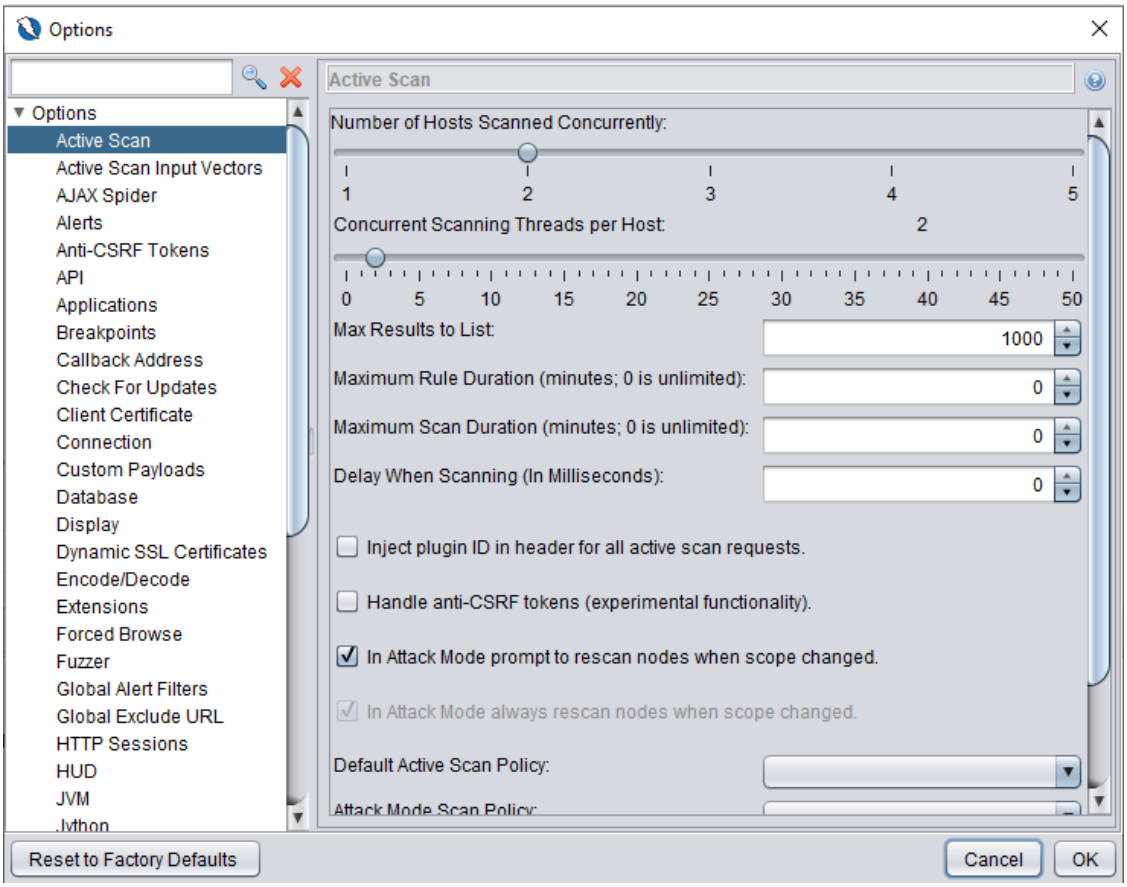
https://t.me/learningnets



(optional) How to slow down the scan

Go to Tools > Options > Active Scan. Here you may change 'Delay When Scanning (in Milliseconds) to 500 - which would be one request per half second. In similar way, if you want to limit how long the scan will last, you can set Maximum Scan Duration.

https://t.me/learningnets



https://t.me/learningnets

3. Session Management

3.1.1 Session tokens in URLs

3.2.1 New session token on authentication

3.2.2 Session token entropy

3.2.3 Token storage in browser

3.3.1 Logout

3.3.2 Re-authentication period

3.4.1 Cookie Secure attribute

3.4.2 Cookie HttpOnly attribute

3.4.3 Cookie SameSite attribute

3.4.4 Cookie Host prefix

3.4.5 Cookie Path attribute

3.7.1 Sensitive transactions

https://t.me/learningnets

3.1.1 Session tokens in URLs

> Verify the application never reveals session tokens in URL parameters or error messages.

CWE 598

Explanation

Due to the fact that URLs are usually recorded in logs, browser history and various other places, it increases the chances of them being captured by an attacker. Errors similarly will be stored in logs. Revealing session tokens in URLs increases the risk and makes it easier for an attacker to launch an attack.

Testing methods

ZAP script

Enable script “3-1-1 Session token in URLs” and browse to the site in scope. Preferably use a spider or and AJAX spider. The script looks for the string ‘token’, ‘jwt’, ‘session’ and ‘cookie’ in the URL or in a error page. If it is found it raises an alert. Note that the script may produce false positives, and it is important to review its findings.

Additionally, you may run active rule script 14-3-1 for better coverage. This script attempts to trigger a few errors.

Control

If session token are found in the URL parameters or error pages, the control is failed.

Resources

https://t.me/learningnets

3.2.1 New session token on authentication

Verify the application generates a new session token on user authentication.

CWE 384

Proactive Control C6

Explanation

If a new session token is not generated on user authentication, then the session tokens can be reused. It is one of the fundamental requirements for a web application that session tokens are generated anew on each user authentication.

Testing methods

ChromeDevTools

Cookies

Log in to the application and go to Application tab > Cookies > choose the name of the target domain. Note the session cookies being set - the name depends on application. For an application with PHP backend, it will probably be PHPSESSID. Usually they will be very easy to identify.

Log out and repeat the process. If you leave the ChromeDevTools open while logging in and out, you should see the cookies' values changing.

Tokens

Follow the same flow but look into the Network tab instead. After logging in, you should see an Authorization header sent with every request. Log out and log in again. Observe if the Authorization header changed.

ZAP

Similarly to ChromeDevTools, you can look at the traffic in ZAP or other proxy. After logging in, you will see the request for login sent in the History tab. In following requests and responses there will be either the cookies or session tokens. Log out and log in again. Observe, if the session token has changed.

Control

If the application does not set new session token on authentication, the control is failed.

Resources

https://dev.to/thecodearcher/what-really-is-the-difference-between-session-and-token-based-authentication-2o39

https://t.me/learningnets

3.2.2 Session token entropy

Verify that session tokens possess at least 64 bits of entropy.

CWE 331

Proactive Control C6

Explanation

Session tokens with a small entropy leave the application open for bruteforcing/session guessing attacks. Assuming that a good Cryptographically Secure Pseudorandom Number Generator is used, the session token value can be estimated to have entropy half the length of the session token. A token 128 bits (16 bytes) in size would give around 64 bits of entropy. While there are cases in which entropy is calculated differently (see resources), for simplicity assume that the length of a session ID should be at least 128 bits (16 bytes). To calculate entropy accurately, one would need a sample of 10 000 tokens to find the character list, to check whether the tokens use pre defined strings and if the random generator really is random. Be sure to check if the token is Base 64 encoded, since then the entropy will be calculated differently.

Testing methods

ChromeDevTools

Cookies

Similarly to 3.2.1, log in and go to Application > Cookies > target domain and find session cookies. Log out and log in again, to see how much the cookies changed. If the application does not set the cookie randomly, you might see repeating strings in the cookie. To calculate the length of the cookie, you may use Python 3:

```
$ python3
Python 3.8.2
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> cookie = "DARFndj49sac-0bL6GWN5SdAFOdJTw-
r4FvUBrE9O_bgPiBVKn7xkMjMUOluw_uBTa2R8fLHUDjnKmZ6Bh-
3U746yLrAxeywReyIYv0U479JV9FH61b02wFzRlm6V-
QxwSUdb0ZK4RsdirwpWaaEHmsc9D6ZHj5fvEjVw7XM"
>>> len(cookie)
168
```

Tokens

Follow the same flow, but look into the Network tab instead. After logging in, you should see an Authorization header sent with every request. If the token is a JWT, it does not make sense to calculate its entropy - in this case mark the control as N/A.

https://t.me/learningnets

ZAP

Similar to ChromeDevTools, you can look at the traffic in ZAP or other proxy. After logging in, you will see the request for login sent in the History tab. In following requests and responses there will be either the cookies or session tokens. Log out and log in again. Observe, how much the session token has changed.

Control

If the session token is shorter than 128 bits or not randomly generated, the control is failed.

Resources

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

https://owasp.org/www-community/vulnerabilities/Insufficient_Session-ID_Length

https://security.stackexchange.com/questions/138995/why-would-the-session-id-entropy-only-be-half-of-the-length-of-the-session-id

https://gist.github.com/4k1/6fbe670807db1d48407685d6cc46b0af

https://www.pleacher.com/mp/mlessons/algebra/entropy.html

https://t.me/learningnets

3.2.3 Token storage in browser

> Verify the application only stores session tokens in the browser using secure methods such as appropriately secured cookies (see section 3.4) or HTML 5 session storage.

CWE 539

Explanation

In a longer discussion on their Github (see references) the ASVS creators have decided to change requirement 3.2.3 to mean that token-based session should only use SessionStorage, and not LocalStorage. Cookies will have their own requirements in section 3.4 and are not relevant for this control.

Tokens stored in LocalStorage do not expire, unless they are set with an expiration date. The caveat is that the token does not get deleted after expiration time, but next time the user access the website.

Tokens stored in SessionStorage get cleared after a user closes their browser; the expiration time for SessionStorage is dependant on when the page session ends. Page session end happens when a tab is closed, a browser is closed or a new tab that creates another SessionStorage object is opened. As such, tokens stored here do require re-authentication. There have been many debates on using LocalStorage - from secure session management point of view, SessionStorage is considered more secure due to deletion of everything stored there, while LocalStorage allow for data to be stored indefinitely. As such OWASP does not recommend using LocalStorage for storing sensitive data, such as tokens.

Testing methods

ChromeDevTools

Search under Application > LocalStorage > target domain if there are any session tokens being stored.

Control

If session tokens are stored under LocalStorage, the control is failed.

