OWASP

Top 10 Vulnerabilities 2019

The de facto list of critical threats to your website. Learn what they are and how to protect your website.*

SUCURI

*Based on the latest OWASP Top Ten list from 2017

OWASP stands for the **Open Web Application Security Project**, that produces articles, methodologies, documentation, tools, and technologies in the field of web application security.

OWASP Core Purpose: Be the thriving global community that drives visibility and evolution in the safety and security of the world's software.

The Top 10 OWASP vulnerabilities are

1. Injection

An injection of code happens when an attacker sends invalid data to the web application with the intention to make it do something different from what the application was 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 WHERECustID = '" + request.getParameter("id")
+ "'";
```

This query can be exploited by calling up the web page executing it with the following URL: `http://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 consumed by the web application, which means that this vulnerability can be present on almost any type of technology.

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

Here is another example of an SQL injection that affected over half a million websites. This code is part of the function get_products(). If attackers set arbitrary values for the variable $limit they can modify the query in a way that can lead to a full compromise on some servers.

```
If ( ! empty( $is_default ) ) {
        if( ! empty( $user_id) ) {
$this->generate_defualt_wishlist( $user_iD );
        }
        $sql. = " AND l. `is_default` = %d";
        }
                if (! empty( $id ) ) {
                $sql. = " AND `i.ID` = %d";
                $sql_args [] = $id;
                }
                $sql .= "GROUP BY i.prod_id, L.ID";
                if (!empty ( $limit ) && isset ( $offset ) ) {
                $sql .= " LIMIT ". $offset . " , ". $limit;
        }
        $wishlist = $wpdb-> get_results ($wpdb<prepare ( $sql, $sql_args ), ARRAY_A );
}
```

2. Broken Authentication

A broken authentication vulnerability can allow an attacker to use manual or automatic mediums to try to gain control over a user account – 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.

To avoid broken authentication, don't leave 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.

The second most common form of this flaw is allowing users to brute force username/password combinations against those pages.

Types of Vulnerabilities

However, broken authentication vulnerabilities can come in many forms. According to OWASP, 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.
