What is the OWASP Top Ten?

OWASP Top Ten is the list of the 10 most common application vulnerabilities. It also shows their risks, impacts, and countermeasures. Updated every three to four years, the latest **OWASP vulnerabilities list** was released September 24, 2021. Let's dive into some of the changes.

The Top 10 OWASP vulnerabilities in 2021 are:

1. Broken Access Control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. Server-Side Request Forgery

1

Broken Access Control

Broken Access Control moved up from the fifth most severe risk in 2017 to the top risk in 2021. There were more instances of Common Weakness Enumerators (CWE) for this than any other category.

Here are some examples of what we consider to be "access":

- Access to a hosting control / administrative panel
- Access to a server via FTP / SFTP / SSH
- Access to a website's administrative panel
- Access to other applications on your server
- Access to a database

Attackers can exploit authorization flaws to accomplish the following:

- Access unauthorized functionality and/or data
- View sensitive files
- Change access rights
- Edit files and records

What Are the Risks of Broken Access Control?

Here are a few examples provided by OWASP of what can happen when there is broken access control:

- **Scenario #1:** The website is using a **vulnerable** version of WordPress, 4.7.1. One of the REST endpoints allows access (via the API) to view, edit, delete and create posts. Within this particular endpoint, a subtle bug allows **unauthenticated visitors to edit any post on the site**.

  This makes it possible for an attacker to send a request like: **/wp-json/wp/v2/posts/1234?id=12345helloworld** which would assign **12345helloworld** to the **ID** parameter – which now contains more than just digits.

- **Scenario #2:** An attacker simply force browses to target URLs. Admin rights are required for access to the admin page. http://example.com/app/getappInfo

Developers are going to be more familiar with the above scenarios, but remember that broken access control vulnerabilities can be expressed in many forms through almost every web technology out there. It all depends on what you use on your website.

Reducing the Risks of Broken Access Control

There are things you can do to reduce the risks of broken access control:

- Employ **least privileged concepts** – apply a role appropriate to the task and only for the amount of time necessary to complete said task.
- Get rid of accounts you don't need or whose user no longer requires access.
- Audit your servers and websites – who is doing what, when, and why.
- If possible, apply multi-factor authentication (MFA) to all your access points.
- Disable access points until they are needed in order to reduce your access windows
- Remove unnecessary services from your server.
- Check applications that are externally accessible versus applications that are tied to your network.
- If you are developing a website, bear in mind that a production box should not be the place to develop, test, or push updates without testing.

Broken Access Control Prevention

To avoid broken access control you should develop and configure software with a security-first philosophy. It is important to work with a developer to make sure there are security requirements in place.

The technical recommendations by OWASP to prevent broken access control are:

- With the exception of public resources, deny by default.
- Implement access control mechanisms once and reuse them throughout the application, including minimizing CORS usage.
- Model access controls should enforce record ownership, rather than accepting that the user can create, read, update, or delete any record.

Note: For example, if a user logs in as "John," he could only create, read, update or delete records associated with the ID of "John," never the data from other users.

- Unique application business limit requirements should be enforced by domain models.
- Disable web server directory listing and ensure file metadata (e.g. .git) and backup files are not present within web roots and are not publicly accessible.

- Log access control failures, alert admins when appropriate (e.g. repeated failures).
- Rate limit API and controller access to minimize the harm from automated attacks.
- JWT tokens should be invalidated on the server after logout.
- Developers and QA staff should include functional access control units and integration tests.

2

Cryptographic Failures

Previously known as "Sensitive Data Exposure", it was renamed to better reflect the root cause of the issue. It moves up from number three to runner-up in widespread vulnerabilities on the OWASP list. It consists of a failure to protect sensitive data that should not have been publicly accessible.

Examples of Cryptographic Failures

Sensitive data that requires protection includes:

- Credentials
- Credit card numbers
- Social Security numbers
- Medical information
- Personally identifiable information (PII)
- Other personal information

It is vital for any organization to understand the importance of protecting users' information and privacy. All companies should understand and comply with their local privacy laws as well as any regional ones where they conduct business in.

Responsible sensitive data collection and handling has become more noticeable, especially with the advent of the General Data Protection Regulation (GDPR). **GDPR** is a fairly recent data privacy law that went into effect May 25, 2018. It mandates how companies collect, modify, process, store, delete and use personal data originating in the European Union for both residents and visitors.

