OWASP top 10 security risks discussion

My Discussion Choice: Cross-site Scripting

Initial post

The Open Web Application Security Project works to improve the security of software. WASP provide tools, training and education resources for developers to use on their own projects.

OWASP has curated a list of Top Ten security risks that developers should have awareness of on their projects.

One of the risks OWASP raise awareness of is Cross-Site Scripting (XSS).

Cross-Site Scripting is a type of injecting attack where a script is injected into the user's browser and executes malicious code.

A simple example of Cross-Site scripting called Reflected cross-site scripting occurs when a website simply sends a script embedded in the websites HTML which executes in the user's browser without them knowing. Scripts like this could steal information from the user's cookies among other things.

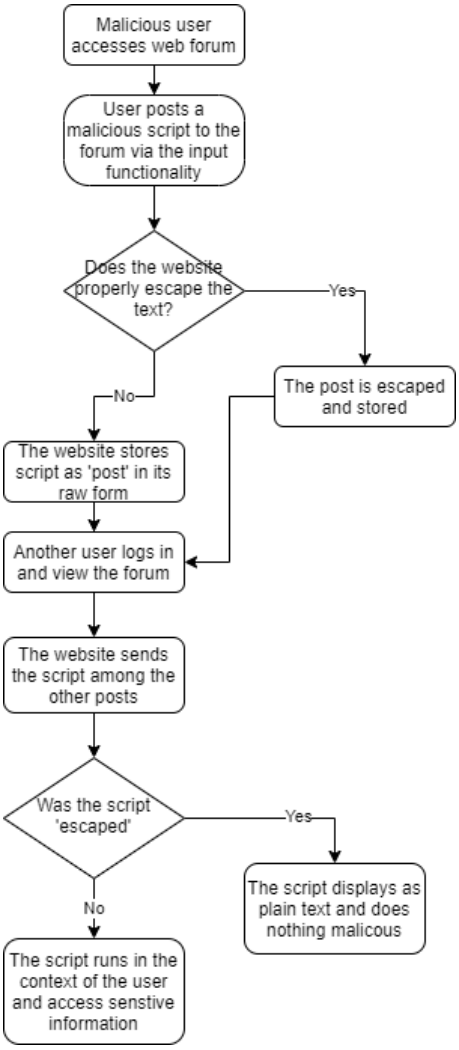
Stored cross-site scripting is a little more advanced than Reflected cross-site scripting, where websites unknowingly propagate malicious scripts, users have to submit to the website. An example of RXXS would be if a user submitted a script as a forum post, then the website would send this script to other people who wanted to view the forum.

All forms of cross-site scripting allow the malicious code to execute in the context of the user, this way the code has access to user-specific data and controls.

To prevent XSS the developers must make sure to escape users' input, developers could use tools such as ReactJS that automatically escape input to ensure this is done.

References:

*OWASP Cross-Site Scripting. Available: **https://owasp.org/www-community/attacks/xss/***



The tutor's response:

Good work, Adam, this is a complex idea. There might be an opportunity to present a little detail on what it means to 'escape' data. Some further reading on escaping data at:

- *https://owasp-top-10-proactive-controls-2018.readthedocs.io/en/latest/c4-encode-escape-data.html*
- *https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html*

My response to the tutor

Thank you for your response, Cathryn.

Escaping data is a technique that can be used to help prevent injection attacks. A classic example of an injection attack would be SQL injection (OWASP, 2021). If a website had a search field that allowed users to search a database by text input, the search field would be vulnerable to injection attacks if the data was not properly escaped. The program would accept the user's input and create an SQL query to query the database using the input as a condition. If the user correctly formatted and entered an SQL query into the search field, they could have their query executed on the back end and expose sensitive data.

Un-escaped data is data that is in a dangerous form. When data is escaped the danger is removed. To escape data we make it explicit that portions of the input(data) should be considered as plain text and not a form that has a function.

To prevent the SQL injection discussed above, the text inputted by the user should be searched for any " characters. These characters indicate the end of a piece of text and as such would leave some/all of the remaining input to be treated as SQL (MYSQL,2021). Where the " characters exists it should be escaped by placing \ in front of it to signal the following character is not treated as a special character.

MYSQL.(2021) String Literals. Available from: https://dev.mysql.com/doc/refman/8.0/en/string-literals.html [Accessed 21 May 2021].

OWASP.(2021) SQL Injection. Available from: https://owasp.org/www-community/attacks/SQL_Injection [Accessed 21 May 2021].

Anrich's response:

Hi Adam

Thank you for your reflection on Cross-Site Scripting. I was utterly unaware of how severe a vulnerability it is. I came across various researchers focused on discovering these vulnerabilities in many mainstream services hosted by Google and Facebook ('MKSB(en),' 2021).

As you had stated previously, escaping user input is a crucial step in preventing XSS attacks. However, there are some fascinating examples of XSS where one can pass unclosed tags, which forces the browser to alternate between the Javascript and HTML parsers. Each parser attempts to complete the unclosed tags and results in the script landing in the body and others within the DOM (LiveOverflow, 2019). Google famously has the Google Vulnerability Reward Program, which rewards those who find vulnerabilities in their system and interestingly, XSS scrips mentioned above are regularly rewarded.

