Snyk Top 10

Code Vulnerabilities in 2022

Staying current on the most prevalent vulnerabilities is vital to creating and maintaining application security. We have the OWASP Top 10 to keep us in the loop for open source security, but knowing how to manage risk in our proprietary is also critical. So, we consulted with our team of Snyk Security Researchers — the same folks responsible for our cutting-edge machine learning and hybrid AI — to list the top 10 code vulnerabilities they encountered in 2022.

This ranking is an aggregate of the ten most common vulnerability types across seven popular languages — JavaScript, Java, Python, Go, PHP, Ruby, and C#. Each vulnerability type presents unique risks and is important to look for. However, their prevalence in your projects will depend on language, coding guidelines, etc. If you're interested in more exact rankings, we've also compiled a cheat sheet for each of the languages listed above. The cheat sheets rank the ten most common vulnerability types by occurrence, with links to a relevant CWE page or Snyk Learn lesson for more information.

1. Directory Traversal

A directory traversal (a.k.a. path traversal) attack aims to access files and directories that are stored outside the intended folder. Manipulating files with "dot-dot-slash (../)" sequences, or absolute file paths, can provide access to arbitrary files and directories stored on the file system.

Take, for example, a picture in HTML code format:

```
<img src="/loadImage?file=images/logo_pic.png"/>
```

Since the server would search the hypothetical image in the /var/www/images/ directory, an attacker could try to call the loadImage to unveil other information like this:

```
https://yourweb.server/loadImage?file=../../etc/ssl/certs/privateKey.key
```

Two possible scenarios result from directory traversal — information disclosure or file overwrites. Information disclosure occurs when data from files that should be inaccessible to the users are read and often leaked. This can give malicious users critical information on everything from application setup and configuration to API tokens.

The other result might be to write or overwrite an existing file to crash the system, change its behavior, or reprogram it. For example, even if an attacker can only transport a few characters into a system, adding these characters to a script file and executing the script creates an opportunity to gain wider access.

To prevent this, the path must be sanitized whenever file access is performed using an externally provided path (even a partial one). Every web framework worth its salt provides functions to generate something called the canonical path or the normalized path that no longer contains relative path elements like ellipses (...). Ensuring the canonical path points to a desired directory is also important.

2. Cross-Site Scripting

Cross-site scripting (or XSS) is a website attack method that utilizes an injection to implant malicious scripts into websites that would otherwise be productive and trusted. Generally, the process involves sending a malicious browser-side script to another user.

A script can be injected into the browser and executed in the user context. This allows a malicious user to change how the website is displayed to the user, changing the content to reflect different “facts.”

An application is vulnerable to XSS whenever external data sources are used to gather the displayed browser text. It’s also important to consider data sources like GitHub, where project descriptions might contain an unwanted script.
In addition to web framework functions designed to sanitize the output, HTML and JavaScript offer functionality to flag text in websites. Check out the official OWASP cheatsheet for more information.

3. Use of Hardcoded Credentials

Hardcoded credentials are used for inbound authentication, outbound communication to external components, and encryption of internal data. Credentials are called hard-coded when they are written directly in the code. This system allows everyone with access to the source code to access the credentials — making it difficult to change potentially infringed credentials as they might be distributed over several places in the code and require a redeploy.

As a best practice, never use hardcoded credentials. Make use of system functions or services to provide credentials when needed. Do not store or cache credentials.

4. Open Redirect

An open redirect vulnerability occurs when an application allows a user to control a redirect or forward it to another URL. If untrusted user input isn't validated, an attacker could supply a URL redirecting an unsuspecting victim from a legitimate domain to an attacker's phishing site. This can lead to phishing, token theft, SSRF, or XSS attacks.

Open redirects are especially nasty when embedded into a normal workflow, as most users only check for phishing when starting a workflow — not in the middle of one. In many social media applications, you see an intermediate screen informing the user that they are leaving the social media site to follow a link.

As a best practice, websites should make it obvious from a user experience point of view when the user is redirected to another location. Programmatic redirects should ensure the application provides the server name, and user-provided data should be parsed with a framework function to sanitize URLs and prevent unwanted URIs.

5. Insecure Hash

An insecure hash vulnerability is a failure related to cryptography. Hashed are often used to store and check secrets like passwords for sign-in messages or blockchain applications. However, some hash functions sufficient for certain aspects of the app, like hash tables, are no longer seen as cryptographically secure. In security, hashes as one-way functions should be next to impossible to invert — aka using the hash value to find the original data.