There are two types of data:

- Stored data – data at rest
- Transmitted data – data that is transmitted internally between servers, or to web browsers

Protecting Data in Transit

Both types of data should be protected. When considering data in transit, one way to protect it on a website is by having an **SSL certificate**

SSL is the acronym for **Secure Sockets Layer**. It is the standard security technology for establishing an encrypted link between a web server and a browser. SSL certificates help **protect the integrity of the data in transit** between the host (web server or firewall) and the client (web browser).

We have created a DIY guide to help every website owner learn how to install an **SSL certificate**.

What are the risks of Cryptographic Failure

OWASP provides a few examples of what can happen when sensitive data is exposed:

- **Scenario #1:** An application encrypts credit card numbers in a database using automatic database encryption. However, this data is automatically decrypted when retrieved, allowing an SQL injection flaw to retrieve credit card numbers in clear text.
- **Scenario #2:** A site doesn't use or enforce TLS for all pages or supports weak encryption. An attacker monitors network traffic (e.g. at an insecure wireless network), downgrades connections from HTTPS to HTTP, intercepts requests, and **steals the user's session cookie**. The attacker then replays this cookie and hijacks the user's (authenticated) session, accessing or modifying the user's private data. Instead of the above, they could alter all transported data, e.g. the recipient of a money transfer.
- **Scenario #3:** The password database uses unsalted or simple hashes to store everyone's passwords. A file upload flaw allows an attacker to retrieve the password database. All the unsalted hashes can be exposed with a rainbow table of pre-calculated hashes. Hashes generated by simple or fast hash functions may be cracked by GPUs, even if they were salted.

Why is Cryptographic Failure so common?

Over the last few years, **sensitive data exposure** has been one of the most common attacks around the world. Some examples of data leaks exposed sensitive data include:

- The Equifax data breach of 2017 resulted in the compromise of personal information of nearly 150 million Americans, over 15 million British citizens and almost 20,000 Canadians. In a resulting lawsuit the firm was ordered to pay over half a billion dollars in fines/payouts. One law firm launched the largest class action lawsuit in US history against Equifax seeking up to $70 billion USD in damages.
- In June of 2021 LinkedIn reported that information from 90% of its user base was compromised and posted on the dark web. Fortunately no sensitive personal information was compromised, but the leaked details included things like email addresses, phone numbers and geolocation data (certainly enough to aid hackers in spear phishing campaigns).
- Clocking in as the most severe data breach of all time: Yahoo! reported that years prior their entire user 3-billion-strong base had been compromised. Specific details of material taken include names, email addresses, telephone numbers, encrypted or unencrypted security questions and answers, dates of birth, and hashed passwords. The company was in the middle of being acquired by Verizon at the time and the reported breach cost Yahoo! $350 million USD, not including a litany of class action lawsuits launched against the company.

Non-encrypted sensitive data is the main reason these attacks are so widespread. However, even if data is encrypted, it can still be broken due to weak areas like:

- Key generation process
- Key management process
- Algorithm usage
- Protocol usage
- Cipher usage
- Password hashing storage techniques

This vulnerability is difficult to exploit; however, the consequences of a successful attack are profound. If you want to learn more about such impacts, we have written a blog post on the **Impacts of a Security Breach**.

How to Prevent Data Exposure

Some of the ways to prevent data exposure, according to OWASP, are:

- Classify the data processed, stored, or transmitted by an application.
- Identify what data is sensitive according to privacy laws, regulatory requirements, or business needs.
- Apply controls as per the classification.
- Don't store sensitive data unnecessarily.

- Discard sensitive data as soon as possible or use PCI DSS compliant tokenization or truncation. **Remember: data that is not retained cannot be stolen**.
- Make sure to encrypt all sensitive data at rest.
- Ensure that up-to-date and strong standard algorithms, protocols, and keys are in place; use proper key management.
- Encrypt all data in transit with secure protocols such as TLS with perfect forward secrecy (PFS) ciphers, cipher prioritization by the server, and secure parameters.
- Enforce encryption using directives like HTTP Strict Transport Security (HSTS).
- Disable caching for responses that contain sensitive data.
- Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor), such as Argon2, scrypt, bcrypt, or PBKDF2.
- Verify independently the effectiveness of configuration and settings.

3

Injection

A code injection happens when an attacker sends invalid data to the web application with the intention of making it do something that the application is not designed/programmed to do.