Resources

https://blog.gds-gov.tech/our-considerations-on-token-design-session-management-c2fa96198e6d

https://dev.to/rdegges/please-stop-using-local-storage-1i04

https://github.com/OWASP/ASVS/issues/696#issuecomment-626231907

https://snyk.io/blog/is-localstorage-safe-to-use/

https://t.me/learningnets

3.3.1 Logout

Verify that logout and expiration invalidate the session token, such that the back button or a downstream relying party does not resume an authenticated session, including across relying parties.

CWE 613

Proactive Control C6

Explanation

Similarly to 3.2.1, where a new token should be generated on each user authentication, this control specifies that session tokens should be invalidated on logout, to prevent reusing session tokens. This is another fundamental requirement within session management.

Testing methods

Log in, log out and try to use the back button. You should receive an error, if the token got invalidated. Bear in mind that if this action is successful, it may be due to insecure cache configuration. To be safe you may check the tokens with DevTools:

ChromeDevTools

Log in to the application and open Application > Cookies > target domain. Find the cookies used for the session. Log out and observe whether the session cookies are deleted. If tokens are stored in WebStorage, follow the same process for Application > Local Storage > target domain and for Application > Session Storage > target domain.

Control

If the logout or expiration does not invalidate the token, the control is failed.

Resources

https://t.me/learningnets

3.3.2 Re-authentication period

> If authenticators permit users to remain logged in, verify that re-authentication occurs periodically both when actively used or after an idle period. L1 - 30 days

CWE 613

Proactive Control C6

Explanation

A user remaining logged gives a possibility to an attacker to be able to use a user's session without the need to know the password, e.g. if an attacker stole a work laptop that was left unlocked, the person will be able to use the application as the victim. Periodical re-authentication prevents this and many other attacks from happening.

Testing methods

Testing that re-authentication happens after 30 days may be tricky to work with. A cookie should however have an expiration date sent with every request. For tokens stored in local or session storage the situation is a bit more complicated, and the expiration time might not be set at all.

Tokens stored in LocalStorage do not expire, unless they are set with an expiration date - a time to live which can be set for example to 30 days (in seconds). The caveat is that the token does not get deleted after expiration time, but next time the user access the website.

Tokens stored in SessionStorage get cleared after a user closes their browser; the expiration time for SessionStorage after a page session ends. Page session end happens when a tab is closed, a browser is closed or a new tab that creates another SessionStorage object is opened. As such, tokens stored here do require re-authentication. At last, SessionStorage is not as much used as LocalStorage.

If the reauthentication period is very short (e.g. 30 minutes) then the answer to the control failure would show up during testing of the application, since the tester would have to log in again.

ChromeDevTools

Cookies

Identify the session cookie under Application > Cookies > target domain and look for its expiration date.

Tokens

Identify the session token under Application > LocalStorage > target domain, and look for the timestamp. If there is no timestamp on the session token, it means that it will be stored in the browser indefinitely.

https://t.me/learningnets

Control

If the expiration date is longer than 30 days, the control is failed.

Resources

https://blog.bitsrc.io/localstorage-sessionstorage-the-web-storage-of-the-web-6b7ca51c8b2a

https://stackoverflow.com/questions/13011944/make-localstorage-or-sessionstorage-expire-like-cookies?noredirect=1&lq=1

https://www.w3schools.com/html/html5_webstorage.asp

https://stormpath.com/blog/where-to-store-your-jwts-cookies-vs-html5-web-storage

https://t.me/learningnets

3.4.1 Cookie Secure attribute

Verify that cookie-based session tokens have the 'Secure' attribute set.

CWE 614

Proactive Control C6

Explanation

Setting Secure attribute on a session cookie instructs a browser to send the cookie only over an encrypted HTTPS connection. Even if a site is configured to use only HTTPS, the web browser can be tricked to disclose the cookie over an unencrypted protocol. Sites using HTTP can't set cookies with this attribute.

Testing methods

Cookies

Log in to the application and go to Application tab > Cookies > choose the name of the target domain. Find the cookies used as session tokens and check if they have the Secure attribute set.

ZAP

Similar to ChromeDevTools, you can look at the traffic in ZAP or other proxy. After logging in in a built in or preconfigured browser, you will see the request for login sent in the History tab. In following requests and responses you will find cookies.

Control

If the cookie-based session tokens do not have Secure attribute set, the control is failed.

Resources

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

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

https://t.me/learningnets

3.4.2 Cookie HttpOnly attribute

Verify that cookie-based session tokens have the 'HttpOnly' attribute set.

CWE 1004

Proactive Control C6

Explanation

HttpOnly attribute makes a cookie inaccessible to Javascript, which helps mitigate XSS attacks.

Testing methods

Cookies

Log in to the application and go to Application tab > Cookies > choose the name of the target domain. Find the cookies used as session tokens and check if they have the HttpOnly attribute set.

ZAP

Similar to ChromeDevTools, you can look at the traffic in ZAP or other proxy. After logging in in a built in or preconfigured browser, you will see the request for login sent in the History tab. In following requests and responses you will find cookies.

Control

If the cookie-based session tokens do not have HttpOnly attribute set, the control is failed.

Resources

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

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

https://owasp.org/www-community/HttpOnly#:~:text=What is HttpOnly%3F,if the browser supports it

https://t.me/learningnets

3.4.3 Cookie SameSite attribute

Verify that cookie-based session tokens utilize the 'SameSite' attribute to limit exposure to cross-site request forgery attacks.

CWE 16

Proactive Control C6

Explanation

SameSite attribute has been implemented as protection against CSRF attacks. It can have one of three values: Strict, Lax and None. Strict means that the browser will never send a cookie with a cross-origin requests. Lax will send cookies with requests that user clicked on. None allows for sending cross origin requests, but only in secure contexts - it means that a cookie must also have Secure attribute set. This control requires that the cookie is set with either Strict or Lax.

In January 2020 Google Chrome has attempted to set all cookies in its browser with Strict attribute. Many enterprise websites have stopped working properly and Chrome reverted the change and introduced Lax value as a default value that a cookie will be set with, if no other value is set. Other browser vendors followed.

Testing methods

Cookies

Log in to the application and go to Application tab > Cookies > choose the name of the target domain. Find the cookies used as session tokens and check if they have the SameSite attribute set.

ZAP

Similar to ChromeDevTools, you can look at the traffic in ZAP or other proxy. After logging in in a built in or preconfigured browser, you will see the request for login sent in the History tab. In following requests and responses you will find cookies.

Control

If the cookie-based session tokens do not have SameSite attribute set, the control is failed.

Resources

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

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

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite

https://medium.com/@renwa/bypass-samesite-cookies-default-to-lax-and-get-csrf-343ba09b9f2b

https://t.me/learningnets

3.4.4 Cookie Host prefix

Verify that cookie-based session tokens use "__Host-" prefix (see references) to provide session cookie confidentiality.

CWE 16

Explanation

__Host prefix can be set only on cookies with Secure flag, that use encrypted HTTPS, have no domain specified (cannot be set to subdomains) and have the Path set to “/”.

Testing methods

Cookies

Log in to the application and go to Application tab > Cookies > choose the name of the target domain. Find the cookies used as session tokens and see if they have the __Host prefix

ZAP

Similar to ChromeDevTools, you can look at the traffic in ZAP or other proxy. After logging in in a built in or preconfigured browser, you will see the request for login sent in the History tab. In following requests and responses you will find cookies.

Control

If the cookie-based session tokens do not have __Host prefix set, the control is failed.

Resources

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

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

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie#Directives

https://t.me/learningnets

3.4.5 Cookie Path attribute

Verify that if the application is published under a domain name with other applications that set or use session cookies that might override or disclose the session cookies, set the path attribute in cookie-based session tokens using the most precise path possible.

CWE 16

Explanation

This is only applicable to domains that are published with several applications. Setting the Path attribute to be very precise will mitigate errors that might come form cookies being overridden.

It may be easier to present an example - a website having three applications that require login: a bank, a shop and a social media site. They would reside on:

example.com/bank

example.com/shop

example.com/socialmedia

Cookies for the bank would have a Path attribute set to /bank, shop to /shop, and the social media app to /socialmedia. This setup would prevent issues with cookie disclosure and errors.

Testing methods

Cookies

Log in to the application and go to Application tab > Cookies > choose the name of the target domain. Find the cookies used as session tokens and check their Path.

ZAP

Similar to ChromeDevTools, you can look at the traffic in ZAP or other proxy. After logging in in a built in or preconfigured browser, you will see the request for login sent in the History tab. In following requests and responses you will find cookies.

Control

If an application is published under a domain with other applications and the cookies do not use precise Path, the control is failed.

Resources

https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies#:~:text=The Path attribute indicates a,%2Fdocs

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

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

https://t.me/learningnets

3.7.1 Sensitive transactions

Verify the application ensures a valid login session or requires reauthentication or secondary verification before allowing any sensitive transactions or account modifications.

CWE 778

Explanation

A session token may be stolen by an attacker and thus be used to perform fraudulent actions as the victim. Having a secondary verification before commencing sensitive actions is a strong security preventive control that will help stop many attacks.

Testing methods

Identify all areas which could be considered sensitive - making a payment or a wire transfer, changing a password or email, changing payments methods… and check if a user is presented with secondary verification.

Control

If sensitive actions do not require secondary verification, the control is failed.

Resources

https://t.me/learningnets

4. Access Control

4.1.1 Client-side trust

4.1.2 Data attributes

4.1.3 Principle of least privilege

4.1.4 Principle of deny by default

4.1.5 Secure failure

4.2.1 Direct object attacks

4.2.2 CSRF protection

4.3.1 Admin MFA

4.3.2 Directory browsing

When approaching a target, it's preferable to test all controls in one section at once - often output from one control will be helpful in determining whether other controls were successful.

All controls from Access Control section could be tested with a new feature in ZAP for testing access control and by looking into the wayback machine. To make the controls clearer to read, the descriptions is provided below.