- 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.

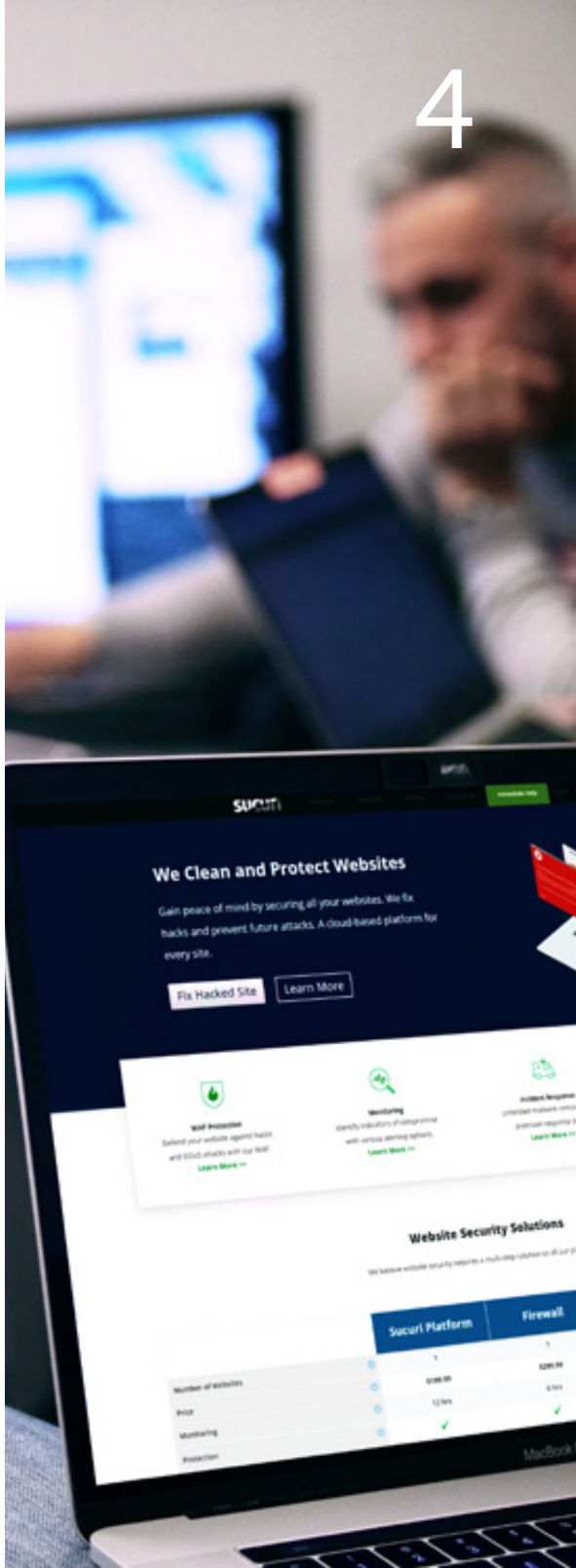
Writing insecure software results in most of the types of broken authentication vulnerabilities. They can be attributed to many factors, like lack of experience from the developers or institutionalized failures such as organizations rushing software releases—in other words, choosing working software over secure software.

How do you prevent broken authentication vulnerabilities?

In order to avoid broken authentication vulnerabilities, make sure the developers apply best practices to website security. Provide access to external security audits and enough time to properly test the code before deploying to production.

OWASP's technical recommendations are the following:

- 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 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. ID's should also be securely stored and invalidated after logout, idle, and absolute timeouts.



3. Sensitive Data Exposure

Sensitive data exposure is one of the most widespread vulnerabilities. It consists of stolen PII (personally identifiable information) data that should have been protected. These are commonly known as data breaches.

Examples of Sensitive Data

Some sensitive data that requires protection is:

- Passwords
- Credit card numbers
- Credentials
- Social Security Numbers
- Health information
- Personally Identifiable Information
- Other personal information

It is vital for any organization to understand the importance of protecting users' information and privacy. All companies should comply with their local privacy laws.

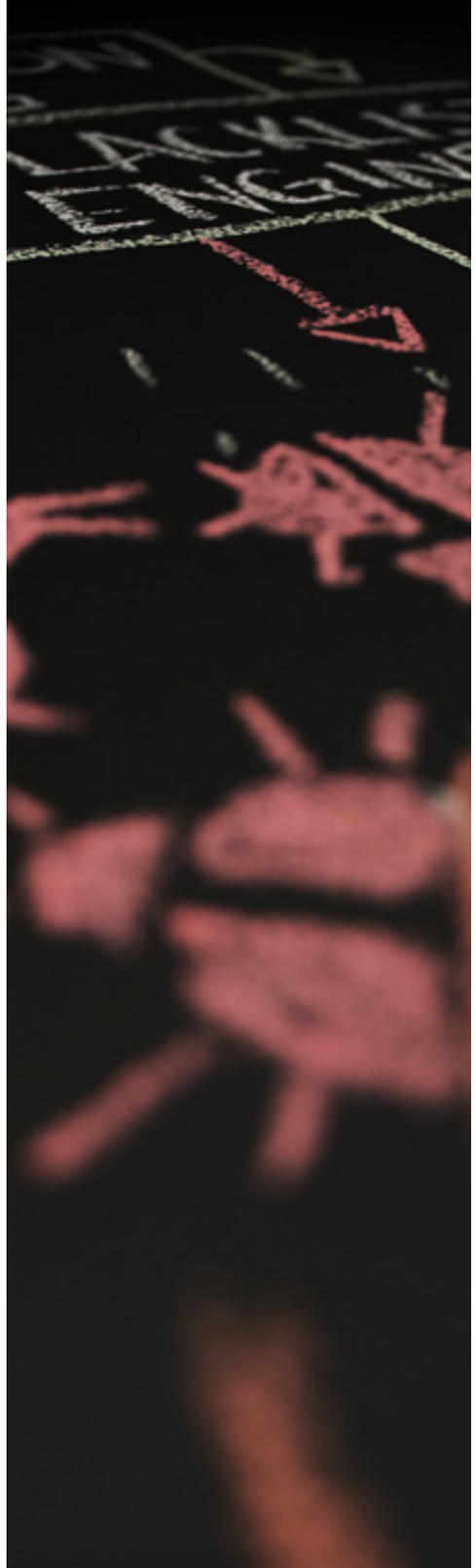
Responsible sensitive data collection and handling have become more noticeable, especially after the advent of the General Data Protection Regulation (GDPR). GDPR is a new landmark data privacy law that came into effect May 2018. Ecommerce websites can reference the PCI DSS requirements to secure cardholder data.

Protecting Data in Transit

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

SSL is the common (but deprecated) name for the TLS protocol, used to establish an encrypted link between a web server and a browser.

We have created a DIY guide to help every website owner install an SSL certificate to their website. You can check out **How to Install an SSL Certificate**.



What Are the Risks?

According to OWASP, here are 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 Sensitive Data Exposure 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 that resulted in sensitive data exposure are:

- The **Brazilian C&A**, a fashion retail clothing chain suffered a gift card platform cyberattack in August 2018.
- The **Uber breach in 2016** that exposed the personal information of 57 million Uber users, as well as 600,000 drivers.
- The **Target store data breach** that occurred around Thanksgiving exposing credit/debit card and contact information of up to 110 million people.

Failure to encrypt sensitive data is the main reason why these attacks are still so widespread. Even encrypted data can be broken due to the following weaknesses:

- Key generation process;
- Key management process;
- Algorithm usage;
- Protocol usage;
- Cipher usage;

This vulnerability is usually very hard to explore; however, the consequences of a successful attack is dreadful. If you want to learn more, we have written a blog post on the **Impacts of the Security Breach**.

How to Prevent Data Exposure

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

- Classify data processed, stored, or transmitted by an application.
- Identify which 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 it as soon as possible or use PCI DSS compliant tokenization or even truncation. Data that is not retained cannot be stolen.
- Make sure to encrypt all sensitive data at rest.
- Ensure 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.

4. XML External Entities (XXE)

According to Wikipedia

> *An* **XML External Entity** *attack is a type of attack against an application that parses* **XML** *input. This attack occurs when* **XML** *input containing a reference to an* **external entity** *is processed by a weakly configured* **XML** *parser.*

Most XML parsers are vulnerable to XXE attacks by default, and an XXE can occur in deeply nested dependencies. That is why the responsibility of ensuring the application does not have this vulnerability lies mostly on the developer

What Are the Attack Vectors?

According to OWASP, the XML external entities (XXE) main attack vectors are:

- Exploitation of vulnerable XML processors if malicious actors can upload XML or include hostile content in an XML document;
- Exploitation of vulnerable code;
- Exploitation of vulnerable dependencies;
- Exploitation of vulnerable integrations.

Example of an XML External Entity Attack

According to OWASP, the easiest way to exploit an XXE is is to upload a malicious XML file.

Scenario #1: *The attacker attempts to extract data from the server:*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

Scenario #2: *An attacker probes the server's private network by changing the above ENTITY line to:*