Perhaps the most common example around this security vulnerability is the **SQL query consuming untrusted data**. You can see one of OWASP's examples below:

```
String query = "SELECT * FROM accounts WHERE custID = '" +
request.getParameter("id") + "'";
```
Copy

This query can be exploited by calling up the web page and executing it with the following URL: https://example.com/app/accountView?id=' or '1'='1 causing the return of all the rows stored on the database table.

The core of a code injection vulnerability is the lack of validation and sanitization of the data used by the web application, which means that this vulnerability can be present on almost any type of technology related to websites.

Anything that accepts parameters as input can be vulnerable to a code injection attack.

We've written a lot about **code injection attacks**. One of the most recent examples was a code injection vulnerability within the very popular Simple 301 Redirects plugin in WordPress. It made it possible for unauthenticated users to inject code that would redirect all website traffic to a malicious domain of the attackers choosing. The vulnerability affected over 300,000 websites and was ranked as a 9.9 on the CVSS scale.

How do you prevent code injection vulnerabilities

Preventing code injection vulnerabilities really depends on the technology you are using on your website. For example, if you use WordPress, you could minimize code injection vulnerabilities by minimizing the number of plugins and themes installed.

If you have a tailored web application and a dedicated team of developers, you need to make sure to have security requirements your developers can follow when they are designing and writing software. This will allow them to keep thinking about security during the lifecycle of the project.
Here are OWASP's technical recommendations to prevent SQL injections:

Preventing SQL injections requires keeping data separate from commands and queries.

- The preferred option is to use a safe API, which avoids the use of the interpreter entirely or provides a parameterized interface or migrate to use Object Relational Mapping Tools (ORMs). Note: Even when parameterized, stored procedures can still introduce SQL injection if PL/SQL or T-SQL concatenates queries and data, or executes hostile data with EXECUTE IMMEDIATE or exec().
- Use positive or "allowlist" server-side input validation. This is not a complete defense as many applications require special characters like text areas or APIs for mobile applications.
- For any residual dynamic queries, escape special characters using the specific escape syntax for that interpreter. Note: SQL structure such as table names and column names cannot be escaped, and thus user-supplied structure names are dangerous. This is a common issue in report-writing software.
- Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL injection.

From these recommendations you can conclude two things:

- Separation of data from the web application logic.
- Implement settings and/or restrictions to limit data exposure in case of successful injection attacks.

Without appropriate measures in place, code injections represent a serious risk to website owners. These attacks leverage security loopholes for a hostile takeover or the leaking of confidential information.

4

Insecure Design

A new addition to the OWASP Top Ten, clocking in at number four on the list, is insecure design. This focuses on the ground-up development of web applications from the very beginning of its life cycle. This is not to be confused with insecure implementation of web applications or policies. One can have a secure design and insecure implementation but not the other way around. It is, essentially, the avoidance of hard-coded security protocols and methods within the initial development of a web application, as well as the failure to take into account risks and attack vectors during the planning, development, and implementation of a web application.

What Are the Risks of Insecure Design?

Software developers have a responsibility to write secure applications that do not put its users at risk. Applications that were not developed with security in mind from the very beginning are more likely to put user data and security at risk, and require updates, patches, and fixes to prevent these risks. Applications without secure design are low hanging fruit for attackers and can cost incalculable sums of damage in terms of leaked data, tarnished reputations, and paid working-hours of cleanup and future prevention.

Examples of Insecure Design

Insecure design is unfortunately quite common within web applications. Some examples include:

- Most CMS platforms, including WordPress, do not limit the number of failed logins on the administrator panel. This renders them particularly vulnerable to **brute force attacks** and requires the installation of third-party security extensions to mitigate.

- By default, **symlink race condition protection** within WHM / cPanel environments is disabled. This allows attackers to move laterally through the network if one website is compromised. Symlink protection must be manually enabled by the administrator to prevent this from being exploited.
- Many CMS platforms use a default administrator panel URL. For example, wp-admin in WordPress and administrator in Joomla. This (especially combined with our first example) renders them even more vulnerable to brute force attacks. Magento2 has taken a step in the correct direction in partially randomising each new website's administrator panel URL
- Many ecommerce platforms do not contain built in protection from automated bot transactions. This renders them vulnerable to both scalpers buying up tickets or computer components, and attackers testing stolen credit card details on victim websites.

