2023

OWASP Top 10

WORD COUNT: 6653

MARC MALONZO

Background

In 2003, the Open Web Application Security Project (OWASP) produced a list of 10 of the most critical web application and database security vulnerabilities. In detailing effective prevention methods for each vulnerability, OWASP aimed to spread awareness for these neglected aspects of programs thus ensuring developers create programs which do not compromise the safety of user's confidential information. The list, fittingly named the OWASP Top 10, is regularly updated to ensure that the developer is provided with up-to-date information about these security concerns and informed of new methods attackers may use to tamper with their applications. It should additionally be mentioned that the OWASP Top 10 is easily accessed online (https://owasp.org/Top10/), thus allowing aspiring developers, with up-and-coming web applications, to address these issues when programming before the damage is of a bigger scale.

As mentioned previously, the OWASP Top 10 is regularly updated, meaning that vulnerabilities become a more of significant threat overtime, new vulnerabilities are identified, or new ranking methods are implemented. There are initially several common weakness enumerations (CWE), which are groups of weaknesses in software and hardware, that then get grouped into a OWASP Top 10 vulnerability based on the root cause or symptoms of the problem. For example, the 2nd vulnerability on the list, Cryptographic Failure, includes CWEs such as use hard-coded passwords and broken or risky crypto algorithm. There are on average 19.6 CWEs that are covered per category in the list (OWASP, 2021). In ordering the list, the team discuss factors including exploitability, detectability, and technical impact. To assist the process, they weigh each CWE's exploit score and impact score. These scores are based on Common Vulnerability and Exposures (CVE), cases of security flaws that are grouped by CWEs, and are contained in the National Vulnerability Database.

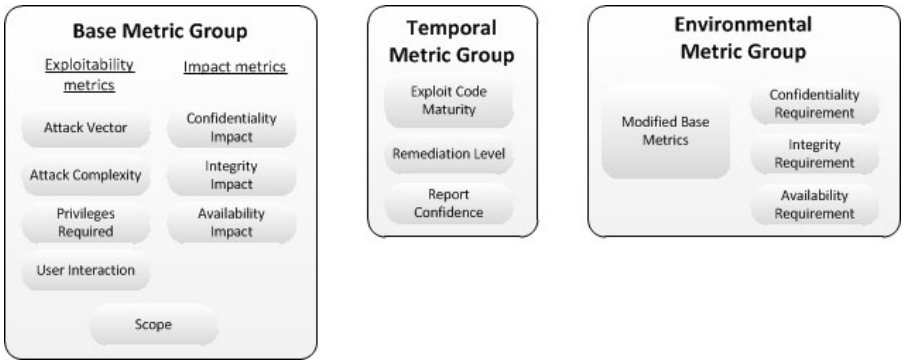

Figure 1 – CVSSv3 Metric Groups (FIRST, n.d.)

CVEs have scores from a Common Vulnerability Scoring System (CVSS) which are composed of the three metric groups seen in figure 1. The base metric group are intrinsic qualities of the vulnerability which are unaffected by environmental factors. The exploitability metrics measure the simplicity and means the vulnerability is exploited whereas the impact metrics measure the magnitude of impact after successful exploitation. The temporal metric group are qualities which may change over time but also remain unaffected by user environment. Environmental factors are specific to a particular user's environment. The exploit score of a CWE is calculated based on the population of CVSSv3 scores and the impact score is calculated based on the remaining CVSSv2 score population. The reason for this is availability of data.

In this report, I will be analysing the each of the OWASP Top 10, explaining what each of them entail, reasons for their placement, their significance and suggested prevention methods.

A01:2021 – Broken Access Control

Access control refers to restrictions imposed upon users such that they cannot act outside of the scope of what is authorised. This functionality is considered 'broken' if restrictions are implemented incorrectly or ineffectively, allowing threat actors to perform malicious activity. The main security concern of broken access control is the leaking of data. If threat actors gain access to this data then they may be able to create greater exploits, compromising user data or program integrity. In the previous iteration of the OWASP Top 10, broken access control was 5th place but has moved up to 1st. The reason for the increase in placement is the large prevalence of new applications and complex security standards. The pervasiveness of this vulnerability is demonstrated as 94% of web applications tested by OWASP contained some form of broken access control, leading to a greater incidence rate of 3.81%, and 318,487 total occurrences (OWASP, 2021).

Notable CWEs:

This vulnerability is grouped based on root cause meaning all CWEs it contains all allow threat actors to manipulate the application database in some way. Of all 34 CWEs listed, the most notable are:

CWE-200 refers to the revealing of sensitive data to an unauthorised user, specifically referring to mistakes that may occur as a result of data management, storage, transmission, or deletion. The severity of the exposure differs to each CVE. Some examples of leaked information include user's confidential information such as the contact information or health data, and code or configurations of the program. These data exposures may arise in various ways. If the program directly inserts sensitive information into the DOM or messages to the client, then the threat actor can easily access and abuse this data. Programs may also indirectly reveal information for example if errors are not handled properly, full path disclosure is possible. Developers may also forget to make sensitive data inaccessible to actors during the development process.

CWE-201 refers to the insertion of possibly confidential information into data objects during responses. An example of a CVE is an application failing to erase the team creator's email address when responding, thus exposing their contact information which is considered sensitive data. As such, the possible ramifications of this weakness are information leakages ranging from the confidential user information to file paths.

CWE-352 is weaknesses which arise due to Cross Site Reference Forgery (CSRF). An application is vulnerable to CSRF if it relies solely on cookies to identify users. This attack requires the threat actor to perform social engineering wherein they deceive their victim into opening a malicious URL that, when opened, will send malicious requests to the application which can allow a threat actor to perform harmful behaviour such as bank transactions.

Real Life Example:

A large-scale example of broken access control is when social media application, Snapchat, had a large data breach where username and contact details of users were leaked in 2014 on a website SnapchatDB.info. The leak occurred due to a vulnerability in the API, specifically, the "find friends with phone numbers" function which contained a security hole (GibsonSec, 2014). The attacker was able to create a script that checks if a phone number is tied to an account, using the fact that phone numbers are incremental, thus they generated a large database of phone numbers mapped to usernames. The incident resulted in the compromising of 4.6 million user's personal information and damage to Snapchat's reputation (Kosner, 2014).