```
<!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>
```

Scenario #3: *An attacker attempts a denial-of-service attack by including a potentially endless file:*

```
<!ENTITY xxe SYSTEM "file:///dev/random" >]>
```

How to Prevent XML External Entity Attacks

Some of the ways to prevent XML External Entity attacks, according to OWASP are:

- Whenever possible, use less complex data formats such as JSON, and avoid serialization of sensitive data.
- Patch or upgrade all XML processors and libraries in use by the application or on the underlying operating system.
- Use dependency checkers (update SOAP to SOAP 1.2 or higher).
- Disable XML external entity and DTD processing in all XML parsers in the application, as per the OWASP Cheat Sheet 'XXE Prevention'.
- Implement positive ("whitelisting") server-side input validation, filtering, or sanitization to prevent hostile data within XML documents, headers, or nodes.
- Verify that XML or XSL file upload functionality validates incoming XML using XSD validation or similar.
- SAST tools can help detect XXE in source code – although manual code review is the best alternative in large, complex applications with many integrations.

If these controls are not possible, consider using virtual patching, API security gateways, and a **Web Application Firewalls (WAFs)** to detect, monitor, and block XXE attacks.

5. Broken Access Control

In website security, access control means to put a limit on what sections or pages visitors can reach, depending on their needs.

For example, if you own an ecommerce store, you probably need access to the admin panel in order to add new products or to set up a promotion for the upcoming holidays. However, hardly anybody else would need it. Having the rest of your website's visitors be able to reach your login page only opens up your ecommerce store to attacks.

And that's the problem with almost all major content management systems (CMSs) these days. By default, they give worldwide access to the admin panel. Most of them also won't force you to establish a second-factor authentication method (2FA).

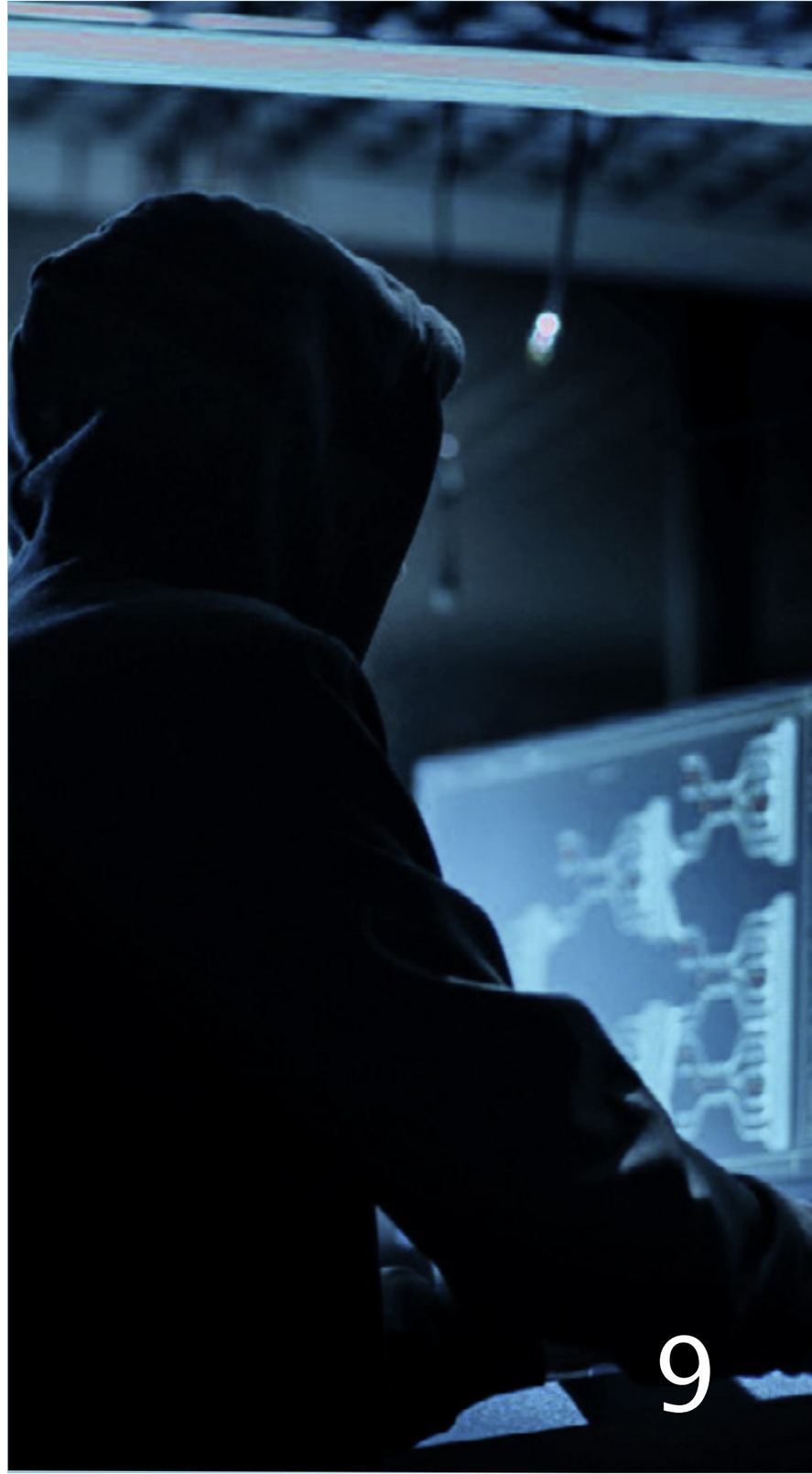
The above makes you look at software development with a security-first philosophy. Access control issues are also present in other areas.

Examples of 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

These are ways attackers can exploit authorization flaws:

- Access unauthorized functionality and/or data;
- View sensitive files;
- Modify other users' data;
- Change access rights, etc.



Broken Access Control

What Are the Risks?

According to OWASP, here are a few examples of what can happen when there is broken access control:

Scenario #1: *The application uses unverified data in a SQL call that is accessing account information:*

