The No-BS Guide to Defending Your Applications Against the OWASP Top 10 2025

The ten most common security vulnerabilities don't stand a chance against sharp, security-focused developers like you. This is your ultimate field guide to understanding each infamous entry in the **OWASP Top 10**, gaining insight into how each bug and vulnerability category operates.

You'll see why they're so dangerous, and most importantly, how you can **banish every one of them from your software forever.**

SECURE CODE WARRIOR

securecodewarrior.com

Index

You have received a series of missions from Secure Code Warrior. Learn how your targets work, understand their motives, and finally, destroy them. Don't forget to check out the walkthrough videos after reading about each vulnerability.

GOOD LUCK, NEW WARRIOR

1 Broken Access Control

When you build a business application, whether for internal use or external use by your customers, you probably don't let every user perform every single function. If you do you may be vulnerable to broken access control.

Business applications have a rich set of functions, sometimes up to hundreds. However each of these functions should not be used by every single user in the system.

Let's take a look at broken access control is, why it's so dangerous, and how to fix it.

Watch Video

Understand Broken Access Control

Broken access control occurs when application code does not have the proper security or access checks in place. It can also occur when an application is misconfigured in some way that allows access to functions or pages to which the user should not have access.

If you handle the finances of your company you may have access to deposit money into certain accounts or transfer money between your company's accounts. However, you shouldn't have access to withdraw money from those accounts or transfer money to accounts outside of your company's control. If the proper access checks are not present, then your employees may be able to do more functions than necessary.

These checks can either be done within the code itself or some configuration files. For example, there may be XML configuration files which tells the web application framework which users are allowed to access which pages. This ensures that only the right people are seeing the right functions.

Why Broken Access Control is Dangerous

Consider an example. An attacker has realized that your user account creation code can be manipulated to allow the attacker to create an admin user with a simple post request. They can send a request with the username and password and then change the request on route to include the role of admin in the URL as a parameter or in the body of the request. The attacker logs into the application and is instantly given administrator rights.

It doesn't always have to be a malicious attacker attacking a system. Without proper access controls, sensitive information that shouldn't be shared between departments may leak out. Imagine if any employee in the company could see HR payroll data or financial data. What would happen if any employee could see that layoffs are coming because of the poor financial situation of the company? This could be damaging to your morale and your company's reputation.

Sensitive information from the customers could also be lost. Companies in the healthcare industry often have personal health information of customers that use their services. Be careful not to accidentally expose personal information because of a lack of access control.

For example, if your system gives users the ability to request a record of their health information, do they also have the ability to request and see the health information of others? If the URL contains a customer ID number, attackers could increment that customer ID number over and over again until they find one that matches another customer, thus revealing their personal data.

Defeat Broken Access Control

Role-based access control (RBAC) is a very effective tool for implementing sound access control. Those using Active Directory may be familiar with the idea of creating groups and giving access to certain items to the group instead of the individual. Applications work the same way, using roles to define who is allowed to see what.

This has two advantages. First, a function doesn't have to be changed when somebody leaves the administrator role. If somebody previously was an administrator and no longer is then you simply place a new person into the administrator role and remove the previous person from the role. The code checks to see if the user has the administrator role instead of checking to see if each individual user has access to a certain page or function.

The second benefit is avoiding a maintenance nightmare. Access control that is so granular that every person has associations with every single possible function or page will be impossible to manage over time. Roles make things easier because multiple people can be added to a role. One role may have the entire company while another role has only five people. This makes managing the roles much easier because there will be fewer roles to manage. A company of 10,000 people could have only 100 roles instead of 10,000 times the number of functions in your application. Research your chosen application framework to see what options exist for robust access control.

Protect Your Sensitive Functions

Broken access control can leave your data and your application wide open for attack and exploitation. Customer data that is not protected properly could lead to a massive data breach, hurting your reputation and your revenue.

Broken access control could also lead to account takeover if attackers are able to access functionality they shouldn't access. Use proper functional level access control and you'll keep your application safe from malicious attackers and even accidental insiders. Then, you'll know that your data and your functions are safe and secure.

A Note on Server-Side Request Forgery

In the OWASP Top 10 2021 list, Server-Side Request Forgery (SSRF) was a category unto itself, ranking tenth. Now folded into the Broken Access Control class as a prominent Common Weakness Enumeration (CWE), it represents a significant vulnerability that can be complex to solve without direct guidance, so please check out the guide below and keep practicing.

The goal of many SSRF attacks is to cause major disruption, or to come away with some serious paydirt in the form of leaking information and stealing valuable data. This vulnerability is dangerous, and the consequences of successfully compromising HTTP servers are far-reaching. Alarmingly, the nature of this particular attack type means it can bypass standard access control methods like VPNs and firewalls. With the rapid adoption of cloud-based services, it has the potential to seriously infect software infrastructure and cause serious headaches for the security team, not to mention customers and users.

Good news, though. With the right knowledge, this insidious pest can be snuffed out before it even has a chance to open doors for threat actors looking to cause trouble.