Beyond escaping user inputs, a well-known technique used to broaden the attack surface is known as the Content Security Policy; this is mentioned in the OWASP Cross-Site Scripting Cheat Sheet (OWASP, 2021). The Content Security Policy is an HTTP header that is added to every page on a website either by page re-write rules defined within the webserver configuration or added as a meta tag on the page. The primary purpose of CSP is to scrutinise the execution of scripts within the browser by limiting the scope in which the script can be executed. For example, using the following header "`Content-Security-Policy: default-src 'self'`" would only allow execution of scripts executed within its domain, disallowing scripts executing from outside of the domain (Mozilla, 2021).

XSS is an endless topic and proves to be a very interesting vulnerability to study. I trust that we will use this knowledge to apply mitigation concepts and procedures to our code in the future and trust that we have done enough to protect ourselves from prying script kiddies.

***Word Count:** 307*

References

XSS on Google Search - Sanitizing HTML in The Client? (2019).

'MKSB(en)', (2021).

Mozilla (2021) 'Content Security Policy (CSP) - HTTP | MDN'.

OWASP (2021) Cross Site Scripting Prevention - OWASP Cheat Sheet Series.

My Response to Anrich:

Hi Anrich, thank you for your response.

I found your discussion of the Content Security Policy to be very interesting. The CSP enables website developers to reduce a webpages' attack surface by limiting the access for scripts. The CSP source allows the developer to specify an inclusive list of sources that can provide content. The CSP also allows developers fine-grain control over what resources the page is allowed to access. Developers could for example specify a limited number of endpoints that forms can send data to (Mike West, 2020).

The CSP reminded me of the permissions system used on Android devices. On Android apps must request permissions in order to access system features that could be exploited. For example, if an app wants to be able to access GPS data, then the app must location permission. For an app to gain access to a permission the app must display a request to the user that they explicitly grant. I think a benefit of how Android restricts this permission is that

the user is aware of what permissions they grant and can therefore deny and requests that don't make sense (providing the user is tech-savvy). I have found that web developers can access a similar technology using the Permissions API (Mozilla,2021). This Permissions API can grant access to using more powerful Web extension APIs.

References

West, M.(2020) Content Security Policy. Available from: https://developers.google.com/web/fundamentals/security/csp [Accessed 21 May 2021].

Mozilla.(2020) Content Security Policy. Available from: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/permissions/request [Accessed 21 May 2021].

Graces's discussion of SQL injections:

This is my college Grace's discussion of the SQL injection vulnerability.

The Open Web Application Security Project (OWASP) is a non-profit foundation that works to improve the security of software, the organization has produced a top 10 standard awareness document aimed at developers to improve security within web applications, identifying the top 10 web application security risks (OWASP, 2017).

This research focuses on SQL injection flaws - one of the most common cyber-attacks, these attacks occur when untrusted data is sent to an interpreter as part of a command or query, tricking the interpreter into executing unintended commands or accessing data without sufficient authorization.

There are several ways in which SQL injection attacks may be prevented, OWASP (2017) identified one prevention method which uses 'whitelist' server-side input validation this requires the use of special characters for input validation.

The model below provides an example of encryption methods which can be used in order to prevent SQL injection attacks, based on research by Munir and Nasrin (2015).

In this example a randomized encryption algorithm can be used to prevent attacks for example the user input could be encrypted and validated for authorized access if attacks are attempted, they will be prevented because only the encrypted values are checked with the database and not the actual input.

References

Jumaa, A. (2017). Online Database Intrusion Detection System Based on Query Signatures. Journal of University of Human Development. 3. 282-287. 10.21928/juhd.20170315.14.

Munira, R and Nasrin, S. (2015) SQL Injection Prevention Using Random4 Algorithm. Department of Computer Engineering. Available from https://core.ac.uk/download/pdf/55305276.pdf [Accessed 16 May 2021]

OWASP. (2017) A3:2017-Sensitive Data Exposure. Available from: https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure [Accessed 16 May 2021]

My peer-review of grace's discussion

Peer Review - Grace Clarke.

Hi Grace, thank you for your very informative post about SQL injections.

I was familiar with the principle of SQL injections before reading this but had not realized just how many solutions there were to protect the attack surface. Having read that SQL injections make up 24.9% of all attacks I can see why this is an area with so much investigation (Karunanithi, 2018).

I thought that the description of how cryptography can be used to prevent SQL injections was fascinating. In this example - as you explain - the act of encrypting the user's input would scramble any injected SQL so it could not function. An added benefit would be that if someone gained access to the database through other means all the stored data would be meaningless.

During further reading, I found that another technique to handle SQLIA's is the use of parameterized queries. A parameterized query involves breaking the query into two distinct types, the query and the parameters. The query is created with special tokens defining where the parameters will go. Then the parameters are bound to the query before execution (PTSecurity, 2019). The parameters are stored as a different type to an SQL command and so the engine knows not to treat the user's input as SQL instructions. Popular languages such as PHP have convenient libraries for creating and using parameterized queries.