```
pstmt.setString(1,request.getParameter("acct")); ResultSetresults
=pstmt.executeQuery( );
```

An attacker simply modifies the 'acct' parameter in the browser to send whatever account number they want. If not properly verified, the attacker can access any user's account.

http://example.com/app/accountInfo?acct=notmyacct

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
http://example.com/app/admin_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 risk of broken access control:

1. Employ **least privileged** concepts – apply a role appropriate to the task and no more.
2. Remove accounts **you don't need**.
3. Audit your servers and websites – who is doing what, when, and why.
4. If possible, apply **multi-factor authentication** to all your access points.
5. Disable access points until needed in order to reduce your access windows.
6. Remove **unnecessary services** off your server.
7. Verify applications that are **externally accessible** versus applications that are tied to your network.
8. 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.

How to Prevent Broken Access Control

With the exception of public resources, deny by default.

Implement access control mechanisms once and reuse them throughout the application, including minimizing CORS usage.

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

- 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 log-ins 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.
- Log access control failures, alert admins when appropriate (e.g. repeated failures). Note: We recommend our free plugin for WordPress websites, that you can download directly from the official WordPress repository.
- Rate limit API and controller access to minimize the harm from automated attack tooling.
- JWT tokens should be invalidated on the server after logout.
- Developers and QA staff should include functional access control unit and integration tests.

6. Security Misconfigurations

Hackers are always looking for ways to penetrate websites, and security misconfigurations can be an easy way in. Here are some examples of things that hackers usually try to exploit in order to gain unauthorized access:

- Unpatched flaws
- Default configurations
- Unused pages
- Unprotected files and directories
- Unnecessary services

One of the most common webmaster flaws is to keep 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 having only default settings.

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

For example, some CMS applications are writeable by the user – allowing a user to install whatever extensions they want.

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?

Security 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 example of application misconfigurations is the memcached servers which, left on default settings, have an open UDP port that attackers used to DDoS huge services in the tech industry. .

Security Misconfigurations

Scenario #1: *The application server comes with sample applications that are not removed from the production server.*
These sample applications have known security flaws 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.*

Scenario #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 Have Secure Installation Systems

According to OWASP, here are some examples of security misconfigurations.

- 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.

7. Cross-Site Scripting (XSS)

Cross Site Scripting (XSS) is a widespread vulnerability that affects many web applications. XSS attacks consist of injecting malicious client-side scripts into a website and using the website as a propagation method.

The danger behind XSS is that it allows an attacker to inject content into a website and modify how it is displayed, forcing a victim's browser to execute the code provided by the attacker while loading the page.

XSS is present in about two-thirds of all applications.

Generally, XSS vulnerabilities require some type of interaction by the user to be triggered, either via social engineering or via a visit to a specific page. If an XSS vulnerability is not patched, it can be very dangerous to any website.

Examples of XSS Vulnerabilities

Imagine you are on your WordPress wp-admin panel adding a new post. If you are using a plugin with a stored XSS vulnerability that is exploited by a hacker, it can force the browser to create a new admin user while in the wp-admin panel or it can edit a post and perform other similar actions.

An XSS vulnerability gives the attacker almost full control of the most important software of computers nowadays: the browsers.

Last year, our research team disclosed a **stored XSS vulnerability in the core of WordPress websites**. Remote attackers could use this vulnerability to deface a random post on a WordPress site and store malicious JavaScript code in it.

Cross-Site Scripting (XSS)

Scenario #1: *The application server comes with sample applications that are not removed from the production server. These sample applications have known security flaws 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.*

Scenario #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.*

Types of XSS

According to OWASP, there are three types of XSS:

XSS Type	Server	Client
Stored	Stored Server	Stored Client
Reflected	Reflected Server	Reflected Client
DOM-Based		Subset of Client

Reflected XSS

OWASP defines the following attack scenarios involving XSS vulnerabilities.

1. The application or API includes unvalidated and unescaped user input as part of HTML output. A successful attack can allow the attacker to execute arbitrary HTML and JavaScript in the victim's browser.
2. Typically the user will need to interact with some malicious link that points to an attacker-controlled page, such as malicious watering hole websites, advertisements, or similar.

Stored XSS

1. The application or API stores unsanitized user input that is viewed at a later time by another user or an administrator. Stored XSS is often considered high or critical risk.

DOM XSS

1. JavaScript frameworks, single-page applications, and APIs that dynamically include attacker-controllable data to a page are vulnerable to DOM XSS. Ideally, the application would not send attacker-controllable data to unsafe JavaScript APIs.
2. Typical XSS attacks include session stealing, account takeover, MFA bypass, DOM-node replacement or defacement (such as Trojan login panels), attacks against the user's browser such as malicious software downloads, keylogging, and other client-side attacks.

8. Insecure Deserialization

In computer science, an object is a data structure; in other words, a way to structure data. s Two key concepts make it easier to understand:

- The process of **serialization** is converting objects to byte strings.
- The process of **deserialization** is converting byte strings to objects.

Example of Attack Scenarios

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

Scenario #1: *A React application calls a set of Spring Boot microservices. Being functional programmers, they tried to ensure that their code is immutable. The solution they came up with is serializing user state and passing it back and forth with each request. An attacker notices the "R00" Java object signature, and uses the Java Serial Killer tool to gain remote code execution on the application server.*

Scenario #2: *A PHP forum uses PHP object serialization to save a "super" cookie, containing the user's user ID, role, password hash, and other state:*

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";
```

```
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

An attacker changes the serialized object to give themselves admin privileges:

```
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";
```

```
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

One of the attack vectors presented by OWASP regarding this security risk was a super cookie containing serialized information about the logged in user. The role of the user was specified in this cookie.

If an attacker is able to deserialize an object successfully, then modify the object to give himself an admin role, they can serialize it again. This set of actions could compromise the whole web application.

How to Prevent Insecure Deserializations

The best way to protect your web application from this type of risk is not to accept serialized objects from untrusted sources.

If you can't do this, OWASP provides more technical recommendations that you (or your developers) can try to implement:

- Implementing integrity checks such as digital signatures on any serialized objects to prevent hostile object creation or data tampering.
- Enforcing strict type constraints during deserialization before object creation as the code typically expects a definable set of classes.
- Bypasses to this technique have been demonstrated, so reliance solely on this is not advisable.
- Isolating and running code that deserializes in low privilege environments when possible.
- Logging deserialization exceptions and failures, such as where the incoming type is not the expected type, or the deserialization throws exceptions.
- Restricting or monitoring incoming and outgoing network connectivity from containers or servers that deserialize.
- Monitoring deserialization, alerting if a user deserializes constantly.

9. Using Components with Known Vulnerabilities

These days, even simple websites such as personal blogs have a lot of dependencies.

We can all agree that failing to update every piece of software on the backend and frontend of a website will, without a doubt, introduce heavy security risks sooner rather than later.

For example, **our hacked website report for 2018** has a dedicated section around outdated CMSs. This report shows that at the time of the infection:

- 36.7% of infected WordPress websites were out of date.
- 50% of infected Modx websites were out of date.
- 63.1% of infected Drupal websites were out of date.
- 72.6% of infected phpBB websites were out of date.
- 83.1% of infected Magento websites were out of date.
- 87.5% of infected Joomla! websites were out of date.
- 91.3% of infected OpenCart websites were out of date.
- 97.2% of infected PrestaShop websites were out of date.

The question is, why aren't we updating our 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.

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

Whatever the reason for running out-of-date software on your web application is, 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.

Vulnerable Applications

Vulnerable applications are usually outdated, according to OWASP 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 blog feed** to stay on top of website security issues.

How to Prevent Using Components with Known Vulnerabilities.

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

- Remove all unnecessary dependencies.
- Have 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.
- Obtain components only from official sources.
- Get rid of components not actively maintained.
- Use virtual patching.

10. Insufficient Logging and Monitoring

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 so you can take immediate action when something happens.

Not having an efficient logging and monitoring process in place can increase the chances 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 Attack Scenarios

According to OWASP, here 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 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 are vital to staying on top of any suspicious change 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.

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 Security Plugin** which can be downloaded from the official WordPress repository.

The Sucuri Website Security Platform has a comprehensive **monitoring solution** that includes:

1. Remote scanner
2. Website blacklist scanner
3. Server-side scanner
4. DNS scanner
5. SSL scanner
6. Uptime scanner

If you are looking for a security solution for your website, check out our comprehensive **Website Security Platform**.

1–888-873-0817 | sales@sucuri.net | www.sucuri.net

Sucuri helps organizations that want to ensure the integrity and availability of their websites. Unlike other website security systems, Sucuri offers a cloud-based WAF along with monitoring and response services for compromised sites, excellent customer service, and a deep passion for research.