Prevention:

As seen from the CWEs and real example, A01:2021 encompasses vulnerabilities that lead to

unauthorised exposure of sensitive data due to a variety of failures in the web application. As such, developers are encouraged to implement various prevention measures. Access control should occur in trusted server-side code or serverless API such that unauthorised users cannot modify the verification process. Developers must follow principle of least privilege where all private resource access should be denied by default and if certain people should have access to certain resources, then verification of their identity should be enforced. Damage caused by automated attack tooling can also be reduced, as seen in our real-life example, by rate limiting the API and logging can help with identification. Applications are encouraged to use unique, randomly generated session tokens to verify user and invalidate them from the database after logout. If JWT tokens are used, they should be killed after a short window of time to reduce risk of an attacker.

A02:2021 – Cryptographic Failure

When storing user's personal data, a certain level of protection is expected especially if the data falls under privacy laws. Most web application utilise cryptographic systems and techniques to ensure attackers cannot easily understand and utilise the information when leaked. Cryptographic failures could describe situations where the encrypted information is usable, predictable, or easily deciphered. It also covers situations where there is a lack of encryption algorithms and sensitive information is stored in plaintext. Cryptographic failure has an average incidence rate of 4.49% with 233,788 total occurrences (OWASP, 2021). The reason for the rise of this vulnerability is how cryptography rapidly advances leading to cryptographic algorithms, which were once thought to be secure, being exploited or deemed unsafe.

Notable CWEs:

As the root cause of these groups are all failures relating to cryptography, we often see the exposure of confidential information because of the encompassed 29 CWEs. Notable CWEs include:

CWE-327 refers to when cryptographic algorithm which are either broken, or have known safety risks or collisions, are utilised in a program. This makes it easy for threat actors to decrypt or exploit meaning that data integrity and confidentiality is compromised. An example of using of a non-industrial standard or a pre-established insecure cryptographic algorithm is DES as it susceptible to brute force attacks (Rothke, 2010). The security of cryptographic algorithms changes over time because of new attacks being discovered or the increasing of computing power and as such, it is important for developers to keep updated.

CWE-331 refers to a lack of randomness in algorithms in generating tokens, keys, or URLs. This leads to combinations and patterns which threat actors can easily predict, compromising data or allowing them to perform malicious activity. An example where entropy is lacking could be where a user's session token is generated using the user's id, which is a static identifier, therefore an attacker can easily predict the user's session token through simulating the generation process.

Real Life Example:

In 2017, researchers from Google found that the hashing algorithm, SHA-1, was in fact an unsafe algorithm. Prior to the finding, the algorithm was used in digital certificates and generating cryptographic keys such as Git which utilised SHA-1 for version control functions. The reason why the system is considered unsafe is due to the finding of a 'collision', which is where two inputs result in the same output. The researchers utilised 'cryptoanalytic techniques in complex ways' (Rashid, 2017) to allow them to generate prefixes to a PDF file such that the same hash would be generated even with different contents. The possible ramifications of this are that, in Git, an attacker could create a

trustworthy-looking source code and a backdoored one, distributing either to unsuspecting users. As such, after this discovery, there was a mass migration to SHA-2 in many web applications.

Prevention:
As web applications are legally required to securely process, store, and transmit sensitive information, it is important to know how to prevent cryptographic failure. The most important preventative measure is to ensure that up-to-date and strongest algorithms, protocols, and keys are used and if any are found to be unsafe, update the code accordingly. Web applications should also not unnecessarily store sensitive data in the database and in caches. For example, it would be unsafe to store credit card information for a one-time purchase. Overall, using secure protocols such as TLS, storing passwords with 'strong adaptive and salted hashing functions' (OWASP, 2021), ensuring high entropy in random generation and using authenticated encryption can mitigate the likelihood and impact of this vulnerability.

A03:2021 – Injection
Injection occurs when a threat actor can send malicious input data into the web application such that the interpreter or database executes commands. This vulnerability occurs when user input is not validated, filtered, or sanitised by the application and context-aware escaping is not used for dynamic queries or non-parameterised calls. The most common forms of injection are SQL, NoSQL, OS command, Object Relation Mapping, and Object Graph Navigation Library. Injection has an average incidence rate of 3.37% and a total of 274,228 occurrences (OWASP, 2021).

Notable CWEs:
The 33 mapped CWEs listed under this vulnerability are all methods attackers use to inject commands into the web application and the underlying cause is the failure to neutralise elements. Important CWEs are:

CWE-79 is known as cross-site scripting (XSS) and is likely to occur in web applications with improper input handling, allowing threat actors to inject scripts into the HTML. Both XSS and CSRF require a form of social engineering for to work where if for example a user presses on a malicious URL, the threat actor is able perform malicious activity on their behalf. XSS can happen in a variety of ways including:

- Malicious HTTP requests crafted by threat actors through URLs, known as reflected XSS.
- Malicious requests that are stored in the database through actions like log poisoning, known as stored XSS.
- Threat actors inputting malicious payloads into un-sanitised inputs to get a response, known as DOM-Based XSS.

XSS can give access to private information like a user's cookies or password and can allow attackers to hijack their accounts to send malicious requests.

CWE-89 are weaknesses due to SQL injection and occurs if a web application does not perform proper checks on user inputs before sending requests to the database. This allows attackers to generate queries that can be interpreted instead of treated as regular data. Threat actors can view, modify, or destroy databases using this method. Database-driven applications are prone to this problem due to SQL's inability to differentiate control and data planes.

Real Life Example:
In 2019, the web site of Epic Games, the developer of popular games such as Fortnite, was revealed

to have XSS vulnerabilities that could compromise the accounts of unsuspecting victims. The security research firm, known as Check Point, found several subdomains of Epic Games which were vulnerable to both SQLI and XSS due to failure to neutralise page elements such as the search bar. They then directed their attention to the SSO implementation which allowed redirection to any website in the ".epicgames.com" domain (Boxiner, et al., 2019). Thus, by using the vulnerable subdomains, they could input scripts into the redirectedURL parameter, giving a threat actor the authentication token of a victim. If this had not been patched, this could have easily been used in phishing scams, risking the credentials of its userbase.