ZAP - access control testing

ZAP has recently added a new feature - access control testing, which is right now in Alpha. It allows for testing what a defined user can access and is an excellent way to determine if specific users have access to more than needed. It requires however, that the tester configures authentication, session management and access control need to be configured. The process may be easy or very challenging - it depends solely on how the application handles authentication. In case of security regression testing or for periodic controls of access control, configuring it may be very useful. More details can be found at: https://www.zaproxy.org/docs/desktop/addons/access-control-testing/

The wayback machine

Oftentimes a lot of information about an application can be found via the wayback machine, which saves snapshots of applications over time. The archived version of the application may give access to resources that can help with later testing. Consider looking for robots.txt files - the file may have changed over the years, but the endpoint that are defined in it might still be accessible on the internet. Check more ideas under https://www.bugbountyhunter.com/guides/?type=waybackmachine and https://medium.com/@ghostlulzhacks/wayback-machine-e678a3567ec

https://t.me/learningnets

4.1.1 Client-side trust

Verify that the application enforces access control rules on a trusted service layer, especially if client-side access control is present and could be bypassed.

CWE 602

Explanation

This control encompasses any client-side access control. A proper way to enforce access control would be by duplicating checks that are made on client side, also on server-side. Even if there are access control checks on client-side, anything coming from the browser (the user) should not be trusted.

It's easiest to explain the control by an example - a user is presented with a standard website (on presentation layer), which does not contain a link to admin panel. The fact that there is no direct link to the admin panel is the "access control" presented in the above requirements. But it is very easy to find the admin panel. Let's say it is found under "/admin-panel" and anyone is able to access it, as long as the person knows where it resides. An attacker may very easily find it by bruteforcing directories.

The example application does not enforce access control rules on a trusted service layer (it enforces them on presentation layer), and the client-side access control can be bypassed.

Another example: when trying to change a user's password the client side code checks if the user is authenticated, and then presents the user with the page for changing that user's password. But the server side does not check if the user is first authenticated and just processes the request to change the password. It essentially allows an attacker to change any user's password unless there are any other controls in place.

Testing methods

Testing depends on what functionality a user is allowed in the application. Usually there are at least several sensitive actions that should first undergo an access control check, such as changing the password, making transactions, accessing sensitive files, writing comments in a private group in a social media app and more. Additionally, there should be controls that ensure that a user is not allowed to do any actions as another user.

Since above we have established which sensitive actions to look for, let's continue to where we can find them. For anything that requires a change (change of password, making transactions, writing comments) observe the traffic through a web proxy and look for POST requests. Pay attention to the what is in the body of the request, but also look for what parameters are found in the URL. In addition you may use script 5-1-1 HTTP parameter pollution script in OWASP ZAP in connection with a spider.

ZAP - bruteforcing directories

Choose the target website in the Sites tree > right-click choose Attack > Forced Browse Site. A new tab will open. Check that you have chosen the right site, then choose in the drop-down list

https://t.me/learningnets

“directory-list-1.0.txt”. You may also add other directory bruteforcing lists such as SecLists in Options > Forced Browse.

Additionally, refer to ZAP access control testing feature, which is described in the summary of the Access Control section.

Control

Bruteforce directories of a target application and do a manual review of what was found. If there are any directories or areas such as admin panels that are not protected by access control, the control is failed.

Resources

https://security.stackexchange.com/questions/196755/how-should-i-interpret-access-controls-on-the-presentation-layer-are-enforced

https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control

https://cheatsheetseries.owasp.org/IndexASVS.html#v4-access-control-verification-requirements

https://github.com/danielmiessler/SecLists

https://t.me/learningnets

4.1.2 Data attributes

Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized.

CWE 639

Explanation

Sometimes access control is handled by a key value that a user has access to, such as a hidden field, a value in a query parameter or an unencrypted cookie variable. The point of this control is to identify any parameters, cookies and form fields that could be used to bypass access control.

Testing methods

ZAP helper scripts

Enable script “4-1-2 Hidden fields.py” and “5-1-1 HTTP parameter pollution.py” and browse to the site in scope. Those scripts find most out in the application if you use it with a spider and active scan scripts when authenticated - but you may also manually login, fill out forms visible on the site and use all search fields. The first script searches html for hidden fields and raises an alert on any fields it has found. The second script looks for any parameters in URLs and in POST form fields and raises an alert on all parameters it has found. Review findings of the scripts. If there is an URL with parameter ‘id’ or a similar name, can you change its value and access areas that should not be accessible to an unauthenticated person? Review found query parameters and hidden fields in a similar way.

Control

If any parameters, cookies and form fields that could be used to bypass access control, the control is failed.

Resources

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

https://t.me/learningnets

4.1.3 Principle of least privilege

> Verify that the principle of least privilege exists - users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection against spoofing and elevation of privilege.

CWE 285

Proactive Control C7

Explanation

Oftentimes an application will have a strict authentication check, but some functionalities might not have a good authorization check. This control builds up on top of the previous two in the category and introduces the principle of least privilege, which says that users should only have access to functionality and files that enable them to do their work.

Additionally, it may happen that documents or files that should only be for internal use of a company, are laid out in a way that a search engine is able to find it.

Testing methods

Configure authentication in a context for a target website and use bruteforcing available in ZAP. The capabilities of the access control testing will make the testing also easier. Similar to 4.1.1, find sensitive functionality that may not enforce principle of least privilege. You may also want to interview developers or managers behind the application.

ZAP - bruteforcing directories

Choose the target website in the Sites tree > right-click choose Attack > Forced Browse Site. A new tab will open. Check that you have chosen the right site, then choose in the drop-down list “directory-list-1.0.txt”. You may also add other directory bruteforcing lists such as SecLists in Options > Forced Browse.

Robots.txt

Robots.txt is a file that is supposed to tell search engines and web crawls not to index specific subdirectories of a website. This file might give ideas as to what subdirectories might contain sensitive data. Try to access the listed directories and see if they should not be allowed to be accessed. If there are any wildcards as part of the disallowed entry, you may consider writing a script which would create a list for bruteforcing directories and use it with ZAP Forced Browse Site.

Search engine dorks

Use search engine dorking to review whether the target websites leaks documents or files that were shared unintentionally to the outside. It may happen that you discover a directory that is unintentionally shared with the internet or discover a functionality that does not perform a

https://t.me/learningnets

precise authorization check. Usually you will find out information either about the backend or a misplaced document, which may give you ideas on how to write more specific dorks.

1. Inurl dork will search for URLs containing a string `site:target.com`
2. Site dork will only search within one site `inurl:login.php`
3. Filetype or ext dork will display files with a specific extension `inurl:target.com filetype:pdf` try to use several different extensions to find out as much as you can about folders which may store sensitive documents - then try to access those folders and documents laying in them.
4. By specifying a keyword to look for `inurl:target.com password` depending on which countries and which languages the site uses, you may want to use keywords like username and password in those languages.

Additionally, refer to ZAP access control testing feature, which is described in the summary of the Access Control section.

Control

If principle of least privilege is not enforced, the control is failed.

Resources

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

https://medium.com/nassec-cybersecurity-writeups/exploring-google-hacking-techniques-using-google-dork-6df5d79796cf

https://t.me/learningnets

4.1.4 Principle of deny by default

> Verify that the principle of deny by default exists whereby new users/roles start with minimal or no permissions and users/roles do not receive access to new features until access is explicitly assigned.

CWE 276

Proactive Control C7

Explanation

Principle of deny is an easy way to ensure that users have the least privileges possible. While the number of permissions or access rights will very from application to application, there should be a model defining what rights do new users have.

Testing methods

Control 4.1.1 gives examples of sensitive actions and transactions. It is preferable to test this control via ZAP access control testing feature, which is described in the summary of the Access Control section, but it is also possible to check manually.

Manually

Register a new user. Access all available resources and try out functionalities (transferring money, viewing messages, posting messages in a private group, viewing other user's files, uploading a file). The minimal functionality that a new user needs will vary depending on the web application's intentional use. From what you have observed try to answer>

1. Was all the functionality that a new user was given required for using an application?
2. Was the user given access to minimal files and resources?
3. Was the user restricted from viewing other user's files, that they did not need to see?

For more ideas refer to WSTG,

Control

If the principle of deny is not enforced, the control is failed.

Resources

https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/05-Authorization_Testing/02-Testing_for_Bypassing_Authorization_Schema.html

https://t.me/learningnets

4.1.5 Secure failure

Verify that access controls fail securely including when an exception occurs.

CWE 285

Proactive Control C10

Explanation

Default error pages usually disclose information about the underlying software running on the web server. Optimally instead of default error pages, there should be a custom, user-actionable error pages returned.

Testing methods

During the course of testing Access Control section, there probably were found directories, resources or areas of the target website, which are sensitive. Try to access them from an unauthenticated and authenticated user point of view. Take a few test cases, preferably having two user accounts:

1. An authenticated user has access to a directory “target.com/personalportfolio”. While being unauthenticated, try to access that directory.
2. A website gives a possibility to upload a file as user A, which should be accessible only by user A. User B, knowing the URL and other information where the file is located, tries to view/edit the file.
3. User A gives user B view rights to view a file user A owns. But then user A decides to revoke access to the file. User B tries to access the file after revocation.

There are many more similar situations and other situations in which a user a user should be presented with a generic error response.

Control

If access controls do not fail securely, the control is failed.

Resources

https://t.me/learningnets

4.2.1 Direct object attacks

> Verify that sensitive data and APIs are protected against direct object attacks targeting creation, reading, updating and deletion of records, such as creating or updating someone else's record, viewing everyone's records, or deleting all records.

CWE 639

Explanation

Insecure Direct Object References are vulnerabilities, which happen when files or internal implementation objects have a key value identifier that a user has access to, such as a hidden field, a value in a query parameter or an unencrypted cookie variable. By changing a value in such a parameter and specifying a few other parameters, it may be possible to create, view or delete a record which a user has no authorization to. This control can be tested in conjunction with 4.1.1.

