

sucuri.net

OWASP Top 10 Security Vulnerabilities 2020

39-50 minutes

***Note:** OWASP expects to complete the next major update of its Top Ten project sometime this year. And it's considering a number of new contenders that have risen in prominence over the past 3-4 years. Follow us here for an update as soon as OWASP Top Ten 2021 officially drops. As of our post date, OWASP recently closed its call for input from the application security industry – hopefully indicating the new report will be coming soon.”.*

When managing a website, it's important to stay on top of the most critical security risks and vulnerabilities. The OWASP Top 10 is a great starting point to bring awareness to the biggest threats to websites in 2021.

What is OWASP?

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

What is the OWASP Top 10?

OWASP Top 10 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 in 2017. Let's dive into it!

The Top 10 OWASP vulnerabilities in 2021 are:

- Injection
- Broken authentication
- Sensitive data exposure
- XML external entities (XXE)

- Broken access control
- Security misconfigurations
- Cross site scripting (XSS)
- Insecure deserialization
- Using components with known vulnerabilities
- Insufficient logging and monitoring

[Stop OWASP Top 10 Vulnerabilities](#)

Injection

A code injection happens when an attacker sends invalid data to the web application with the intention to make it do something that the application was 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") + "";
```

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 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 potentially be vulnerable to a code injection attack.

We've written a lot about code injection attacks. One of the most recent examples is the [SQL injection vulnerability in Joomla! 3.7](#).

Here is another example of an SQL injection that affected over half a million websites that had the [YITH WooCommerce Wishlist](#) plugin for WordPress:

```
if( ! empty( $is_default ) ){  
    if( ! empty( $user_id ) ){  
        $this->generate_default_wishlist( $user_id );  
    }  
    $sql .= " AND l.`is_default` = %d";  
    $sql_args[] = $is_default;  
}  
  