Prevention:
There are many tools which can be used to detect injection vulnerabilities such as SAST, DAST and IAST before deploying an application. The key to preventing injection is separating data from commands and queries. The most effective method is using a safe API which avoids the interpreter entirely and uses a parameterised interface. On the sever-side, the developer could implement input validation and ensure special characters are escaped from using the proper syntax. Despite these methods, injection is still a large problem due to various limitations such as SQL structure being unescapable and parameterised interfaces still being injectable. As such, LIMIT and other SQL commands can limit the mass disclosure of data.

A04:2021 – Insecure Design

This category was newly introduced in 2021 and refers to a broad category of vulnerabilities where security measures are either missing or ineffective. As a web application developer, it is important to go through the necessary steps to ensure that the proper level of security design is implemented. The process would involve collecting data, negotiating with the clients, and incorporating threat modelling. Despite being a newly introduced vulnerability, the average incidence rate is 3.00% and the total occurrences are 262,407 (OWASP, 2021). Developers are encouraged to incorporate threat modelling when reviewing code, analysing data flow, misuse cases and mitigations in a similar fashion to the diagram in figure 2.

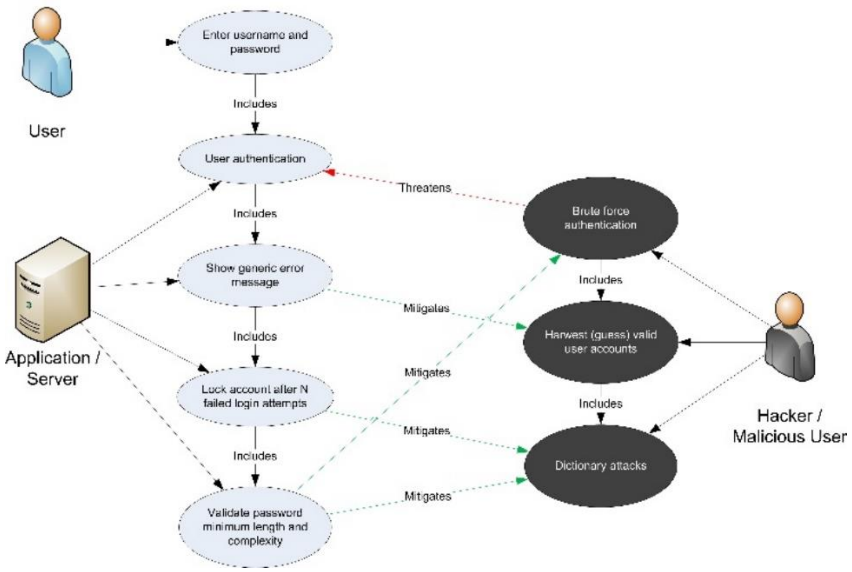

Figure 2- OWASP Example Use and Misuse Cases Graph (Conklin, 2021)

Notable CWEs:
This category includes 40 CWEs and involve errors in application design where the developer may have not considered the requirement of security. The most notable listed by OWASP are:

CWE-209 refers to producing error messages in the web application that contain sensitive information. When an error occurs in a program, messages can either be self-generated through the code or handled by the language interpreter. If attackers gain access to error messages, they gain knowledge of how the web application can be exploited. As seen in figure 2, a mitigation method for this would be to construct self-generated error messages with generic information to protect from threat actors.

CWE-256 is weakness in a program due to passwords being stored in plaintext. This makes file storage of password ineffective as anyone with access can easily utilise the data in malicious ways. Plaintext passwords that are stored in memory must be cleared otherwise they are vulnerable for extraction. Overall, due to the prevalence of injection, storing credentials in plaintext puts users at serious risk, compromising important data.

CWE-501 refers to program weaknesses when trusted and untrusted data are placed in the same data structure or structured message (CWE, 2006). Trust boundary violations are relevant for validation logic, where it becomes difficult to gauge what data should be trusted or rejected. An example of this could be if a web application receives and accepts a request to change someone's password but does not first check if they are logged in.

CWE-522 is the broad collection of CVEs where a web application may not adequately protect credentials through storage or method invocations and allows threat actors to easily retrieve the data. An example of this could be a password recovery system where an attacker can request to retrieve password by solely entering the user's username without authentication.

Real Life Example:
Small-scale examples of this category are significantly more prevalent as applications we use today are required to meet the industry standard when it comes to secure design. An example of insecure design could be CVE-2018-1999036 where the SSH password for the private key was stored inside of the build log (NIST, 2018). This meant that users with access to the log had unauthorised access to the server, potentially compromising accounts, and data.

Prevention:
To ensure web applications are being designed securely, as mentioned before, you should collect data, negotiate, and regularly do threat modelling. It is also helpful to work with security professionals such as AppSec when you are in the designing stage. Utilising libraries with secure design patterns, segregating your system by tiers based on protection requirements and writing unit and integration tests also assist with preventing insecure design.

A05:2021 – Security Misconfiguration

Security misconfiguration is a category which encompasses vulnerabilities where the web application or components are not configured securely. This may mean missing security hardening, security features being disabled, unnecessary features being enabled, default accounts and unchanged passwords. From over 90% of the applications tested by OWASP, there was some form of security misconfiguration, leading to an average incidence rate of 4.51% and total occurrence number of

208,387 (OWASP, 2021). A likely cause of the high average incidence rate is a result of high use of highly configurable software that are widely used in application servers, frameworks, and libraries.

Notable CWEs:
Another result of highly configurable software is a considerable amount of CWEs which fit under this category. There are 20 CWEs mapped to this vulnerability and the most notable of these are:

CWE-16 includes all possible vulnerabilities that could occur during configuration. This CWE is a category as it is a byproduct of the Software Development Life Cycle. As such, there are not CVEs mapped to the category. Related weaknesses include the use of default configurations which may be inappropriate for the coding environment, using weak passwords for protected resources, and the use of obsolete security software.

CWE-611 is a group of CVEs relating to the processing of XML documents. These weaknesses arise from applications that do not sufficiently defend from XXE attacks where threat actors are able to read the contents of local files through manipulating the XML's parser through defining external entities such as 'file://' URIs.