How to Prevent Insecure Design

To borrow from the **OWASP top ten list**:
"Secure design is a culture and methodology that constantly evaluates threats and ensures that code is robustly designed and tested to prevent known attack methods"

By taking security into account from the very bedrock of the development of a web application, many easily preventable risks can be avoided. Secure design is not a ruleset nor a tool, it is a culture, mindset and methodology.

- Security specialists should be consulted at the beginning of a project and throughout the entire development lifecycle
- Make heavy usage of threat modeling
- Consider potential attack vectors and the level of exposure that your web application will have
- Analyze (and re-analyze) all data flows, particularly ones that resist the threat modeling
- Use a href="https://owaspsamm.org/"SAMM as a guide to development

5

Security Misconfigurations

This category moves up one notch from the previous top 10 list published in 2017. The previous category for XML External Entities (XXE) has been rolled into this one. There is a litany of possible security misconfigurations, but here are the most common:

- Unpatched flaws
- Default configurations
- Unused pages
- Unprotected files and directories
- Unnecessary services
- Usage of vulnerable XML files

One of the most common webmaster flaws is keeping the CMS default configurations.

Today's CMS applications (although easy to use) can be tricky from a security perspective for the end users. By far, the most common attacks are entirely automated. Many of these attacks rely on users to have only default settings.

This means that a large number of attacks can be mitigated by changing the default settings when installing a CMS.

There are settings you may want to adjust to control comments, users, and the visibility of user information. The file permissions are another example of a default setting that can be hardened.

Where Can Security Misconfiguration Happen?

Misconfiguration can happen at any level of an application stack, including:

- Network services
- Platform
- Web server
- Application server
- Database
- Frameworks
- Custom code
- Pre-installed virtual machines
- Containers
- Storage

One of the most recent examples of application misconfigurations is the **memcached servers** used to **DDoS** huge services in the Tech Industry.
Examples of Security Misconfiguration Attacks

According to OWASP, these are some examples of attack scenarios:

- **Scenario #1:** The application server comes with sample applications that are not removed from the production server. These sample applications have known security flaws that attackers use to compromise the server. If one of these applications is the admin console and default accounts weren't changed, the attacker logs in with default passwords and takes over.
- **Scenario #2:** Directory listing is not disabled on the server. An attacker discovers they can simply list directories. They find and download the compiled Java classes, which they decompile and reverse engineer to view the code. The attacker then finds a serious access control flaw in the application.
- **Scenario #3:** The application server's configuration allows detailed error messages, e.g. stack traces, to be returned to users. This potentially exposes sensitive information or underlying flaws, such as component versions. They are known to be vulnerable.
- S**cenario #4:** A cloud service provider has default sharing permissions open to the Internet by other CSP users. This allows stored sensitive data to be accessed within cloud storage.

How to Secure Installation Systems

In order to prevent security misconfiguration use:

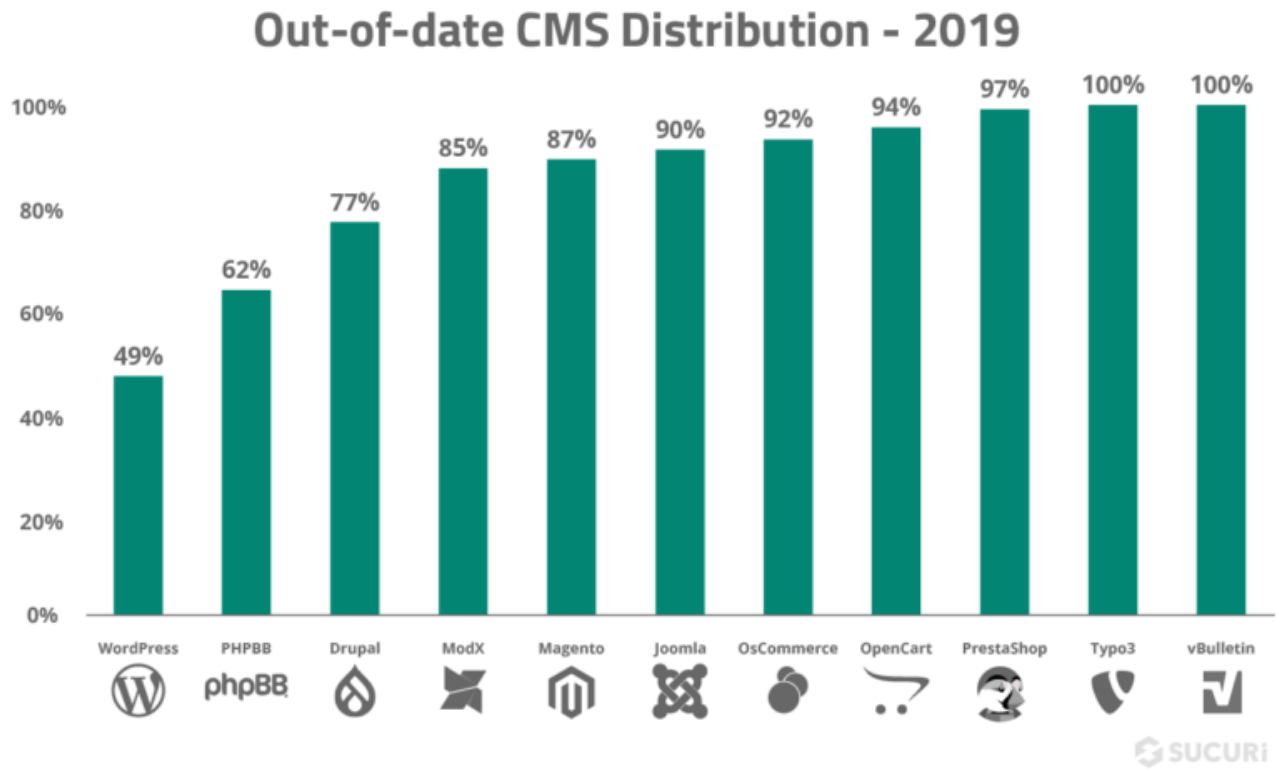
- A repeatable hardening process that makes it fast and easy to deploy another environment that is properly locked down. Development, QA, and production environments should all be configured identically, with different credentials used in each environment. Automate this process in order to minimize the effort required to set up a new secure environment.
- A minimal platform without any unnecessary features, components, documentation, and samples. Remove or do not install unused features and frameworks.
- A task to review and update the configurations appropriate to all security notes, updates, and patches as part of the patch management process. In particular, review cloud storage permissions.
- A segmented application architecture that provides effective and secure separation between components or tenants, with segmentation, containerization, or cloud security groups.
- Sending security directives to clients, e.g. Security Headers.
- An automated process to verify the effectiveness of the configurations and settings in all environments.

6

Vulnerable and Outdated Components

Even simple websites such as personal blogs have a lot of dependencies, plugins, extensions and third party code. Failing to update every piece of software on the backend and frontend of a website will introduce heavy security risks sooner rather than later. Attackers actively seek out websites using vulnerable components and aggressively exploit them to spread malware, spam and phishing.

For example, in 2019, 56% of all CMS applications were out of date at the point of infection.



The question is, why aren't we updating their software on time? Why is this still such a huge problem today?

There are some possibilities, such as:

- Webmasters/developers cannot keep up with the pace of the updates; after all, updating properly takes time.
- Legacy code won't work on newer versions of its dependencies.
- Webmasters are scared that something will break on their website.
- Webmasters don't have the expertise to properly apply the update.

This might sound dramatic, but every time you disregard an update warning you might be allowing a now known vulnerability to survive in your system. Trust us, cybercriminals are quick to investigate software and changelogs.

Whatever the reason for running out-of-date software on your web application, you can't leave it unprotected. Both Sucuri and OWASP recommend virtual patching for the cases where patching is not possible.

Virtual patching affords websites that are outdated (or with known vulnerabilities) to be protected from attacks by preventing the exploitation of these vulnerabilities on the fly. This is usually done by a **firewall** and an intrusion detection system (IDS).

Vulnerable Applications

Vulnerable applications are usually outdated, according to OWASP guidelines, if:

- You do not know the versions of all components you use (both client-side and server-side). This includes components you directly use as well as nested dependencies.
- The software is vulnerable, unsupported, or out of date. This includes the OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries.
- You do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion. This commonly happens in environments when patching is a monthly or quarterly task under change control, which leaves organizations open to many days or months of unnecessary exposure to fixed vulnerabilities.
- The software developers do not test the compatibility of updated, upgraded, or patched libraries.
- You do not secure the components' configurations.