The only way to protect an application from IDORs is by using strict access control checks. At the same time, there are modern web frameworks, like Django, that do not have problems with this type of a vulnerability.

Testing methods

Take an example vulnerable bank application, in which you can make a transfer. The transfer is made by sending a form in a POST request.

```
POST bank.com/transfer HTTP/1.1
...

fromAccount=1111&toAccount=2222&amount=100
```

If an attacker could perform a MiTM attack, the person could change the value to of "toAccount" to their own.

Another example - a user want to update a file and the browser sends a request:

```
bank.com/fileview?file=1234&action=update
```

An attacker could craft a request to delete the file, although it does not belong to them:

```
bank.com/fileview?file=1234&action=delete
```

ZAP helper scripts

Enable script "4-1-2 Hidden fields.py" and "5-1-1 HTTP parameter pollution.py" and browse to the site in scope. Those scripts find most out in the application if you use it with a spider and active scan scripts when authenticated - but you may also manually login, fill out forms visible on the site and use all search fields. The first script searches html for hidden fields and raises an alert on any fields it has found. The second script looks for any parameters in URLs and in POST form fields and raises an alert on all parameters it has found. Review findings of the scripts. If there is an URL with parameter 'id' or a similar name, can you change its value and access files

https://t.me/learningnets

that should not be accessible to an unauthenticated person? Review found query parameters and hidden fields in a similar way.

Control

If it is possible to perform an IDOR on the web application, the control is failed.

Resources

https://www.acunetix.com/blog/web-security-zone/what-are-insecure-direct-object-references/

https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/05-Authorization_Testing/04-Testing_for_Insecure_Direct_Object_References.html

https://t.me/learningnets

4.2.2 CSRF protection

Verify that the application or framework enforces a strong anti-CSRF mechanism to protect authenticated functionality, and effective anti-automation or anti-CSRF protects unauthenticated functionality.

CWE 352

Explanation

Cross Site Request Forgery is an attack that tricks a user (a browser) into executing an action (changing password, making a wire transfer) in an application in which a user is logged in. It usually is conducted by tricking a user into clicking a link, which would execute a malicious request to the web server as a victim. Take an example of a bank:

A bank makes a transfer by making a request to the web server like this: `GET https://bank.com/transfer?toAccount=1111&amount=100`

The attacker crafts a request which would send money to his account, 2222:

```
GET https://bank.com/transfer?toAccount=2222&amount=100
```

And sends it as a phishing email. A victim is authenticated in a bank, receives a phishing email with a link and clicks it. Since the victim is authenticated, the bank will make a transfer from the victim's account to attacker's account. This is a simple example, but illustrates the concept well.

There are many ways to protect an application from CSRF and OWASP Cheatsheets gives a detailed overview of them. Currently the most effective solution is using CSRF tokens. CSRF tokens, called synchronizer tokens, are tokens that is generated by the server, embedded as a hidden field in every form on the webpage and sent together with a POST request with the form data. The server then compares the sent token against the generated one. CSRF tokens protects against CSRF, because the attacker would have to guess the token for the attack to work. These tokens are often built-in into many modern web frameworks.

Another solution is using SameSite flag for session cookies, which could be used as an additional layer of defense, next to CSRF tokens. SameSite flag prevents cookies from being sent if the origin - domain, port, protocol - is not related to the cookie. There are cases in which for a limited time itpossible to bypass the protection. For more information check control 3.4.3

Testing methods

ZAP helper script

Enable script "4-2-2 CSRF tokens.py" and browse to the site in scope. Preferably use a spider or and AJAX spider. The script looks for POST forms and searches for csrf tokens.

Bear in mind that while your application may be using CSRF tokens, if the token included in the form has a name does not contain the word "token" case insensitive, the alert will be raised. CSRF tokens are usually named "csrf-token", "_token","csrftoken" or similar and in this case using the word token as a way to detect if the synchronizer token is in use. Be sure to check the alert for false positives.

https://t.me/learningnets

The tested application might use other CSRF protection, so the alert being raised does not mean that it does not have any CSRF protection, but only that it does not use CSRF tokens.

For information on checking SameSite cookies see control 3.4.3.

Control

Resources

https://owasp.org/www-community/attacks/csrf

https://www.imperva.com/learn/application-security/csrf-cross-site-request-forgery/

https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.md

https://medium.com/@renwa/bypass-samesite-cookies-default-to-lax-and-get-csrf-343ba09b9f2b

https://t.me/learningnets

4.3.1 Admin MFA

Verify administrative interfaces use appropriate multi-factor authentication to prevent unauthorized use.

CWE 419

Explanation

Admin interfaces require particular protection against misuse.

Testing methods

Even if admin interface is accessible on the internet, it is not possible to tell if all people that authenticate via it have MFA set. It requires an interview with the business owners and developers.

Control

If any admin does not use MFA for logging in, the control is failed.

Resources

https://t.me/learningnets

4.3.2 Directory browsing

> Verify that directory browsing is disabled unless deliberately desired. Additionally, applications should not allow discovery or disclosure of file or directory metadata, such as Thumbs.db, .DS_Store, .git or .svn folders.

CWE 548

Explanation

Directory browsing is a way a server can display all its files instead of a webpage. When a browser requests a webpage, the web server returns an index file for a directory that the browser is trying to access and displays a webpage. If directory listing is turned on and the index file does not exist, the browser will instead display all its files that are available in the accessed directory. Currently directory listing is allowed only if the server is used as a file directory. The problem that comes with is that someone might either accidentally place there a sensitive file or forget to disable access to directory metadata. Recently, a very known misconfiguration is to leave .git file accessible on a web server - an attacker that obtains the .git file can reconstruct project's source. Leaving .git available to public is categorized as information disclosure. For modern web applications it is not usual to see a directory listing unless it is explicitly configured to show one.

Apache has had directory browsing of `/icons/` and `/icons/small/` enabled until recently, in which sysadmins placed sensitive files accidentally, which also lead to information disclosure.

Testing methods

Usually directory browsing is enabled on root of the webserver "example.com/" .It may be enabled for one specific folder, but it would most likely require either manually requesting each subdirectory or using a spider and an already implemented ZAP passive rule:

ZAP - passive rule

ZAP has a passive rule in Beta already implemented, which detects directory listings, but only for Apache 2 and IIS 7.5. To use it, install in the Marketplace Passive Beta rules. Then navigate to Options > Passive Scan rules > set Threshold for Directory Browsing rule to Medium.

https://www.zaproxy.org/docs/alerts/10033/

Run standalone script "4-3-2 Directory browsing" if you'd like to update the title and description to match ASVS control for generating a report.

ZAP bruteforcing

To test is files or folders such as .git, .svn exist, you can quickly bruteforce all directories with Forced Browse feature in ZAP. After using the spider, right click on target application > Attack > Forced Browse Site. To test only bruteforcing these, create a short word list containing the three names and add it to ZAP wordlists via Options > Forced Browse.

https://t.me/learningnets

Control

If the directory browsing is enabled or any of the named files are accessible, the control is failed.

Resources

https://www.sciencedirect.com/topics/computer-science/directory-browsing

https://medium.com/swlh/hacking-git-directories-e0e60fa79a36

https://www.acunetix.com/blog/articles/directory-listing-information-disclosure

https://pentester.land/tutorials/2018/10/25/source-code-disclosure-via-exposed-git-folder.html

https://bugs.ghostscript.com/show_bug.cgi?id=695858

https://hackerone.com/reports/142549

https://t.me/learningnets

5.1.1 HTTP parameter pollution

> Verify that the application has defenses against HTTP parameter pollution attacks, particularly if the application framework makes no distinction about the source of request parameters (GET, POST, cookies, headers, or environment variables).

CWE 235

Explanation

HTTP parameter pollution tests how the applications responds to multiple parameters with the same name. Since there is no RFC standard specifying how a web server should respond to multiple parameters, and every web server technology can define the default behavior. Some applications ignore the second instance of a parameter in a URL, some ignore the first, some concatenate them. Any of those behaviors can cause severe vulnerabilities.

Proper input encoding and input validation is sufficient to prevent HPP vulnerabilities. For many web applications, to prevent HPP the query string delimiter (&, ;) and its encoded versions should be filtered in GET, POST, cookies, headers, or environment variables as a minimum. As a general rule: if existing input validation and other security mechanisms are sufficient on single inputs, and if the server assigns only the first or last polluted parameters, then parameter pollution does not reveal a vulnerability. If the duplicate parameters are concatenated, different web application components use different occurrences or testing generates an error, there is an increased likelihood of being able to use parameter pollution to trigger security vulnerabilities.

Testing methods

ZAP helper script

Enable script “5-1-1 HTTP parameter pollution” and browse to the site in scope. Preferably use a spider or and AJAX spider. The script looks for any parameters in URLs and in POST form fields and raises an alert on all parameters it has found. Most of the parameters found will be innocuous, but it is important to find among them any that may be used in a malicious way - such as a parameter that symbolize a session token (‘session’, ‘jwt’, ‘id’, ‘auth’), ones that symbolize access (‘access’, ‘verify’, ‘retries’) or ones that may mean execution of a command or code (‘cmd’, ‘file’, ‘action’). There may be many more parameters which depends on the tested application.

While it is possible to run an automated test which would try to supply a parameter twice, it is not possible to detect whether application’s response poses a security risk, especially when testing server-side HTTP parameter pollution. Testing for HPP should be done manually. The test would require inputting parameters in GET, POST, cookies, headers and environment variables.

ZAP active scan rule

ZAP has an active scan rule which can help find some of the HPP client-side vulnerabilities, called ‘HTTP parameter pollution’, which is currently in Beta. To install and activate it, you first need to go to Marketplace > find Beta Active Rules > install. Then go to Active Scan Policy > choose a Policy > Injection > set Threshold for HTTP parameter pollution to Medium. Then scan an application using ‘recurse’ just like with any other active scan.

https://t.me/learningnets

Control

Review the helper script findings and find out how the application reacts to if any of the parameters poses a security risk.