Real Life Example:
In 2017, many companies were subject to sensitive data leaks because of security misconfiguration of the software AWS S3 which is commonly used for cloud storage. The reason for the large number of leaks was poorly secured 'buckets', which is the directory that stores data, that were misconfigured (Thomson, 2017). An example of a large impact is evident with the Australian Broadcasting Corporation who ended up having a large amount of information leaked including MySQL database backups from 2015-2017, thousands of emails, logins and hashed passwords, requests for licensed content and access keys for another repository (iTnews, 2017).

Prevention:
The main prevention method suggested by OWASP is that programs must have secure installation processes. The process should be minimal such that there are no unnecessary features or documentation. The user must also review and update configurations as appropriate to the environment, security notes and updates and ensure that proper permissions are given. OWASP also suggests automating both the process of deploying a new environment and verifying the effectiveness of configurations and settings in all environments. Regular security assessment and code reviews should be conducted to meet industry standards.

A06:2021 – Vulnerable and Outdated Components
In a program, the use of third-party components which have known vulnerabilities, are prone to be exploited by a threat actor. Vulnerable components can come in the form of libraries, plugins, or frameworks. Third-party components are commonly updated to amend bug fixes so if these are not monitored and adjusted in your program, a threat actor could compromise the security of sensitive data or gain access to the source code. It is additionally possible to be at risk if you update but do not check the compatibility of components. This category does not have any mapped CVEs however, it does have an average incidence rate of 8.77% with 30,457 cases (OWASP, 2021).

Notable CWEs:
This category only has 3 CWEs that are mapped to it which all relate to the impact of not monitoring the components in your software. The reason for the small amount of CWEs is that the root cause of

the component's obsoletion can arise from any other vulnerability. Additionally, two of these CWEs are directly from the OWASP Top 10 in the years 2013 and 2017. These CWEs are:

CWE-937 and CWE-1035 both encompass to the OWASP Top 10s of previous years and are considered categories. As such, they are not inherently weaknesses in themselves but are used as a means to group the associated data.

CWE-1104 refers to weaknesses where the application solely relies on a third-party component which no longer receives updates from the developer. The obsoletion of the component can create difficulties when the developer tries to perform code reviews, implement new functions, or debug. Consequently, the program becomes difficult to maintain, affecting the overall security by possibly allowing threat actors to easily find and abuse exploits.

Real Life Example:
The infamous transportation application, Uber, was found to have an outdated version of the WordPress plugin on their website. It was reported by a user that the version, 4.4.2, was vulnerable to Full Path Disclosure (Nour, 2016). This meant that a threat actor could gain view the path of the root file, gaining access to sensitive information or configuration data. The theoretical attack the user listed required an administrator to visit an attacker-controlled website and giving the attacker access to the account. Then the attacker could perform SQL injections as the plugin failed to check POST data before sending the query. The implications of this hypothetical attack would be the compromising of sensitive information which Uber requires to function.

Prevention:
The suggested prevention methods given by OWASP involve a patch management process that allows the developer to maintain the integrity and safety of their program. The monitoring of third-party components is essential and as such, subscribing to email alerts or keeping up to date with sources such as CVE and NVD can keep developers informed regarding new vulnerabilities that may arise. Software composition analysis tools can assist this process such as OWASP Dependency Check and retire.js which automatically inform you of outdated libraries and components. Additionally, for components which are no longer maintained or updated, it is recommended to deploy a virtual patch to monitor, detect, or protect against any known issues.

A07:2021 – Identification and Authentication Failures

Identification and authentication failures encompasses all weaknesses relating to the management of user credentials and sessions. If users are allowed to use weak passwords or reuse session identifiers, then there are authentication weaknesses where attackers can steal sessions and compromise user data. The storage method could also contribute to weaknesses as if you store plaintext or weakly hashed passwords, then this could permit brute force and other automated attacks. Programs that fail to invalidate sessions, expose session identifiers, have ineffective password recovery systems, or missing multi-factor authentication also are considered failures in identification and authentication. These failures have an average incidence rate of 2.55% with 132,195 occurrences (OWASP, 2021).

Notable CWEs:
Since previous years, the identification failures were appended to the list of authentication failures with the total number of CWEs being 22. These CWEs describe cases where there is a failure to invalidate sessions or there is an exposure of user details without authentication. Notable CWEs among the list include:

CWE-297 refers to weaknesses in certificate validation, wherein if host-specific data is insufficiently checked allowing a threat actor to pose as a trusted host. Certificates are commonly used for internet applications which utilise public-key cryptography and may use them for client authentication. As such, developers are encouraged to ensure that certificates are valid and relating to the site being accessed through checking data such as Common Name or Subject Alternative Name.

CWE-287 are the group of CVEs involving the authorisation of the user, where the application inadequately checks whether they are providing correct credentials. An example of this vulnerability is a program which relies solely on cookies when identifying and authenticating a user. A threat actor could bypass the system if they set a victim's cookies through sending a HTTP request meaning the username and password would not need to be verified by the code.

CWE-384 are weaknesses in the system which stem from an inadequate session invalidation system. If a program fails to properly invalidate a user's session after a certain period of time or after logging out, an attacker can hijack their account and perform malicious activity in their place. This could occur in applications where a user can have multiple sessions, attackers can easily predict, or force sessions on their victims.

Real Life Example:
Adobe ColdFusion, an application for building web applications, was recently found to have a weakness fitting under CWE-307, which is Improper Restriction of Excessive Authentication Attempts (Hanchagaiah, 2023). This weakness describes a lack of prevention measures being in place for multiple failed attempts to login in a short time frame. A theoretical threat actor could therefore exploit the weakness by performing brute force attacks which risks the sensitive information of a victim or allowing the attacker to perform malicious activity in their place.

Prevention:
There are various ways to avoid identification and authentication failures. A very common method is the implementation of multi-factor authentication and limiting number of failed login attempts which both counteract brute force attacks. Additionally, ensuring passwords are in line with National Institute of Standards and Technology password policies including the length and complexity, ensures that the program is not susceptible to brute force attacks. It should also be ensured that APIs, messages, and URLs do not contain sensitive information when registering or logging in. This can be done by ensuring registration messages are streamlined and session tokens are randomly generated and securely stored in a session manager. Sessions should also be properly invalidated after a certain amount of time or after logout.