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 );  
}
```

The SQL injection shown above could cause a leak of sensitive data and compromise an entire WordPress installation.

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 keeping it to a minimum of plugin 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 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, such as 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, column names, and so on 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 abstract 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 measure 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.

Broken Authentication

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 broken authentication 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.

The second most common form of this flaw is allowing users to [brute force](#) username/password combination against those pages.

Types of Broken Authentication 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.
- 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 broken authentication vulnerabilities?

In order to avoid broken authentication vulnerabilities, 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.

Sensitive Data Exposure

Sensitive data exposure is one of the most widespread vulnerabilities on the OWASP list. It consists of compromising data that should have been protected.

Examples of Sensitive Data

Some sensitive data that requires protection is:

- 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 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). This is a new data privacy law that came into effect May 2018. It mandates how companies collect, modify, process, store, and delete 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 thinking about 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 on [how to install an SSL certificate](#).

What are the risks of sensitive data exposure?

According to the [OWASP Top 10](#), 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 ended up in exposing sensitive data are:

- The [Brazilian C&A](#) retail fashion retail clothing

chain gift card platform cyberattack that happened 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 information and contact information of up to 110 million people.

Not encrypting sensitive data is the main reason why these attacks are still so widespread. Even encrypted data can be broken due to weak:

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

This vulnerability is usually very hard to exploit; however, the consequences of a successful attack are dreadful. If you want to learn more, 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 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.

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. That is why the responsibility of ensuring the application does not have this vulnerability lays mainly on the developer.

What are the XML external entity attack vectors?

According to the OWASP Top 10, the XML external entities (XXE) main attack vectors include the exploitation of:

- Vulnerable XML processors if malicious actors can upload XML or include hostile content in an XML document
- Vulnerable code
- Vulnerable dependencies
- Vulnerable integrations

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 (“allowlisting”) 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
- [Web Application Firewalls \(WAFs\)](#) to detect, monitor, and block XXE attacks

Broken Access Control

In website security, access control means putting 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. Allowing the rest of your website's visitors 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 (CMS) these days. By default, they give worldwide access to the admin login page. Most of them also won't force you to establish a two-factor authentication method (2FA).

The above makes you think a lot about software

development with a security-first philosophy.

Examples of Broken Access Control

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 the following:

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

What are the risks of broken access control?

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 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 and no more.
- Get rid of accounts you don't need or whose user no longer requires them.
- Audit your servers and websites – who is doing what, when, and why.
- If possible, apply multi-factor authentication to all your access points.
- Disable access points until they are needed in order to reduce your access windows.
- Remove unnecessary services off 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 is to develop and configure software with a security-first philosophy. That's why 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.
- 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 units and integration tests.

Security Misconfigurations

At its core, brute force is the act of trying many possible combinations, but there are many variants of this attack to increase its success rate. Here are the most common:

- Unpatched flaws

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

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 Attack Scenarios

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.

- **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

In order to prevent 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.

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 risks 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 your browser to create a new admin user while you're 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.