Resources

https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/07-Input_Validation_Testing/04-Testing_for_HTTP_Parameter_Pollution.html

https://owasp.org/www-pdf-archive/AppsecEU09_CarettoniDiPaola_v0.8.pdf

http://www.madlab.it/slides/BHEU2011/whitepaper-bhEU2011.pdf

https://shahjerry33.medium.com/http-parameter-pollution-its-contaminated-85edc0805654

https://andresriancho.com/recaptcha-bypass-via-http-parameter-pollution/

https://www.ikkisoft.com/stuff/HPParticle.pdf

https://t.me/learningnets

9. Communication

9.1.1 Communication over TLS

9.1.2 Strong algorithms

9.1.3 TLS versions

https://t.me/learningnets

9.1.1 Communication over TLS

> Verify that secured TLS is used for all client connectivity, and does not fall back to insecure or unencrypted protocols.

CWE 319

Explanation

Allowing a site to connect using unencrypted protocols is a major security risk and should not be allowed. Unencrypted protocols give an attacker a possibility to sniff the traffic and see all sensitive data being sent.

Testing methods

Bash script

I have written a small tool in bash that utilizes openssl for testing, which tests against the three L1 controls in Network Security category of ASVS. To execute the script, run:

```
bash test.sh <domain to scan>
```

It prints out which controls are failed and which are successful.

Curl

To test if website connects over HTTP run:

```
curl -kis http://domain.com
```

- `-k` allows to proceed with an insecure connection
- `-i` includes HTTP response headers in the output
- `-s` is silent mode

Nmap

```
nmap -sV --script ssl-enum-ciphers -p 443 <host>
```

You may also use tools like sslyze and testssl.sh

Control

If communication over HTTP is allowed, the control is failed. If this requirement is failed, all L1 controls are failed in this section.

Resources

https://github.com/nabla-c0d3/sslyze
https://portswigger.net/bappstore/474b3c575a1a4584aa44dfefc70f269d
https://github.com/drwetter/testssl.sh

https://t.me/learningnets

9.1.2 Strong algorithms

Verify using online or up to date TLS testing tools that only strong algorithms, ciphers, and protocols are enabled, with the strongest algorithms and ciphers set as preferred.

Explanation

SSLabs has written a TLS deployment best practice document which comes with a list of recommended ciphersuites.

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

Here is the list from the document:

```
ECDHE-ECDSA-AES128-GCM-SHA256
ECDHE-ECDSA-AES256-GCM-SHA384
ECDHE-ECDSA-AES128-SHA
ECDHE-ECDSA-AES256-SHA
ECDHE-ECDSA-AES128-SHA256
ECDHE-ECDSA-AES256-SHA384
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-RSA-AES256-GCM-SHA384
ECDHE-RSA-AES128-SHA
ECDHE-RSA-AES256-SHA
ECDHE-RSA-AES128-SHA256
ECDHE-RSA-AES256-SHA384
DHE-RSA-AES128-GCM-SHA256
DHE-RSA-AES256-GCM-SHA384
DHE-RSA-AES128-SHA
DHE-RSA-AES256-SHA
DHE-RSA-AES128-SHA256
DHE-RSA-AES256-SHA256
```

As a rule of thumb only ciphersuites that utilize ECDHE and DHE are recommended, which provide forward secrecy.

Testing methods

Bash script

I have written a small tool in bash that utilizes openssl for testing, which tests against the three L1 controls in Network Security category of ASVS. To execute the script, run:

```
bash test.sh <domain to scan>
```

It prints out which controls are failed and which are successful. The script compares the connected ciphersuites to the ones recommended by SSLabs in their best practices document.

Nmap

```
nmap -sV --script ssl-enum-ciphers -p 443 <host>
```

https://t.me/learningnets

You may also use tools like sslyze and testssl.sh

Control

If the supported ciphersuites are different the recommended ones, the control is failed.

Resources

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

https://github.com/nabla-c0d3/sslyze

https://portswigger.net/bappstore/474b3c575a1a4584aa44dfefc70f269d

https://github.com/drwetter/testssl.sh

https://t.me/learningnets

9.1.3 TLS versions

Verify that old versions of SSL and TLS protocols, algorithms, ciphers, and configuration are disabled, such as SSLv2, SSLv3, or TLS 1.0 and TLS 1.1. The latest version of TLS should be the preferred cipher suite.

Explanation

As of January 2020 Google Chrome marks sites using TLS1.0 and TLS1.1 certificates for encryption as insecure and in 2021 will not load websites that try to connect over TLS1.0 or TLS1.1. All protocol versions below TLS1.2 have known vulnerabilities or are otherwise insecure. Therefore, a website has to use only secure protocols, such as TLS1.2 and TLS1.3.

Testing methods

Bash script

I have written a small tool in bash that utilizes openssl for testing, which tests against the three L1 controls in Network Security category of ASVS. To execute the script, run:

```
bash test.sh <domain to scan>
```

It prints out which controls are failed, and which are successful.

Curl

To test if which TLS versions does the server support run:

```
curl -2 -is https://example.com
```

- `-2` uses SSL version 2 for connection
- Test other TLS versions by replacing `-2` option with other options: `-3` (for SSLv3), `--tlsv1.0`, `--tlsv.1.1`, `--tlsv1.2`, and `--tlsv1.3`

The response for attempt to connect over SSLv2, SSLv3, TLSv1.0 and TLSv1.1 should not be allowed to connect and should be instead redirected to HTTPS over TLSv1.2 or TLSv1.3 with HTTP/1.1 301 Moved Permanently or similar status code.

Nmap

```
nmap -sV --script ssl-enum-ciphers -p 443 <host>
```

You may also use tools like sslyze and testssl.sh

Control

If communication is allowed using TLS versions lower than TLSv1.2 the control is failed.

Resources

https://github.com/nabla-c0d3/sslyze
https://portswigger.net/bappstore/474b3c575a1a4584aa44dfefc70f269d
https://github.com/drwetter/testssl.sh

https://t.me/learningnets

14. Configuration

14.2.1 Up to date components

14.2.2 Excess configuration removed

14.2.3 Subresource Integrity

14.3.1 Error messages

14.3.2 Debug modes

14.3.3 Version information of system components

14.4.1 Charset

14.4.2 Content-Disposition: attachment

14.4.3 Content Security Policy

14.4.4 X-Content-Type-Options: nosniff

14.4.5 HTTP Strict Transport Security

14.4.6 Referrer-Policy

14.4.7 X-Frame-Options or CSP: frame-ancestors

14.5.1 HTTP methods

14.5.2 Origin header

14.5.3 CORS header

https://t.me/learningnets

14.2.1 Up to date components

> Verify that all components are up to date, preferably using a dependency checker during build or compile time.

CWE 1026

Explanation

Using older versions of software packages, for example jQuery, may allow for exploitation of XSS on a website. This problem is also described in OWASP Top 10 "A9: Using Components with Known Vulnerabilities". The problem of using known vulnerable components is very spread, but essential to fix.

Testing methods

There are several tools that can be used for checking dependencies such as OWASP dependency-check, Retire.js and many others. Consult OWASP Cheat Sheet Vulnerable Dependency Management for more recommendations and best practices. If the assessment is run without access to source code, only f.ex. on a website, you may use Retire.js. Retire.js has an extension to both Burp (only in Pro version) and ZAP, which will automatically detect issues.

If the assessment is done with access to source code, you may use OWASP dependency-checker. The tool contains several file type analyzers, which can be found under https://jeremylong.github.io/DependencyCheck/analyzers/index.html. It can be run as a command line tool or added to your CI/CD pipeline.

Proxy - RetireJS

Click the icon "Mange Add-ons" > choose tab "Marketplace" > filter for: "Retire.js" > check the tick in Retire.js entry > Click "Install selected". The plugin should work right away unless otherwise stated by the application. Browse to the site in scope. Preferably use a spider or and AJAX spider. The add-on will raise alerts with title "Vulnerable JS library" if there are any components that have known vulnerabilities.

If you would prefer to see the alerts as this ASVS control, you may use a standalone script "14-2-1 Up to date components.js". The script makes cosmetic changes to the alerts, in case the user would like to make a report which would only include ASVS controls - it changes the title to "14.2.1 Verify that all components are up to date, preferably using a dependency checker during build or compile time." and adds a description. Remember to run it only once - if you run it more than once, the same title and description will be appended again to the alert.

Control

If any alerts were raised by RetireJS or any other dependency check tool, the control is failed.

Resources

https://techbeacon.com/app-dev-testing/13-tools-checking-security-risk-open-source-dependencies

https://t.me/learningnets

14.2.2 Excess configuration removed

> Verify that all unneeded features, documentation, samples, configurations are removed, such as sample applications, platform documentation, and default or example users.

CWE 1002

Explanation

Default configurations and code may contain users, credentials or open ports and overall lower security hardening, which is typically used for development.

Testing methods

Based on information gained about the backend systems from other controls such as 14.2.1, 14.3.1 and 14.3.3 find default configuration for the component in its documentation. Afterwards check whether it is present in the application.

Proxy

If you have run other scripts, ZAP has possibly raised alerts on controls which show information disclosure (such as 14.2.1, 14.3.1 and 14.3.3 but also others). You can start by assessing the highlighted responses and then by using the Search tab.

Control

This control is very dependent on the codebase and frameworks used. While having some default configuration is not bad by itself, ensure there are no default or example users, ports or unneeded code or documentation left. If any such information is found, the control is failed.

Resources

https://t.me/learningnets

14.2.3 Subresource Integrity

> Verify that if application assets, such as JavaScript libraries, CSS stylesheets or web fonts, are hosted externally on a content delivery network (CDN) or external provider, Subresource Integrity (SRI) is used to validate the integrity of the asset.

CWE 714

Explanation

Resources loaded from external providers should be checked against their hashes before they are loaded using subresource integrity. Any `<script>` and `<link>` elements should contain integrity attribute with a hash of the file and crossorigin attribute.

SRI is used to check that a file that is loaded on a website has not been tampered with. SRI should be implemented on files loaded from a content delivery network. As an example - if an attacker controls a CDN, they may inject arbitrary content to the files that the target website is fetching. SRI will prevent such files from loading, since its hash would change.