A08:2021 – Software and Data Integrity Failures

Software and Data Integrity Failures are a new category of vulnerabilities introduced in 2021 and refers to applications that does not defend against breaches in integrity. This includes a reliance on plugins, libraries, modules, repositories, and CDNs which may leave the program open to unauthorised access, malicious code, or system compromise. Assuming that all software updates and CI/CD pipelines are trustworthy and ensure integrity allows threat actors to upload and distribute their own updates or pipelines that compromise data. The average incidence rate of this vulnerability is 2.05% with 47,972 occurrences (OWASP, 2021). However, the average weighted impact of this is much greater than previous entries at 7.94, which is the second largest among the Top 10, demonstrating the significance of the category.

Notable CWEs:
This category has 10 CWEs mapped to it which all relate to the compromising of data integrity, resulting from failures in code or infrastructure. Some of the notable CWEs are:

CWE-829 refers to applications that utilise third-party imports, such as libraries, that are inherently malicious or give rise to other weaknesses from its functionality. Threat actors could also disguise libraries or repositories as safe, or they may modify a trusted source to fool developers into using their software, potentially compromising data that allows threat actors to perform other attacks such as malware injection and XSS.

CWE-494 are weaknesses where the program downloads and executes code without first doing checks on the integrity, authenticity, and source. This can allow a threat actor to construct malicious code that is injected into the program. An example could be a code which loads a class from a local subdirectory but does not check if it is the correct one, allowing a threat actor to modify the class to compromise security of the host.

CWE-502 are weaknesses where the program would deserialize potentially untrusted data without validating or sanitising. This is particularly significant because most programs serialize data objects for later use and if proper checks are not in place, then attackers may be able to make unauthorised calls on methods through injecting malicious code, giving them unauthorised access.

Real Life Example:
In December of 2020, a massive data security breach was discovered in the SolarWinds's Orion IT monitoring and management software. The hackers, a group known a Nobelium, performed a supply chain attack wherein, malicious code was inserted into the system and was distributed as an update. The code blended in with normal SolarWinds activity and created a backdoor which allowed the threat actors to impersonate the users of the affected organisations system and compromised their confidential system files. The Orion management system software is utilised by many companies to manage IT resources, so the consequences of the attack was massive. This incident affected over 30,000 organisations including the US government compromising data, networks, and system information, with the attackers also being able to access the networks of partners and customers (Oladimeji & Kerner, 2023).

Prevention:
There are various checks which are important to do in preventing software data and integrity failures. The digital signatures of third-party software can confirm that it is from the expected source and has not been altered by a threat actor. Developers should ensure that libraries and dependencies consume trusted repositories and that there is a code review process in place to check for malicious code in pipelines. CI/CD pipelines should also be checked for code integrity. Supply chain security tools can be used to ensure components have no known vulnerabilities.

A09:2021 – Security Logging and Monitoring Failures

This vulnerability encompasses all weaknesses which arise from lacking response or ability to detect active breaches. Security logging and monitoring is essential during significant runtime events like high value transactions and failed logins as these could be signs of malicious activity. It is also important that error messages are clear and not just stored locally to ensure administrators can monitor and respond to issues. Responses to errors such as dynamic application security testing tools not providing alerts or prevention mechanisms also relate to this category. The average incidence

rate is 6.51% with 53,615 occurrences, however, this data is from interviews asking whether software penetration is detected, impacting the accuracy (OWASP, 2021).

Notable CWEs:
There are only 4 CWEs mapped to this vulnerability which consider different aspects which may make security logs and monitoring insufficient or not up to industry standards. These CWEs are:

CWE-778 groups together CVEs where a program fails to log events properly. When an event occurs, it is expected that it is logged in such a way that malicious behaviour can be identified, and forensic analysis can be used to amend the code after attacks. Cloud storage systems, which are popular among organisations, need to be configured in order to allow detailed logging.

CWE-117 involves programs which fail to neutralise possibly malicious outputs that are written to logs. This may allow threat actors to forge logs or insert malicious content into the log list to be used for later. An example of a way this can be exploited is Log Injection where an attacker may inject an XSS attack into the logs which compromises the program if there is a method invocation when the administrator opens the log storage.

CWE-223 refers to when programs the logs of a program do not display information that is vital in forensic analysis. This makes it impossible to determine whether an attack is taking place and makes it easier for threat actors to perform brute force attacks without detection. An example would be if the program only logs after a certain amount of login attempts and if a threat actor knows the amount, they can strategically segment their automated attacks.

CWE-532 are weaknesses which stem from revealing sensitive information in logs. This information could give threat actors the data they require in devising more malicious attacks. This is prevalent in recently released programs which may accidentally leave logs which were used for debugging during the app's development.

Real Life Example:
There are several examples of this vulnerability which can be from small or large scale in impact. A common occurrence you may see in the news is that the exploit used for a data breach may have been unknown for a significant amount of time. This is a result of insufficient logging. An example of this is Florida Healthy Kids Corporation who revealed in 2021 that the sensitive data of users may have been compromised since 2013 due to security vulnerabilities. The information exposed includes full names, contact information, addresses, insurance, and financial details. It was additionally stated that there was 'no evidence that personal information was altered, used, or accessed' (Bannister, 2021), demonstrating the importance of proper logging in tracing attacks.

Prevention:
For logs to be secure and sufficient in monitoring failures, vulnerabilities and attacks, there are various guidelines the program and developer should follow. Significant events such as logins, access control actions, server-side input validation failures must be logged with enough user information to identify threat actors. Log data should be properly encoded to protect from injections and high-value transactions should be logged in a read-only form, preventing the tampering of information. The NIST also has an incident response and recovery plan which can be followed to prevent insufficient logging and monitoring. DevSecOps should implement effective alerts when monitoring and identifying malicious activity such that it can be dealt with swiftly.

A10:2021 – Server-Side Request Forgery

Web applications which utilise the client-server architecture to function utilise requests in order to fetch resources or perform actions. This category of vulnerability occurs when the program fetches a remote resource without validating the user-supplied URL, allowing threat actors to send malicious requests to unexpected destinations through the application. Despite the low ranking, it is expected to increase as a result of developers prioritizing the user experience and the rise of cloud services and complex architectures. As such, there is an average incidence rate is 2.72% with 9,503 occurrences (OWASP, 2021).