Back in 2017, 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.

Types of XSS

According to the OWASP Top 10, there are three types of cross-site scripting:

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

- **Reflected XSS:** 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. 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:** 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:** 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. 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.

Reducing the Risks of XSS

There are technologies like the [Sucuri Firewall](#) designed to help mitigate XSS attacks. If you are a developer, here is some insight on how to identify and account for these weaknesses.

How to Prevent XSS Vulnerabilities

Preventive measures to reduce the chances of XSS attacks should take into account the separation of untrusted data from active browser content. OWASP guidelines gives some practical

tips on how to achieve it:

- Using frameworks that automatically escape XSS by design, such as the latest Ruby on Rails, React JS. Learn the limitations of each framework's XSS protection and appropriately handle the use cases which are not covered.
- Escaping untrusted HTTP request data based on the context in the HTML output (body, attribute, JavaScript, CSS, or URL) will resolve Reflected and Stored XSS vulnerabilities. The [OWASP Cheat Sheet for XSS Prevention](#) has details on the required data escaping techniques.
- Applying context-sensitive encoding when modifying the browser document on the client side acts against DOM XSS. When this cannot be avoided, similar context-sensitive escaping techniques can be applied to browser APIs as described in the [OWASP Cheat Sheet for DOM based XSS Prevention](#).
- Enabling a [content security policy \(CSP\)](#) is a defense-in-depth mitigating control against XSS. It is effective if no other vulnerabilities exist that would allow placing malicious code via local file

includes (e.g. path traversal overwrites or vulnerable libraries from permitted content delivery networks).

Insecure Deserialization

Note: The OWASP Top 10 noted that this security risk was added by an industry survey and not based on quantifiable data research.

Every web developer needs to make peace with the fact that attackers/security researchers are going to try to play with everything that interacts with their application—from the URLs to serialized objects.

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

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

Examples of Insecure Deserialization Attack

Scenarios

According to OWASP guidelines, 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, [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 security 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.

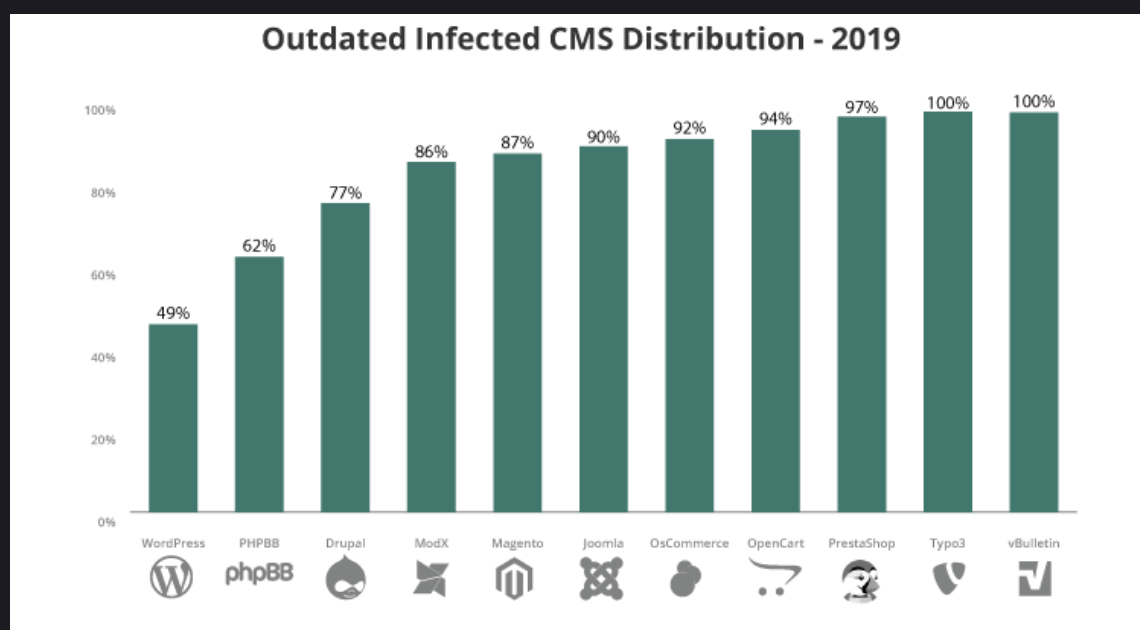
Using Components with Known Vulnerabilities

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, in 2019, [56% of all CMS applications were out of date](#) at the point of infection.



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.
- Webmasters are scared that something will break

on their website.

- Webmasters don't have the expertise to properly apply the update.

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. 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.

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 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.
- 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 website 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.
- 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 with the help of a [Website Application Firewall](#).

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 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 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.

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.

Vulnerable Applications

The Sucuri Website Security Platform has a comprehensive [website monitoring solution](#) that includes:

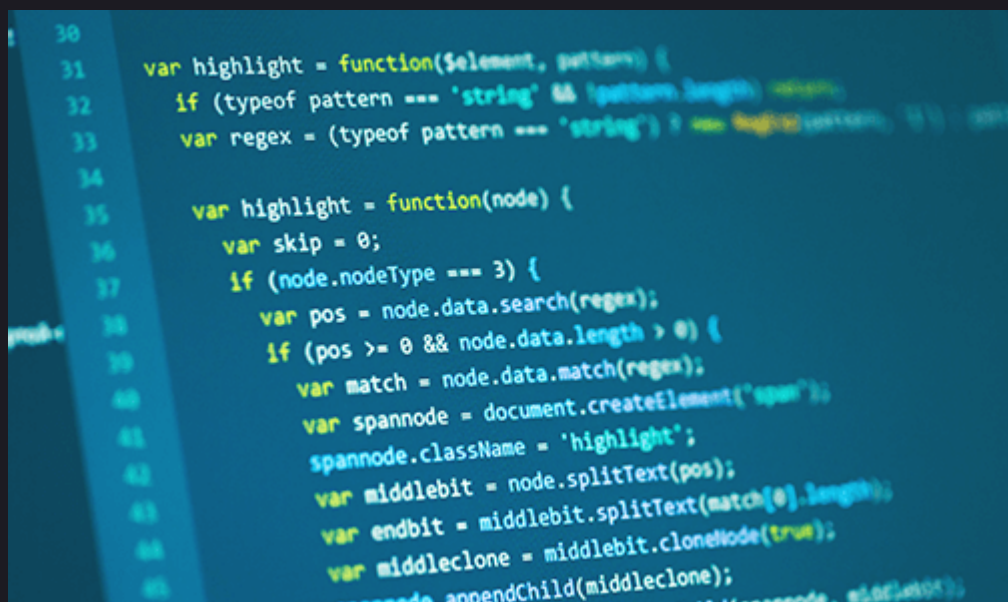
- Remote monitoring

- Website blocklist monitoring
- Server-side monitoring
- DNS monitoring
- Uptime monitoring

Protect Your Website from Security Vulnerabilities

The Sucuri Website Security Platform can protect your site from the top 10 website threats and security risks. [Sign up](#) to have peace of mind.

Additional Resources



```
30
31 var highlight = function($element, pattern) {
32   if (typeof pattern === 'string' && (pattern.length > 0)) {
33     var regex = (typeof pattern === 'string') ? new RegExp(pattern, 'gi') : pattern;
34
35     var highlight = function(node) {
36       var skip = 0;
37       if (node.nodeType === 3) {
38         var pos = node.data.search(regex);
39         if (pos >= 0 && node.data.length > 0) {
40           var match = node.data.match(regex);
41           var spannode = document.createElement("span");
42           spannode.className = 'highlight';
43           var middlebit = node.splitText(pos);
44           var endbit = middlebit.splitText(match[0].length);
45           var middleclone = middlebit.cloneNode(true);
46           spannode.appendChild(middleclone);
47           endbit.parentNode.insertBefore(spannode, endbit);
48         }
49       }
50     };
51     $element.each(function() { highlight(this); });
52   }
53 }
```

Learn how to identify issues if you suspect your WordPress site has been hacked.

[Watch Now](#)