The crossorigin attribute is used for making cross origin requests and loading Javascript files from another domain requires this attribute for SRI to function properly.

Testing methods

ZAP script

Enable script “14-2-3 Subresource Integrity” and browse to the site in scope. Preferably use a spider or and AJAX spider. The script searches for script and link elements and checks if integrity attribute is present for scripts and CSS loaded externally. If not present, it will raise an alert.

Manually

SRI can be found in HTML. Using DevTools, inspect the html for all `<script>` and `<link rel="stylesheet">` tags. Then look into Sources tab to see where are scripts and CSS files loaded from - knowing domains where scripts are loaded from will make it easier to find all loaded scripts in the HTML. If you see any loaded from places other than the target domain itself, see the names of the scripts. Back to Inspect tab and look in between

tags if the files are loaded against their hash.

Example of properly configured SRI:

```
// Loading Javascript scripts
<script src="[https://example.com/example-
framework.js](https://example.com/example-framework.js)"
integrity="sha384-
oqVuAfXRKap7fdgcCY5uykM6+R9GqQ8K/uxy9rx7HNQlGYl1kPzQho1wx4JwY8wC"
crossorigin="anonymous"></script>

//Loading CSS
```

https://t.me/learningnets

```
<link rel="stylesheet" href="correct_hash.css"
integrity="sha256-qvuZLpjL9TNV6yI1kNdGCPnSTrWM6Y0ILEzzyvA9hGY=">
```

If the hash is wrong or there is no crossorigin attribute, there will appear an error in the Console in Developer Tools. If not, the files will be loaded under Network tab with no error.

Control

If there are files hosted externally e.g. on a CDN and they do not use SRI, the control is failed.

Resources

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

https://shubhamjain.co/til/subresource-integrity-crossorigin/

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

https://t.me/learningnets

14.3.1 Error messages

Verify that web or application server and framework error messages are configured to deliver user actionable, customized responses to eliminate any unintended security disclosures.

CWE 209

Explanation

Default error pages usually disclose information about the underlying software and web server in use. Optimally instead of default error pages, there should be a custom, user-actionable error pages returned.

Testing methods

There is no one full proof way to test that all errors are user actionable and contain no security disclosures - most often they come up after a longer reconnaissance process of a web application during a web app pentest or a scan. There are though several tests that may help triggering several types of errors, e.g. by changing HTTP methods, headers and body of the message to trigger one of the status codes. Some are presented in the OWASP WSTG chapter 8.1.

There are many tools which can be used for testing: telnet, curl, a proxy and many other. It is best to trigger errors that would give you one of the 4xx and 5xx http code responses.

Examine the response and see if there are any information about the services running or the server version.

Proxy

1. Browse to the site in scope using the built-in browser.
2. Enable script "14-3-1 Error messages.py" which can be found under Scripts > Active Rules.
3. Go to Sites and select the site you wish to scan. Use a spider if you wish. Then right click the site (node) and choose Attack > Active Scan. Choose "Scripts only" policy (if you have not created a policy for active scan rules, follow the instruction under "Getting Started") and tick recurse if you wish to execute the script on each subnode - in this case there is no need to tick it off, but if you wish to have more coverage you may tick it off. Caution: this will produce much more traffic, which in turn will generate errors and will give you many more alerts.
4. Click Start Scan.
5. The script attempts to trigger errors in a web application and checks the response for any software component disclosure - if it finds one, it raises an alert. Most of the attacks are based on OWASP WSTG v4.1 chapter 8.1 Testing for Error Code.
6. Click the "Reponse" tab and go through each response - it may happen that the script didn't detect a security disclosure. If this is the case feel free to report it on this project's Github.

https://t.me/learningnets

Control

If errors do not disclose any information about the server and its services, the control is successful.

Resources

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

https://owasp.org/www-project-proactive-controls/v3/en/c10-errors-exceptions

https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/01-Information_Gathering/02-Fingerprint_Web_Server.html

https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/08-Testing_for_Error_Handling/01-Testing_for_Error_Code.html

https://t.me/learningnets

14.3.2 Debug modes

> Verify that web or application server and application framework debug modes are disabled in production to eliminate debug features, developer consoles, and unintended security disclosures.

CWE 497

Explanation

Debug mode often allows for much more functionality and employs poorer security practices to make debugging easier. Therefore it should not be present in production.

Testing methods

Debug code may not be visible right away.

Proxy

Enable script “14-3-2 Debug modes.py” and browse to the site in scope. Preferably use a spider or and AJAX spider. The script searches the body of the response for words such as ‘debug’, ‘medio’ and raises an alert if it did find them.

Feel free to add to the script other words that software developers might have used for debugging. You may also use the Search tab to look for words which mean “debug” or similar in the language of the website.

Control

While it is preferable that no debug code is found, the control is not failed just because debug code was found. The point is that access to it should be disabled. For that, a short code review would need to be undertaken.

Conditions for success of this control are more flexible, but it is preferable that there is no debug code in production. If some is found, consider deleting it.

Resources

https://blog.pentesteracademy.com/hacking-jwt-tokens-debug-mode-in-production-4f8119b9b755

https://t.me/learningnets

14.3.3 Version information of system components

Verify that the HTTP headers or any part of the HTTP response do not expose detailed version information of system components.

CWE 200

Explanation

Giving away too much information in HTTP responses allows attackers to make more targeted attacks. Information may be exposed in Server header or based on the ordering of the header fields. Some software adds also it's own headers, which may expose what is running on the server.

Testing methods

This control can be tested in DevTools, via a proxy or by viewing response returned by nmap script http-headers. A request can be also sent using curl or openssl or using any other banner grabbing tool.

Proxy

Enable script "14-3-3 Version information of system components.py" and browse to the site in scope. Preferably use a spider or and AJAX spider. The script will raise an alert if 'Server' or X-Powered-By' header is present. It may happen that detailed information about the system component is obscured - review the findings of the script before assessing the control.

Nmap

Nmap is a scanner which has built in scripts. One of them returns HTTP headers from a website, like Server and X-Powered-By headers:

nmap -sV –script=http-headers <target.com>

Control

Observe if the Server header with server version is in the response or if it is possible to tell what the server version is from other headers. If yes, control is failed.

Resources

https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/01-Information_Gathering/02-Fingerprint_Web_Server.html

Disabling server header in IIS https://support.qlik.com/articles/000063710

https://t.me/learningnets

14.4.1 Charset

> Verify that every HTTP response contains a Content-Type header. text/*, /+xml and application/xml content types should also specify a safe character set (e.g., UTF-8, ISO-8859-1).

CWE 173

Explanation

The header Content-Type denotes what the content is encoded in. By declaring Content-Type in a response it is possible to hinder XSS attacks leveraging different encodings than the server expects.

Not setting a charset in the Content-Type header can cause the browser to attempt to sniff the charset. An attacker can send a payload by encoding special characters in f. ex. UTF-7 and bypass application's defensive measures. This is not an issue in modern browsers support, since UTF-7 is not supported anymore, but nevertheless it is a best practice not to allow the browser to sniff the charset by setting it in Content-Type header.

Testing methods

This control can be tested via a proxy. Observing the traffic in DevTools or sending single requests using curl or other banner grabbing tools is not recommended, since all resources should use a 'Content-Type' header and additionally the types specified above should contain a charset.

Proxy

Enable script "14-4-1 Charset.py" and browse to the site in scope. Preferably use a spider or and AJAX spider. The script will raise an alert if 'Content-Type' header is present and if the header specifies a safe charset for text/, /*+xml and application/xml content types.

Control

If there is an HTTP response not containing Content-Type header or Content-Type header with types text/*, /+xml and application/xml do not have a declared charset, the control is failed.

Resources

https://portswigger.net/kb/issues/00800200_html-does-not-specify-charset

https://www.leviathansecurity.com/white-papers/flirting-with-mime-types

https://www.w3.org/International/questions/qa-html-encoding-declarations

https://www.w3.org/Protocols/rfc1341/4_Content-Type.html

https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types

https://www.w3.org/International/questions/qa-css-charset

https://github.com/OWASP/ASVS/issues/710

https://t.me/learningnets

14.4.2 Content-Disposition: attachment

Verify that all API responses contain Content-Disposition: attachment; filename="api.json" (or other appropriate filename for the content type).

CWE 116

Explanation

Content-disposition header in an API response is used to automatically give a name to a file downloaded by a user. Adding the header to API responses helps protect against misunderstanding of the MIME type between client and server. The "filename" option helps protect against Reflected File Download attacks.

Testing methods

Manually

Find and try to fetch some resources from the API using DevTools, curl, Postman or other tool. Observe the response for Content Disposition header.

Proxy

Enable the script "14-4-2 Content-Disposition: attachment.py" found under Scripts > Passive Rules. Using the built-in browser, browse directly to the API and fetch one or more of the resources. If the response does not contain Content-Disposition header, the script will raise an alert.

You may also look in the proxy at the response yourself.

Control

Observe if the header Content-Disposition with a proper name is in the response. If yes, control is successful.

Resources

https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/reflected-file-download-a-new-web-attack-vector/

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Disposition

https://t.me/learningnets

14.4.3 Content Security Policy

> Verify that a content security policy (CSPv2) is in place that helps mitigate impact for XSS attacks like HTML, DOM, JSON, and JavaScript injection vulnerabilities.

CWE 1021

Explanation

Content-Security-Policy is an HTTP header that helps detect, mitigate and report on many kinds of data injection attacks including XSS and clickjacking. CSP restricts how resources such as JavaScript, CSS and many more are loaded by the browser. CSP has many directives and how CSP looks like depends solely on what resources the site loads. A proper CSP policy will not be too long or too complex.