Karunanithi, J.(2018) SQL Injection Prevention Technique Using

Cryptography. Available from: https://repository.stcloudstate.edu/cgi/viewcontent.cgi?article=1105&context=msia_etds#:~:text=Cryptography%20is%20one%20of%20the,used%20to%20encrypt%20the%20data. [Accessed 22 May 2021].

PTSecurity.(2019)How to prevent SQL injection attacks. Available from: https://www.ptsecurity.com/ww-en/analytics/knowledge-base/how-to-prevent-sql-injection-attacks/ [Accessed 22 May 2021].

Anrich's Discussion of sensitive data

A3:2017-Sensitive Data Exposure

Before this module, I had no prior experience with OWASP, although I was familiar with most of the vulnerabilities listed within the OWAP Top Ten. This initial post will explore the sensitive data exposure vulnerability scenario 3 (A3:2017-Sensitive Data Exposure, 2017).

Scenario 3 outlines a scenario where an intruder could access a servers database using what is known as a "file upload flaw" (A3:2017-Sensitive Data Exposure, 2017). A file upload flaw leverages a vulnerability in a web application where an intruder can upload a script to a file upload area, such as a profile picture upload. This upload could facilitate running scripts against the server through the browser URL bar. This type of script will ultimately reveal sensitive configuration information, such as the database configuration file, which stores the username and password of the database. Hacksplainer, a popular website outlining various vulnerabilities, has an interactive experience outlining how a file upload flaw is executed (File Upload Vulnerabilities, 2021).

Once an intruder has access to the database using the configuration information obtained, they can query the database for all usernames and passwords. The passwords in this scenario are poorly hashed; poorly hashed passwords can be unencrypted using a brute force rainbow table attack; a rainbow table is a list of pre-computed hashes which match against poorly hashed passwords.

To mitigate against this type of vulnerability, a developer should ensure they use a hashing algorithm that applies a concept known as salting; salting applies randomly generated characters to a hash in an attempt to obscure the hashed password even further (Password Storage - OWASP Cheat Sheet Series, 2021). When salting, it becomes nearly impossible to match a pre-computed hash to a salted hash due to the added complexity.

Word Count: *280*

References

A3:2017-Sensitive Data Exposure. 2017.

File Upload Vulnerabilities (2021). Available at: https://www.hacksplaining.com/exercises/file-upload.

Password Storage - OWASP Cheat Sheet Series (2021). Available at: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html.

My peer response to Anrich's discussion focusing on the subject of 'Salting':

Thank you for this very informative post Anrich.

I was particularly interested in your discussion of salting and decided to read further about the subject.

You raised the usage of 'rainbow tables' as a means of accessing passwords from exposed databases. I wondered why 'rainbow tables' would work for different databases, and found the reason is that a common set of hashing functions are used (Arias, D, 2021). Commonly used hash algorithms include MD5, SHA-1 and SHA-256 (Rountree, D, 2011). People use functions created by security experts because it is very difficult to create secure hashing algorithms. Because the same hashing algorithms are used, the results for common passwords and their equivalent hashes as output by the 'common' algorithms can be stored and are applicable to any 'unsalted' databases.

I found that it can be dangerous to use randomly generated salts from libraries like C's built-in random() function as they are not random enough. The C implementation of random can produce predictable results (Long, F., 2021). If you use the same salt for the same password on different occasions it exposes a vulnerability in the system.

At first, I thought that perhaps salting was a 'bulletproof' technique but found out that it only really slows hackers down. Salting the passwords does not make the data invulnerable, it merely makes it harder for hackers to access the passwords. The primary reason for salting passwords is to buy time for preventative measures such as asking users to change passwords to occur (Polynomial, 2019).

Salting passwords stop the hackers from using a pre-defined hashtable and forces them to create new hashtables for every salt. This is computationally expensive. It is important to project users salts. If the hackers find the salt for a target user before hacking the database then they can once again create the precomputed hashtable and therefore you won't have time for 'preventative measures.

[324 Words]

Arias, D. (2021) Adding Salt to Hashing: A Better Way to Store Passwords. Available from: https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/ [Accessed 24 May 2021].

Long, F. (2021) Do not use the rand() function for generating pseudorandom numbers. Available from: https://wiki.sei.cmu.edu/confluence/display/c/MSC30-

C.+Do+not+use+the+rand%28%29+function+for+generating+pseudorandom+numbers [Accessed 24 May 2021].

Polynomial. (2019) How to store salt. Available from: https://security.stackexchange.com/questions/17421/how-to-store-salt [Accessed 24 May 2021].

Rountree, D. (2011) Hashing Algorithm. Available from: https://www.sciencedirect.com/topics/computer-science/hashing-algorithm#:~:text=Some%20common%20hashing%20algorithms%20include,very%20commonly%20used%20hashing%20algorithm. [Accessed 24 May 2021].