You can **subscribe to our security blog feed** to be on top of security issues caused by vulnerable applications.

How to Avoid Using Components with Known Vulnerabilities

Some of the ways to prevent the use of vulnerable components are:

- Remove all unnecessary dependencies.
- Keep an inventory of all your components on the client-side and server-side.
- Monitor sources like Common Vulnerabilities and Disclosures **(CVE)** and National Vulnerability Database **(NVD)** for vulnerabilities in the components.
- Scan your website with a security testing tool such as **WPScan**
- Obtain components only from official sources.
- Delete components not actively maintained.
- Use virtual patching with the help of a **Website Application Firewall**.

7

Identification and Authentication Failures

Previously number two on the OWASP list, "broken authentication" has been renamed to this and now ranked at number seven. A broken authentication vulnerability can allow an attacker to use manual and/or automatic methods to try to gain control over any account they want in a system – or even worse – to gain complete control over the system.

Websites with broken authentication vulnerabilities are very common on the web. Broken authentication usually refers to logic issues that occur on the application authentication's mechanism, like bad session management prone to username enumeration – when a malicious actor uses brute-force techniques to either guess or confirm valid users in a system.

To minimize authentication failure risks avoid leaving the login page for admins publicly accessible to all visitors of the website:

- /administrator on Joomla!
- /wp-admin/ on WordPress
- /index.php/admin on Magento

- /user/login on Drupal.

Brute force username/password combinations against those pages.

Types of Authentication Failure Vulnerabilities

According to the OWASP Top 10, these vulnerabilities can come in many forms. A web application contains a broken authentication vulnerability if it:

- Permits automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords.
- Permits brute force or other automated attacks.
- Permits default, weak, or well-known passwords, such as"Password1" or "admin/admin."
- Uses weak or ineffective credential recovery and forgot-password processes, such as "knowledge-based answers," which cannot be made safe.
- Uses plain text, encrypted, or weakly hashed passwords.
- Has missing or ineffective multi-factor authentication (MFA).
- Exposes session IDs in the URL (e.g., URL rewriting).
- Does not rotate session IDs after successful login.
- Does not properly invalidate session IDs. User sessions or authentication tokens (particularly single sign-on (SSO) tokens) aren't properly invalidated during logout or a period of inactivity.

Writing insecure software results in most of these vulnerabilities. They can be attributed to many factors such as lack of experience from the developers. It can also be the consequence of more institutionalized failures such as lack of security requirements or organizations rushing software releases, in other words, choosing working software over secure software.

How do you prevent authentication failures?

In order to avoid authentication failure make sure the developers apply to the best practices of website security. Support them by providing access to external security audits and enough time to properly test the code before deploying to production.

OWASP's technical recommendations are the following:

- Where possible, implement multi-factor authentication to prevent automated, credential stuffing, brute force, and stolen credential reuse attacks.
- Do not ship or deploy with any default credentials, particularly for admin users.
- Implement weak-password checks, such as testing new or changed passwords against a list of the top 10,000 worst passwords.
- Align password length, complexity and rotation policies with **NIST** 800-63 B's guidelines in section 5.1.1 for Memorized Secrets or other modern, evidence-based password policies.
- Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes.
- Limit or increasingly delay failed login attempts. Log all failures and alert administrators when credential stuffing, brute force, or other attacks are detected.
- Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login. Session IDs should not be in the URL. Ids should also be securely stored and invalidated after logout, idle, and absolute timeouts.

8

Software and Data Integrity Failures

Another new addition to the 2021 roster is software and data integrity failures. These failures can take many forms, particularly since as the web evolves it is more and more common to use third party code and services within web applications. These failures can be summarised as follows:

- Usage of code that does not verify integrity of source
- Usage of third party plugins where you do not control the source
- Plugins and extensions from untrusted sources
- The introduction of or potential for compromise or unauthorised access
- Auto-updates assume trust of the source

WordPress website administrators make heavy usage out of the official WordPress repository. Other CMS platforms have similar libraries that are used. Unlike proprietary software platforms these repositories are all open source and the code is publicly

accessible and able to be scrutinised. This is a major advantage but not foolproof. Many open source plugins over the last few years have been targeted by attackers after serious vulnerabilities were discovered within them.

It is also unfortunately common for website owners to use hacked or "nulled" plugins/themes on their website. These are almost **always** coupled with backdoors that will be used to compromise the website environment.