As an example - a site target.com might load a javascript analytics file from analytics.com, images from images.com and CSS from fonts.com. In this case a good CSP would look like this:

content-security-policy: default-src 'self'; img-src 'self' images.com; script-src 'self' analytics.com; style-src 'self' fonts.com; report-uri /some-report-uri;

Using report-uri or report-to directive is a good practice, since it instructs the browser to send reports of policy failures to the specified URI. The above CSP is very simple, but it was made only to show the principle behind it.

There also are versions of the header called X-Content-Security-Policy and X-Webkit-CSP, but they create unexpected behaviour and should be avoided.

Testing methods

You may test it in a proxy or simply in Developer Tools of your favourite browser. Any request to the domain should receive in response the configured CSP.

It is hard to specify what a proper configuration looks like, but as a rule of a thumb it should not allow too many domains, shouldn't use wildcards '*' and shouldn't use 'unsafe-eval' and 'unsafe-inline'. If any scripts are loaded via inline script they should contain a nonce value or hash value.

Proxy

Enable script "14-4-3 Content Security Policy.py" and browse to the site in scope. Preferably use a spider or and AJAX spider. The script will raise an alert if Content-Security-Policy header is not present or if the CSP is configured with the directives: 'unsafe-inline', 'unsafe-eval' and wildcards.

Online CSP evaluator

You may also use an online CSP evaluator which will give you recommendation based on how the CSP looks like, such as:

https://csp-evaluator.withgoogle.com/

https://t.me/learningnets

Control

If HTTP requests do not contain the header Content-Security-Policy, which is properly configured, the control is failed.

Resources

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

https://t.me/learningnets

14.4.4 X-Content-Type-Options: nosniff

Verify that all responses contain X-Content-Type-Options: nosniff.

CWE 116

Explanation

X-Content-Type-Options: nosniff is a protection against MIME sniffing vulnerabilities. MIME sniffing is a technique which determines an asset's file format, when no content type is provided. While MIME sniffing was created as a feature to detect when f.ex. a Javascript file was sent as Content-type: text/plain which would create a MIME type mismatch. Since the file is examined by the browser, it can cause Javascript to be rendered and opens a possibility for XSS attacks.

Developers might incorrectly set a value for Content-Type header for a response's content - for example a server may send Content-Type: text/plain for a Javascript resource. Browsers may render such resourcesm so the website operates as intended, even though it is a mismatch.

The header prevents that from happening.

Testing methods

This control can be tested in DevTools, via a proxy or by viewing response returned by nmap script http-headers. A request can be also sent using curl or using any other banner grabbing tool.

Proxy

Enable script "14-4-4 X-Content-Type-Options.py" and browse to the site in scope. Preferably use a spider or and AJAX spider. The script will raise an alert if X-Content-Type-Options header is not present.

Nmap

Nmap is a scanner which has built in scripts. One of them returns HTTP headers from a website, like X-Content-Type-Options headers:

nmap -sV –script=http-headers <target.com>

Control

If HTTP requests do not contain the header X-Content-Type-Options: nosniff, the control is failed.

Resources

https://www.denimgroup.com/resources/blog/2019/05/mime-sniffing-in-browsers-and-the-security-implications/

https://t.me/learningnets

14.4.5 HTTP Strict Transport Security

Verify that HTTP Strict Transport Security headers are included on all responses and for all subdomains, such as Strict-Transport-Security: max-age=15724800; includeSubdomains.

CWE 523

Explanation

HTTP Strict-Transport-Security HSTS header is used to tell a browser that a website should only be accessed using HTTPS. It has two directives, which should both be included in a configuration:

- max-age=, tells the browser to enable HSTS for a domain and remember it for a given number of seconds. Best practices specify `max-age=31536000` which is a year.
- includeSubDomains - HSTS covers all subdomains of a domain.

There is one more directive, preload, but it is not required to use.

Testing methods

This control can be tested in DevTools, via a proxy or by viewing response returned by nmap script http-headers. A request can be also sent using curl or using any other banner grabbing tool.

Proxy

Enable script "14-4-5 HTTP Strict Transport Security.py" and browse to the site in scope. Preferably use a spider or and AJAX spider. The script will raise an alert if 'Strict-Transport-Security' header is not present. Additionally, the script test if the HSTS header is configured with the directives: max-age=15724800; includeSubdomains. Max age should be at least 15724800, but a longer time is preferred.

Nmap

nmap -sV –script=http-headers <target.com>

Control

If HTTP requests do not contain the header Strict-Transport-Security: max-age=31536000; includeSubDomains; the control is failed.

Resources

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

https://t.me/learningnets

14.4.6 Referrer-Policy

Verify that a suitable “Referrer-Policy” header is included, such as “no-referrer” or “same-origin”.

CWE 116

Explanation

HTTP requests may include Referrer header, which indicates origin or web page URL request was sent from. Referrer-Policy specifies how much information is sent in the Referer header. Sending full URL in Referer header may expose a lot of sensitive information.

Referrer-Policy has many directives, but the control focuses on two:

- no-referrer, which means that no Referer header is sent, and
- same-origin, which only shows the origin of the URL, f.ex https://example.com/ even if the URL the request was made from was https://example.com/login or similar

Referrer-Policy should be used on sites that f.ex have a social media link in the footer or an image that is hosted on a third party but embedded in your page.

There are several ways Referrer-Policy can be implemented, such as:

- Via the Referrer-Policy HTTP header.
- Via a meta element with a name of referrer.
- Via a referrerpolicy content attribute on an a, area, img, iframe, or link element.
- Via the noreferrer link relation on an a, area, or link element.
- Implicitly, via inheritance.

Taken from W3C Candidate Recommendation https://www.w3.org/TR/referrer-policy/#referrer-policy-delivery-meta

It is also possible to implement it via Content-Security-Policy - see resources.

Although there are many ways to implement the policy, this control only tests against the header, since it it the most often used way of implementation.

Testing methods

This control can be tested in DevTools, via a proxy or by viewing response returned by nmap script http-headers. A request can be also sent using curl or using any other banner grabbing tool.

Proxy

Enable script “14-4-6 Referrer-Policy.py” and browse to the site in scope. Preferably use a spider or and AJAX spider. The script will raise an alert if ‘Referrer-Policy’ header is not present. Additionally, the script tests if the Referrer-Policy header is configured with either of the directives: ‘no-referrer’ or ‘same-origin’.

https://t.me/learningnets

Nmap

nmap -sV –script=http-headers <target.com>

Control

If other directives than “no-referrer” or “same-origin” are specified, the control is failed.

Resources

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

https://scotthelme.co.uk/csp-cheat-sheet/#referrer

https://t.me/learningnets

14.4.7 X-Frame-Options or CSP: frame-ancestors

Verify that a suitable X-Frame-Options or Content-Security-Policy: frame-ancestors header is in use for sites where content should not be embedded in a third-party site.

CWE 346

Explanation

X-Frame-Options header tells your browser how to behave when handling your site's content. It protects against clickjacking by not allowing rendering of a page in a frame. This can include rendering of a page in a `<frame>`,`<iframe>`, or `<object>`.

Iframes are used to embed and isolate third party content into a website. Examples of things that use iframes might include social media sharing buttons, Google Maps, video players, audio players, third party advertising, and even some OAuth implementations.

X-Frame-Options has three directives: deny, sameorigin and allow-from *uri*.

- The deny directive completely disables the loading of the page in a frame, regardless of what site is trying.
- The sameorigin directive allows the page to be loaded in a frame on the same origin (domain, http/https scheme and port) as the page itself. It is up to a browser vendor whether this applies to all of the frame's ancestors.
- The allow-from *uri* directive allows the page to be loaded in an iframe on the specified domain or origin, but it is obsolete. There are browsers that do not support this directive and if a user uses one of those browsers, they will not be protected with X-Frame-Options from clickjacking.

Using Content-Security-Policy: frame-ancestors is very similar to X-Frame-Options and this control specifies that either of them can be used, as long as it is properly configured.

CSP: frame ancestors directive can be set to several values:

- 'none' which is the same as X-Frame-Options: deny
- 'self' which refers to the origin from which the protected document is being served, including the same URL scheme and port number.

  which can be defined in several ways. It's fairly simple to configure but in case of doubts visit

Testing methods

This control can be tested in DevTools, via a proxy or by viewing response returned by nmap script http-headers. A request can be also sent using curl or using any other banner grabbing tool.

Proxy

Enable script "14-4-7 X-Frame-Options or CSP frame-ancestors.py" and browse to the site in scope. Preferably use a spider or and AJAX spider. The script will raise an alert if 'X-Frame-

https://t.me/learningnets

Options' or Content-Security-Policy header with directive frame-ancestors is not present. Additionally, the script tests if the CSP header uses a wildcard, which renders the header useless.

Nmap

nmap -sV –script=http-headers <target.com>

Control

If any of:

- X-Frame-Options: deny or
- X-Frame-Options: sameorigin or
- Content-Security-Policy: frame-ancestors 'none'
- Content-Security-Policy: frame-ancestors

is present in server response, the control is successful.

Resources

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/frame-ancestors

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

https://t.me/learningnets

14.5.1 HTTP methods

Verify that the application server only accepts the HTTP methods in use by the application or API, including pre-flight OPTIONS.

CWE 749

Explanation

OWASP testing guide has a very good explanation of the problem:

https://github.com/OWASP/wstg/blob/master/document/4-Web_Application_Security_Testing/02-Configuration_and_Deployment_Management_Testing/06-Test_HTTP_Methods.md

HTTP defines a set of request methods to indicate the desired action to be performed for a given resource.

`GET`

The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.

`HEAD`

The HEAD method asks for a response identical to that of a GET request, but without the response body.

`POST`

The POST method is used to submit an entity to the specified resource, often causing a change in state or side effects on the server.

`PUT`

The PUT method is used to store the enclosed entity on a server - replaces all current representations of the target resource with the request payload.

`DELETE`

The DELETE method deletes the specified resource.

`CONNECT`

The CONNECT method establishes a tunnel to the server identified by the target resource.

`OPTIONS`

The OPTIONS method is used to describe the communication options for the target resource - to request available methods on a server - is employed to return the request that was received by the final recipient from the client so that it can diagnose the communication.