Notable CWEs:

A reason for the low number of occurrences in comparison to other categories is the fact that there is only 1 CWE mapped to the listing since it is a new addition from previous years. More CWEs are likely to arise as more data is collected. The CWE in question is:

CWE-918 are weaknesses that relate to SSRF where the webserver of a program fetches remote resources such as a URL but does not validate whether the request is being sent to the correct destination.

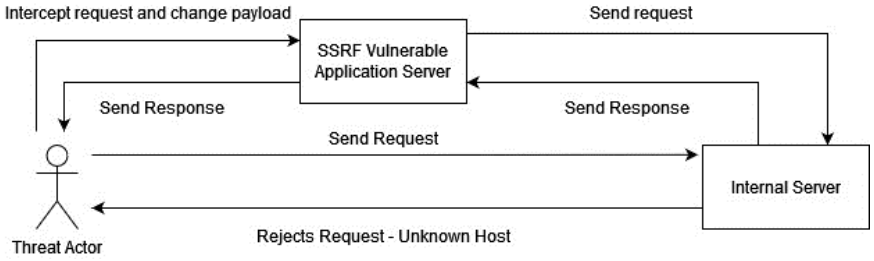

Figure 3 - SSRF Example

As seen in figure 3, threat actors may be able to perform malicious activity by intercepting normal requests in a vulnerable application through the use of interceptors and they may receive a response from the internal server. Thus, they gain access to information they should not be authorised to view, compromising user or server data, possibly bypassing any firewalls.

Real Life Example:

The Web Stories plug-in by Google for WordPress was found to contain a vulnerability that could be exploited via SSRF. This vulnerability allowed threat actors to steal metadata contained within sites hosted on Amazon Web Service servers. The data in question included the AccessIdKey, SecretAccessKey, and Token. This CVE was discovered by Wordfence, who stated that the vulnerability exists due to improper validation of URLs called in the REST API endpoint, '/v1/hotlink/proxy' which could easily be exploited by a threat actor if they found their REST nonce, or identifier, by logging in (Wordfence, 2022). As such, the threat actor does not need high level permissions to utilise SSRF and this makes it possible for them to perform administrative level tasks such as the modification and querying of internal data.

Prevention:

In order to prevent SSRF, you must make considerations of the program in both the Application and the Network layer. In the network layer, a deny by default policy should be integrated such that it blocks all but the essential intranet traffic. Remote resource access functions are also encouraged to be segmented in separate networks to reduce threat actors' access via SSRF. In the application layer, the developer should ensure all user-supplied input data is sanitised and validated and raw responses should not be sent to the client. HTTP redirections should also be restricted, and URL

schemas, ports and destinations should be enforced with a positive allow list. OWASP also suggests checking URL consistency in an effort to prevent DNS rebinding.

Summary

In conclusion, through examination of the vulnerabilities outlined in the OWASP Top 10, web application developers can enhance and develop the security of their program. For each vulnerability, we have used the information provided by OWASP to explain the significance, describe notable common weakness enumerations among the category, and discuss possible prevention methods which can be used to mitigate the frequency and impact.

In observing real life examples of exploitations, we gauge our understanding of the attacker mindset and the consequences of disregarding common security flaws within programs, with many well-known organisations being at risk of large data leaks. Through the scoring system, which involves weighing average incidence rate and impact, we can create a risk assessment table as seen below where 6.44 is the average of all the weighted impacts, 4.12% is the average of all incidence rates and all vulnerabilities are ranked by number of occurrences.

	Lower Impact (Avg Weighted Impact < 6.44)	**Higher Impact (Avg Weighted Impact > 6.44)**
Lower Likelihood (Avg Incidence Rate < 4.12%)	1. Broken Access Control	2. Injection 3. Insecure Design 6. Identification and Authentication Failure 8. Software and Data Integrity Failure 10. Server-Side Request Forgery
Higher Likelihood (Average Incidence Rate > 4.12%)	7. Security Logging and Monitoring Failure 9. Vulnerable and Outdated Components	4. Cryptographic Failure 5. Security Misconfiguration

As seen from the table above, even though broken access control is considered on the lower end in terms of impact and frequency, it still has the most occurrences, demonstrating that the significance of all vulnerabilities in the Top 10 should not be underestimated, and the scoring system is holistic, based on discussion by professionals, and subject to change as datasets become larger.

Through the integration of industry standard practices, such as the regular conducting of security assessments and staying updated on new data breaches, the risk of a threat actor exploiting programs is reduced. The use of various security detection tools listed on the OWASP website is encouraged to ensure that potential security concerns are addressed before a program is made available for public use.

Furthermore, with the growing reliance on digital solutions which use sensitive information for user convenience and the continual evolution of attack methods, confidentiality and integrity of user data is expected. As such, an understanding of the OWASP Top 10 is essential for all developers in bringing awareness and addressing possible exploitations contained in programs during both the development cycle and runtime. Through keeping informed, and the adopting professional practices, we can minimise potential losses and ensure security in the ever-expanding cyberspace.

References

Bannister, A., 2021. *Florida Healthy Kids blames health insurance data breach on third-party hack.* [Online]
Available at: https://portswigger.net/daily-swig/florida-healthy-kids-blames-health-insurance-data-breach-on-third-party-hack
[Accessed 23 October 2023].

Boxiner, A., Vaknin, E. & Vanunu, O., 2019. *Hacking Fortnite Accounts.* [Online]
Available at: https://research.checkpoint.com/2019/hacking-fortnite/
[Accessed 8 October 2023].

Conklin, L., 2021. *Threat Modeling Process.* [Online]
Available at: https://owasp.org/www-community/Threat_Modeling_Process
[Accessed 9 October 2023].

CVE, 2022. *CVE-2022-0708.* [Online]
Available at: https://www.cve.org/CVERecord?id=CVE-2022-0708
[Accessed 3 October 2023].

CVE, 2022. *CVE-2022-3708.* [Online]
Available at: https://www.cve.org/CVERecord?id=CVE-2022-3708
[Accessed 23 October 2023].

CWE, 2006. *CWE CATEGORY: Configuration.* [Online]
Available at: https://cwe.mitre.org/data/definitions/16.html
[Accessed 10 October 2023].