Examples of Integrity Failure

- In 2021, attackers were able to circumvent Microsoft's verification process and were able to release a digitally **signed** driver infected with rootkit
- The SolarWinds hack **distributed** backdoored software to thousands of organisations including US government and hundreds of major corporations
- In 2016 the website of the popular Linux distribution Linux Mint was hacked and the ISO file **replaced** with backdoored version

How to Prevent Integrity Failure

As our list of examples above indicates, sometimes verifying proper integrity of software is impossible. However, to do our best due diligence the following principles should be adhered to:

- Use software that was digitally signed by a trusted vendor
- Use trusted software repositories, or your own repository
- Verify that your extensions contain no known vulnerabilities
- Verify checksums and file hashes
- Ensure there is a review process for code changes/updates
- Ensure proper access control to ensure data integrity

9

Security Logging & Monitoring Failures

The importance of securing a website cannot be understated. While 100% security is not a realistic goal, there are ways to **keep your website monitored** on a regular basis. This allows you to take immediate action when something happens.
Not having an efficient logging and monitoring process in place can increase the damage of a website compromise.

Here at Sucuri, we highly recommend that every website is properly monitored. If you need to monitor your server, **OSSEC** is freely available to help you. OSSEC actively monitors all aspects of system activity with file integrity monitoring, log monitoring, root check, and process monitoring.

Example of Logging and Monitoring Attack Scenarios

According to OWASP, these are some examples of attack scenarios due to insufficient logging and monitoring:

- **Scenario #1:** An open-source project forum software run by a small team was hacked using a flaw in its software. The attackers managed to wipe out the internal source code repository containing the next version and all of the forum contents. Although the source could be recovered, the lack of monitoring, logging, or alerting led to a far worse breach. The forum software project is no longer active as a result of this issue.
- **Scenario #2:** An attacker scans for users with a common password. They can take over all accounts with this password. For all other users, this scan leaves only one false login behind. After some days, this may be repeated with a different password.
- **Scenario #3:** A major U.S. retailer reportedly had an internal malware analysis sandbox analyzing attachments. The sandbox software had detected potentially unwanted software, but no one responded to this detection. The sandbox had been producing warnings for some time before detecting the breach due to fraudulent card transactions by an external bank.

How to Have Efficient Website Monitoring

Keeping audit logs give visibility to suspicious changes to your website. An audit log is a document that records the events in a website so you can spot anomalies and confirm with the person in charge that the account hasn't been compromised.
Whatever the reason for running out-of-date software on your web application, you can't leave it unprotected. Both Sucuri and OWASP recommend virtual patching for the cases where patching is not possible.

We know that it may be hard for some users to perform audit logs manually. If you have a WordPress website, you can use our free **WordPress Security Plugin** to help you with your audit logs. The plugin can be downloaded from the official WordPress repository.

10

Server-Side Request Forgery

OWASP resource:

SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL).

As cloud services increase in usage and popularity as well as their complexity, the prevalence and risk of SSRF attacks increase too. As cloud services increase in usage and popularity as well as their complexity, the prevalence and risk of SSRF attacks increase too.

How to Prevent SSRF in Web Applications

- Sanitize all user input
- Use a positive allow list rather than a punitive block list
- Do not send raw responses to users/clients
- Disable unencrypted (HTTP) redirections

Example Attack Scenarios

Attackers can use SSRF to attack systems protected behind web application firewalls, firewalls, or network ACLs, using scenarios such as:

Scenario #1 – Port scan internal servers:

If the network architecture is unsegmented, attackers can map out internal networks and determine if ports are open or closed on internal servers from connection results or elapsed time to connect or reject SSRF payload connections.

Scenario #2 – Sensitive data exposure:

Attackers can access local files such as or internal services to gain sensitive information such as file:///etc/passwd and https://localhost:28017/.

Scenario #3 – Access metadata storage of cloud services:

Most cloud providers have metadata storage such as https://169.254.169.254/. An attacker can read the metadata to gain sensitive information.

Scenario #4 – Compromise internal services:

The attacker can abuse internal services to conduct further attacks such as Remote Code Execution (RCE) or Denial of Service (DoS).

Reference

https://sucuri.net/guides/owasp_top_10_2021_edition/#:~:text=OWASP%20Top%20Ten%20is%20the,into%20some%20of%20the%20changes!