The best practice is always to use security-rated functions in cryptography. This starts with random numbers. Avoid relying on simple random number generators for cryptographic applications. Instead, apply hard hashes like NIST suggests and ensure you use sufficient key lengths.

6. Cross-Site Request Forgery (CSRF)

Cross-site request forgery (CSRF) is a vulnerability where an attacker performs actions while impersonating another user by using the fact that browsers rely on automatic identification and authorization using cookies, tokens, or a Windows login. A malicious actor could use a CSRF attack to transfer funds from a victim's account to their own, change a victim's email address, or even redirect a pizza!

A common vector in this vulnerability type is phishing attacks containing a simple-looking hyperlink that triggers major changes or activities. To prevent this from an application perspective, it's important to guardrail critical functions and data disclosure with additional authentication that doesn't rely on automatic mechanisms provided by the web infrastructure.

7. SQL Injection

SQL injection is a common method attackers use to manipulate and access database information. This is done by exploiting application vulnerabilities to inject malicious SQL code that alters SQL queries.

You need to be extra cautious whenever external data can cross the line into a command. In the case of SQL, the need to formulate queries using external data might be necessary. But this data can also be used to alter the query in itself and the attacker-injected SQL.

A common best practice is to encapsulate external data and use parameterized queries to inform the SQL that this part of the query is data and contains no script code. Another point is not to trust the data coming back from a client machine, like IDs assigned to buttons on a website, and never underestimate where this data can stem from.

8. Sensitive Cookie Without 'HttpOnly' Flag

A sensitive cookie without 'HttpOnly' vulnerability occurs when a cookie not marked with the HttpOnly flag is used to store sensitive information. The HttpOnly flag directs compatible browsers to prevent client-side scripts from accessing cookies, which helps prevent data leaks.

Whenever data stored in cookies is needed within local JavaScript, the HttpOnly flag prevents the local script from accessing the data and fails. Additionally, this flag alone cannot guarantee that the communication is encrypted using SSL (see cleartext transmission).

9. Cleartext Transmission of ` Sensitive Information

Cleartext transmission occurs when software transmits sensitive or security-critical data via cleartext in a channel that can be sniffed by unauthorized actors, significantly lowering the difficulty of exploitation by attackers.

Sensitive data should be encrypted in transit and rest. Modern systems provide all the necessary cryptographic facilities to ensure high trust — but they must be used correctly. A major first step is to list all the data that needs to be protected and in which manner. For example, traffic between a client and server might include personal information and should be encrypted from eavesdropping. Authentication tokens might be even more critical and prevented from accessing users directly.

Using encrypted channels can be enforced partly by the application by requesting HTTPS rather than HTTP channels. But the infrastructure must play a major enablement role and provide valid certificates.

10. Improper Certificate Validation

Improper certificate validation occurs when a certificate is incorrectly validated or not validated at all. When a certificate is invalid or malicious, an attacker might spoof a trusted entity by interfering with communication between the host and the client.

Using cryptography relies a great deal on proper key exchange and management. Keys are exchanged through certificates. These certificates have several aspects that might render them invalid — such as the valid date, the role of the base certification authority, or the certificate revocation list showing no longer valid certificates. It is a best practice to ask the system to validate certificates before using the keys. This process might take some time (e.g., the machine needs to download the revocation list), but it enforces the benefits of certificates. It is also advisable to consider the requirements for these certificates — key length, algorithms used, and possible issuing certification authority can be checked before accepting the certificate.

Stay security aware

While this report has given you a look at the ten most prevalent high/critical code vulnerabilities our security researchers encountered in 2022, the list expands considerably once all vulnerability types are included (e.g., medium/low). To keep these vulnerabilities out of your code, we recommend the following:

- **Shift security left:** The sooner you catch a vulnerability, the easier it is to fix.
- **Implement developer security tooling:** Implement a tool (like Snyk) that can find vulnerabilities in real-time and offer developers contextual prioritization and simple fixes (instead of a link to documentation).
- **Cover your full application:** Proprietary and third-party code vulnerabilities are as dangerous as misconfigurations in your IaC or cloud configs. Be sure to secure all aspects of your application.
- **Automate, automate, automate:** Automate scans and fixes as much as possible to make security a paved path.

Keep learning: Stay ahead of attackers by keeping your security skills fresh. Check out Snyk Learn for free, interactive lessons on vulnerabilities across various languages and ecosystems.

Keep vulnerabilities out!

Find out how simple Snyk makes it to find and fix vulnerabilities in your proprietary code and across your entire SDLC.

BOOK A DEMO