CWE, 2006. *CWE-117: Improper Output Neutralization for Logs.* [Online]
Available at: https://cwe.mitre.org/data/definitions/117.html
[Accessed 23 October 2023].

CWE, 2006. *CWE-200: Exposure of Sensitive Information to an Unauthorized Actor.* [Online]
Available at: https://cwe.mitre.org/data/definitions/200.html
[Accessed 2 October 2023].

CWE, 2006. *CWE-201: Insertion of Sensitive Information Into Sent Data.* [Online]
Available at: https://cwe.mitre.org/data/definitions/201.html
[Accessed 2 October 2023].

CWE, 2006. *CWE-209: Generation of Error Message Containing Sensitive Information.* [Online]
Available at: https://cwe.mitre.org/data/definitions/209.html
[Accessed 9 October 2023].

CWE, 2006. *CWE-223: Omission of Security-relevant Information.* [Online]
Available at: https://cwe.mitre.org/data/definitions/223.html
[Accessed 23 October 2023].

CWE, 2006. *CWE-256: Plaintext Storage of a Password.* [Online]
Available at: https://cwe.mitre.org/data/definitions/256.html
[Accessed 9 October 2023].

CWE, 2006. *CWE-259: Use of Hard-coded Password.* [Online]
Available at: https://cwe.mitre.org/data/definitions/259.html
[Accessed 8 October 2023].

CWE, 2006. *CWE-287: Improper Authentication.* [Online]
Available at: https://cwe.mitre.org/data/definitions/287.html
[Accessed 19 October 2023].

CWE, 2006. *CWE-297: Improper Validation of Certificate with Host Mismatch.* [Online]
Available at: https://cwe.mitre.org/data/definitions/297.html
[Accessed 19 October 2023].

CWE, 2006. *CWE-327: Use of a Broken or Risky Cryptographic Algorithm.* [Online]
Available at: https://cwe.mitre.org/data/definitions/327.html
[Accessed 8 October 2023].

CWE, 2006. *CWE-331: Insufficient Entropy.* [Online]
Available at: https://cwe.mitre.org/data/definitions/331.html
[Accessed 8 October 2023].

CWE, 2006. *CWE-352: Cross-Site Request Forgery (CSRF).* [Online]
Available at: https://cwe.mitre.org/data/definitions/352.html
[Accessed 2 October 2023].

CWE, 2006. *CWE-384: Session Fixation.* [Online]
Available at: CWE-384: Session Fixation
[Accessed 19 October 2023].

CWE, 2006. *CWE-494: Download of Code Without Integrity Check.* [Online]
Available at: https://cwe.mitre.org/data/definitions/494.html
[Accessed 19 October 2023].

CWE, 2006. *CWE-501: Trust Boundary Violation.* [Online]
Available at: https://cwe.mitre.org/data/definitions/501.html
[Accessed 9 October 2023].

CWE, 2006. *CWE-502: Deserialization of Untrusted Data.* [Online]
Available at: https://cwe.mitre.org/data/definitions/502.html
[Accessed 19 October 2023].

CWE, 2006. *CWE-522: Insufficiently Protected Credentials.* [Online]
Available at: https://cwe.mitre.org/data/definitions/522.html
[Accessed 9 October 2023].

CWE, 2006. *CWE-532: Insertion of Sensitive Information into Log File.* [Online]
Available at: https://cwe.mitre.org/data/definitions/532.html
[Accessed 23 October 2023].

CWE, 2006. *CWE-73: External Control of File Name or Path.* [Online]
Available at: https://cwe.mitre.org/data/definitions/73.html
[Accessed 8 October 2023].

CWE, 2006. *CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting').* [Online]

Available at: https://cwe.mitre.org/data/definitions/79.html
[Accessed 8 October 2023].

CWE, 2006. *CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection').* [Online]
Available at: https://cwe.mitre.org/data/definitions/89.html
[Accessed 8 October 2023].

CWE, 2007. *CWE-611: Improper Restriction of XML External Entity Reference.* [Online]
Available at: https://cwe.mitre.org/data/definitions/611.html
[Accessed 10 October 2023].

CWE, 2009. *CWE-778: Insufficient Logging.* [Online]
Available at: https://cwe.mitre.org/data/definitions/778.html
[Accessed 23 October 2023].

CWE, 2010. *CWE-829: Inclusion of Functionality from Untrusted Control Sphere.* [Online]
Available at: https://cwe.mitre.org/data/definitions/829.html
[Accessed 19 October 2023].

CWE, 2013. *CWE CATEGORY: OWASP Top Ten 2013 Category A9 - Using Components with Known Vulnerabilities.* [Online]
Available at: https://cwe.mitre.org/data/definitions/937.html
[Accessed 18 October 2023].

CWE, 2013. *CWE-918: Server-Side Request Forgery (SSRF).* [Online]
Available at: https://cwe.mitre.org/data/definitions/918.html
[Accessed 23 October 2023].

CWE, 2018. *CWE CATEGORY: OWASP Top Ten 2017 Category A9 - Using Components with Known Vulnerabilities.* [Online]
Available at: https://cwe.mitre.org/data/definitions/1035.html
[Accessed 18 October 2023].

CWE, 2018. *CWE-1104: Use of Unmaintained Third Party Components.* [Online]
Available at: https://cwe.mitre.org/data/definitions/1104.html
[Accessed 18 October 2023].

Doerrfeld, B., 2022. *Why Is Broken Access Control OWASP's #1 Threat?.* [Online]
Available at: https://accelerationeconomy.com/cybersecurity/why-is-broken-access-control-owasps-1-threat/
[Accessed 29 September 2023].

Eddy, N., 2022. *Google WordPress Plug-in Bug Allows AWS Metadata Theft.* [Online]
Available at: https://www.darkreading.com/vulnerabilities-threats/google-wordpress-plugin-bug-aws-metadata-theft
[Accessed 23 October 2023].

FIRST, n.d. *Common Vulnerability Scoring System v3.0: Specification Document.* [Online]
Available at: https://www.first.org/cvss/v3.0/specification-document
[Accessed 2 October 2023].