`TRACE`

The TRACE method performs a message loop-back test along the path to the target resource.

https://t.me/learningnets

`PATCH`

The PATCH method is used to apply partial modifications to a resource.

There are also extended HTTP methods such as web-based distribution authoring and versioning (WEBDAV). WEBDAV can be used by clients to publish web contents and involves HTTP methods such as PROPFIND, MOVE, COPY, LOCK, UNLOCK, and MKCOL.

While GET and POST methods are used in most attacks, they themselves are not a problem and are required for a common plain server. PUT, DELETE, CONNECT and TRACE are not needed for functioning of a server and many have known exploits for them.

Pre-flight OPTIONS

A CORS preflight request is a CORS request that checks to see if the CORS protocol is understood and a server can use specific methods and headers.

CORS allows one site to make a request to another.

A preflight request asks for the server's permission to send the request - it isn't the request itself. Instead, it contains metadata about it, such as which HTTP method is used and if the client added additional request headers. The server inspects this metadata to decide whether the browser is allowed to send the request.

For example, a client might be asking a server if it would allow a DELETE request, before sending a DELETE request, by using a preflight request:

```
OPTIONS /resource/foo
Access-Control-Request-Method: DELETE
Access-Control-Request-Headers: origin, x-requested-with
Origin: [https://](https://foo.bar.org/)example.com
```

If the server allows it, then it will respond to the preflight request with an Access-Control-Allow-Methods response header, which lists DELETE:

```
HTTP/1.1 204 No Content
Connection: keep-alive
Access-Control-Allow-Origin: https://example.com
Access-Control-Allow-Methods: POST, GET, OPTIONS, DELETE
Access-Control-Max-Age: 86400
```

One other set of Methods needs mentioning: ALL OTHERS. For some webservers, in order to enable/disable/restrict certain HTTP Methods, you explicitly set them one way or another in the configuration file. However, if no default is set, it can be possible to "inject" additional methods, bypassing certain access controls that the web server may have implemented (poorly). See for example some more info on OWASP.

https://t.me/learningnets

Testing methods

Manually

You can test HTTP methods by sending single requests using telnet, openssl, netcat or other tools. You may also capture traffic using a proxy and then change the requests to match your needs.

Using telnet:

```
$ telnet <target> 80
```

Then specify the method, the resource and HTTP version.

```
GET / HTTP/1.1
Host: www.target.com
```

Then hit enter twice to send and to receive the messages.

Nmap

You can run an nmap script to find supported methods:

nmap –script=http-methods.nse –script-args http-methods.retest=1

Proxy

1. Browse to the site in scope using the built-in browser.
2. Enable script "14-5-1 HTTP methods.py" which can be found under Scripts > Active Rules.
3. Go to Sites and select the site you wish to scan. Use a spider if you wish. Then right click the site (node) and choose Attack > Active Scan. Choose "Scripts only" policy (if you have not created a policy for active scan rules, follow the instruction under "Getting Started") and tick recurse if you wish, but there is no requirement for it.
4. Click Start Scan.
5. The script sends several HTTP methods to the target domain. If the answer is 2xx, the method may be allowed. Review the alerts.

Control

Find all supported methods. If there are any others than GET, HEAD, POST and OPTIONS, the control is failed.

Resources

https://security.stackexchange.com/questions/21413/how-to-exploit-http-methods

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

https://www.sans.org/reading-room/whitepapers/testing/penetration-testing-web-application-dangerous-http-methods-33945

https://t.me/learningnets

14.5.2 Origin header

Verify that the supplied Origin header is not used for authentication or access control decisions, as the Origin header can easily be changed by an attacker.

CWE 346

Explanation

While it is not a common behavior, it may happen that a developer restricted access to some subpages or resources only to people that would use a specific Origin header in a request. It's a way to enforce access control, though not a very good one, and should be avoided. As the control specifies, it is very easy to change the Origin header.

Testing methods

There is no straight forward way to test it in a black box scenario. If during testing other controls you have encountered an error with a status code 403 forbidden, you may try to add an Origin header with the domain name you are testing or one of the domains belonging to 3rd party providers of the code that is loaded on the tested domain. Triggering 403 may also happen if you spider an application using OWASP ZAP or Burp Suite.

Let's say during testing you have sent a request as below:

```
GET /secret.txt HTTP/1.1
Host: target.com
```

And received a response:

```
HTTP/1.1 403 Forbidden
```

Then you can add an Origin header with the domain name (3rd party providers' domains) to the same request:

```
GET /secret.txt HTTP/1.1
Host: target.com
Origin: target.com
```

If it succeeds in fetching the resource, it means the Origin header was used for access control:

```
HTTP/1.1 200 OK
```

Control

If the Origin header is used for any authentication or access control decisions, the control is failed.

Resources

https://owasp.org/www-community/attacks/CORS_OriginHeaderScrutiny

https://t.me/learningnets

14.5.3 CORS header

Verify that the cross-domain resource sharing (CORS) Access-Control-Allow-Origin header uses a strict white-list of trusted domains to match against and does not support the “null” origin.

CWE 346

Explanation

Cross-Origin Resource Sharing (CORS) is a mechanism that uses additional HTTP headers to tell browsers to give a web application running at one origin, access to selected resources from a different origin. A web application executes a cross-origin HTTP request when it requests a resource that has a different origin (domain, protocol, or port) from its own.

The Origin request header indicates where a fetch originates from. It is set automatically by the user agent to describe the security contexts that caused the user agent to initiate an HTTP request. It is sent with CORS requests, as well as with POST requests. It is similar to the Referer header, but, unlike this header, it doesn’t disclose the whole path.

The origin header is always sent by the browser in a CORS request and indicates the origin of the request. To avoid using the wildcard or maintaining a white-list of websites, some implementations copy the Origin header from the request back in the response - the website ‘reflects’ it. Combined with Access-Control-Allow-Credentials: true, an attacker can trick a user visiting his website into doing Cross-Origin request and read the content of the response.

This control focuses on whitelisting of domains. A site requests a resource from another site, which will be called ‘target’ in this example. Target then checks if the requesting site is one of its whitelisted sites. If it is, the target sends a response with a header Access-Control-Allow-Origin: , and if not, the file is not loaded in the browser. Properly configured CORS is a good mechanism for preventing CSRF vulnerabilities.

A typical CORS request looks like this:

```
GET /resource HTTP/1.1
Host: target.com
Origin: https://hacker.com
Cookie: sessionid=...
```

And the response:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://malicious-website.com
Access-Control-Allow-Credentials: true
...
```

Testing methods

While we may not know what the whitelisted domains are, it doesn’t matter for the sake of this control.

https://t.me/learningnets

Access-Control-Allow-Origin can have values, , , *null. The* and null should not be used. If CORS is not configured or has a strict whitelist of trusted domains, and we will try to send a request with random value in the origin, the response will not contain Access-Control-Allow-Origin. null is a special value for the Origin header. The specification mentions it being triggered by redirects, and local HTML files. Some applications might whitelist the null origin to support local development of the application, but it is not a good practice.

Proxy

1. Browse to the site in scope using the built-in browser.
2. Enable script "14-5-3 CORS header.py" which can be found under Scripts > Active Rules.
3. Go to Sites and select the site you wish to scan. Use a spider if you wish. Then right click the site (node) and choose Attack > Active Scan. Choose "Scripts only" policy (if you have not created a policy for active scan rules, follow the instruction under "Getting Started") and tick recurse.
4. Click Start Scan.
5. The script sends a CORS request from an inexisitent domain to determine if the Access-Control-Allow-Origin header is configured with a wildcard, if it reflecting the Origin header or if the request was blocked.

Netcat

We are going to send the request over netcat and based on the response we will see if CORS policy is misconfigured.

```
echo -ne "HEAD / HTTP/1.1\r\nHost: example.com\r\nOrigin:
randomvalue.test\r\n\r\n" | nc example.com 80
```

In the response you should see Access-Control-Allow-Origin header. If CORS is not configured or uses a strict whitelist, you will not see the header. Although the HTTP response was 200 and the resource would be fetched, it would not be loaded on a website.

If the randomvalue.test is reflected in the response in the "Access-Control-Allow-Origin" header, at this point it means that any domain can access resources from this domain. If the response contains any sensitive information, it may be possible to retrieve them with a script from a website.

It may also happen that you recieve a response such as 301 or 302. Then you need to adjust the domain address accordingly.

Curl

Unless you already know of specific resources on the website that are loaded using CORS, it is better to run the ZAP proxy script, but you may also use curl.

```
curl -H "Origin: https://randomvalue.test" --verbose https://target.com
or
curl -I -X OPTIONS -H "Origin: exampletestsite.com" -H "Access-Control-Request-
Method: GET" https://www.webscantest.com/cors/cors.php
```

If there is a problem with certificate use option -k.

https://t.me/learningnets

Javascript (only tests null Origin)

Nick (njgibbon) has created a tiny CORS test on his github for testing CORS requests with null Origin, which can be accessed at

https://github.com/njgibbon/nicks-cors-test.

It consists of a Javascript file which makes a request to a target using AJAX without any origin specified. The author specifies that first you should change the dataType (in our case to html) and url to your target. Save it and load the files in the browser. Then in DevTools look at Console and see if there was raised an error. If there was, the app does not allow requests with null Origin. If there wasn't, the app DOES allow requests with null Origin and that means that the control is failed.

Control

If you see a response with header Access-Control-Allow-Origin with directives that do not have a strict whitelist (uses f.ex. * or null), the control is failed.

If you see a response with header Access-Control-Allow-Origin with reflected Origin, and the value is always reflected even if the domain does not exist, the control is failed.

Resources

https://portswigger.net/web-security/cors

https://www.packetlabs.net/cross-origin-resource-sharing-cors/

https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/11-Client_Side_Testing/07-Testing_Cross_Origin_Resource_Sharing.html

https://www.sjoerdlangkemper.nl/2018/09/12/authorization-header-and-cors/

http://demo.sjoerdlangkemper.nl/auth/fetch.html

https://t.me/learningnets