GibsonSec, 2014. *Snapchat - GSFD.* [Online]
Available at: https://gibsonsec.org/snapchat/fulldisclosure/#the-find_friends-exploit
[Accessed 4 October 2023].

Hanchagaiah, P., 2023. *Adobe ColdFusion Vulnerabilities Exploited in Wild.* [Online]
Available at: https://securityboulevard.com/2023/07/adobe-coldfusion-vulnerabilities-exploited-in-wild/
[Accessed 19 October 2023].

IBM, 2023. *Uses for digital certificates in Internet applications.* [Online]
Available at: https://www.ibm.com/docs/en/aix/7.2?topic=concepts-uses-digital-certificates-in-internet-applications
[Accessed 19 October 2023].

iTnews, 2017. *ABC exposes sensitive data in S3 bungle.* [Online]
Available at: https://www.itnews.com.au/news/abc-exposes-sensitive-data-in-s3-bungle-477833
[Accessed 10 October 2023].

Kosner, A. W., 2014. *4.6 Million Snapchat Usernames And Phone Numbers Captured By API Exploit.* [Online]
Available at: https://www.forbes.com/sites/anthonykosner/2014/01/01/4-6-million-snapchat-usernames-and-phone-numbers-captured-by-api-exploit/?sh=7600d1535294
[Accessed 4 October 2023].

LIFARS, 2019. *Fortnite Vulnerability Exposes 80 million accounts.* [Online]
Available at: https://www.lifars.com/2019/01/fortnite-vulnerability-exposes-80-million-accounts/
[Accessed 8 October 2023].

NIST, 2018. *NVD - CVE-2018-1999036.* [Online]
Available at: https://nvd.nist.gov/vuln/detail/CVE-2018-1999036
[Accessed 9 October 2023].

Nour, A., 2016. *Multiple vulnerabilities in a WordPress plugin at drive.uber.com.* [Online]
Available at: https://hackerone.com/reports/135288
[Accessed 18 October 2023].

Oladimeji, S. & Kerner, S. M., 2023. *SolarWinds hack explained: Everything you need to know.* [Online]
Available at: https://www.techtarget.com/whatis/feature/SolarWinds-hack-explained-Everything-you-need-to-know
[Accessed 23 October 2023].

OWASP, 2019. *Full Path Disclosure.* [Online]
Available at: https://owasp.org/www-community/attacks/Full_Path_Disclosure
[Accessed 18 October 2023].

OWASP, 2019. *Log Injection.* [Online]
Available at: https://owasp.org/www-community/attacks/Log_Injection
[Accessed 23 October 2023].

OWASP, 2021. *A01:2021 – Broken Access Control.* [Online]
Available at: https://owasp.org/Top10/A01_2021-Broken_Access_Control/
[Accessed 2 October 2023].

OWASP, 2021. *A02:2021 – Cryptographic Failures.* [Online]
Available at: https://owasp.org/Top10/A02_2021-Cryptographic_Failures/
[Accessed 8 October 2023].

OWASP, 2021. *A03:2021 – Injection.* [Online]
Available at: https://owasp.org/Top10/A03_2021-Injection/
[Accessed 8 October 2023].

OWASP, 2021. *A04:2021 – Insecure Design.* [Online]
Available at: https://owasp.org/Top10/A04_2021-Insecure_Design/
[Accessed 9 October 2023].

OWASP, 2021. *A05:2021 – Security Misconfiguration.* [Online]
Available at: https://owasp.org/Top10/A05_2021-Security_Misconfiguration/
[Accessed 10 October 2023].

OWASP, 2021. *A06:2021 – Vulnerable and Outdated Components.* [Online]
Available at: https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/
[Accessed 18 October 2023].

OWASP, 2021. *A07:2021 – Identification and Authentication Failures.* [Online]
Available at: https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/
[Accessed 19 October 2023].

OWASP, 2021. *A08:2021 – Software and Data Integrity Failures.* [Online]
Available at: https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/
[Accessed 19 October 2023].

OWASP, 2021. *A09:2021 – Security Logging and Monitoring Failures.* [Online]
Available at: https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/
[Accessed 23 October 2023].

OWASP, 2021. *A10:2021 – Server-Side Request Forgery (SSRF).* [Online]
Available at: https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/
[Accessed 23 October 2023].

OWASP, 2021. *Introduction - OWASP Top 10:2021.* [Online]
Available at: https://owasp.org/Top10/A00_2021_Introduction/
[Accessed 29 September 2023].

Rashid, F. Y., 2017. *Google kills SHA-1 with successful collision attack.* [Online]
Available at: https://www.infoworld.com/article/3173845/google-kills-sha-1-with-successful-collision-attack.html
[Accessed 8 October 2023].

Red Hat, 2021. *What is a CVE?.* [Online]
Available at: https://www.redhat.com/en/topics/security/what-is-cve
[Accessed 29 September 2023].

Rothke, B., 2010. *Brute Force: Cracking the Data Encryption Standard.* [Online]
Available at: https://www.rsaconference.com/library/blog/brute-force-cracking-the-data-encryption-standard
[Accessed 8 October 2023].

Security Journey, 2023. *OWASP Top 10 Broken Access Control Explained.* [Online]
Available at: https://www.securityjourney.com/post/owasp-top-10-broken-access-control-explained
[Accessed 29 September 2023].

Sharwood, S., 2017. *Australian Broadcasting Corporation leaks passwords, video from AWS S3 bucket.* [Online]
Available at:
https://www.theregister.com/2017/11/16/australian_broadcasting_corporation_leaks_data_from_s3_bucket/
[Accessed 10 October 2023].

Stevens, M. et al., n.d. *SHAttered.* [Online]
Available at: https://shattered.io/
[Accessed 8 October 2023].

Thomson, I., 2017. *Amazon's answer to all those leaky AWS S3 buckets: A dashboard warning light.* [Online]
Available at: https://www.theregister.com/2017/11/07/amazon_aws_s3_alert/
[Accessed 10 October 2023].

Wordfence, 2022. *Exploiting WordPress Plugin Vulnerabilities to Steal AWS Metadata.* [Online]
Available at: https://www.wordfence.com/blog/2022/12/exploiting-wordpress-plugin-vulnerabilities-to-steal-aws-metadata/
[Accessed 23 October 2023].