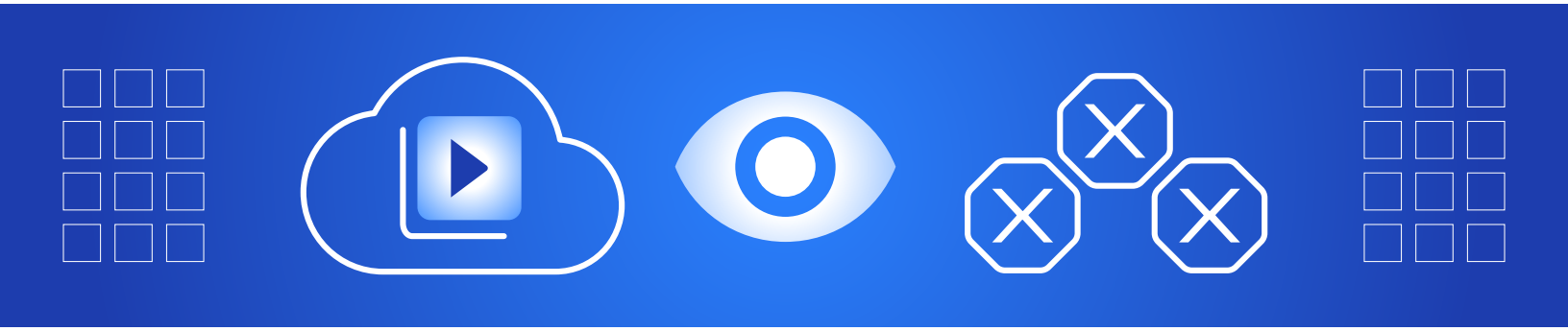
To that end, we'll discuss three key aspects of SSRF attacks:

- How they work
- Why they are so dangerous
- How you can put defenses in place to stop them cold

How Do SSRF Attacks Work?

Essentially, a successful SSRF attack allows a threat actor to trick a server into performing requests on their behalf. These forgeries typically open the door for things like port scanning, file retrieval, and access to internal services that were never supposed to be visible to those outside of the organization.

This vulnerability can be exploited as a result of an application not restricting the type of resources it can access by location, or file type. For example, let's say a production studio has a website containing showreels and samples of their work. Inevitably, they'd like potential clients to see high-resolution images and video, and make them available to view. An attacker notices that the file paths point to an external, separate server, and is able to guess the studio is utilizing cloud services to deal with the huge file sizes associated with high-res video. They could then attempt to request the instances' metadata and user data. These URLs are supposed to be inaccessible, and if discovered, result in sensitive data exposure like access tokens and public keys, not to mention potential data relating to individual users.



Our hypothetical production studio has a subdomain, watch.vulnerablestudio.com, that uses multiple servers. Potential clients can visit watch.vulnerablestudio.com to browse through their portfolio, but the full, high-res videos sit on another server, to which browsing users don't have direct access. However, it can be assumed that the server behind watch.vulnerablestudio.com does.

When you click on a project, the studio's main server makes a secondary request. We'll go ahead and use a common GET request as an example, with the URL visible in your browser's address bar: watch.vulnerablestudio.com/?url=video-storage.vulnerablestudio.com/ZombieSurfers/.

Going to the URL in full provides you with the video and details of the "Zombie Surfers" project. If an attacker can work out that they don't have direct access to http://video-storage.vulnerablestudio.com/ZombieSurfers/, but watch.vulnerablestudio.com does, they have now found a way to use the studio's main server as neutral territory. They can potentially breach this webserver and access restricted assets and data.

If watch.vulnerablestudio.com has been configured with lax security controls and is vulnerable to SSRF, an attacker could simply navigate to watch.vulnerablestudio.com/?url=file:///etc/passwd, and it's highly likely the vulnerable server now returns the sensitive and restricted contents of the /etc/passwd file if it's poorly configured and insecure.

Why is SSRF Dangerous?

Depending on the server and how it's configured, the attacker could enumerate ports to discover all services on the localhost address, use file:// or smb:// to request files on these internal directories, and download files from internal FTP servers. Sadly, that's just the tip of the iceberg. Aside from exposure and theft of sensitive data, successful SSRF can allow abuse of internal services to conduct further attacks, such as Remote Code Execution (RCE) or Denial of Service (DoS). All scenarios are reputational poison, highly disruptive, and in the age of GDPR and more focus on data privacy and safety than ever before, potentially financially devastating.

How Do I Stop SSRF Attacks?

It is imperative that developers are keeping security front-of-mind in their day-to-day processes, and best practices would dictate that the potential attack surface is kept as small as possible. In the case of squashing SSRF, this means employing a zero-trust approach to access control, leaving no room for APIs that are too talkative, account levels with unnecessary access, or the potential for end-users to manipulate inputs. This can take the form of:

- Comprehensive whitelisting; restrict requests made by the server to whitelisted locations wherever possible.
- Verifying the requested file type matches that which is expected.
- Displaying a generic error message in the event of failure.
- Restrict requests to only approved URL schemas.

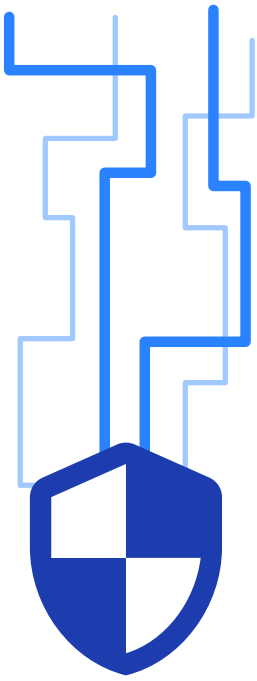
Slamming the Door on SSRF

With such a high potential payout and applications becoming more complex and integrated by the day, SSRF attacks will likely never perish entirely, but we hope you learned why they are so persistent, and how to block them from your network for good.

For further reading, you can take a look at the OWASP Server-Side Request Forgery **Prevention Cheat Sheet** which serves as a living document chronicling this vulnerability as it evolves.

2 Security Misconfiguration

The term security misconfiguration is a bit of a catchall that includes common vulnerabilities introduced due to the application's configuration settings, instead of bad code. The most common ones normally involve simple mistakes that can have big consequences for organizations that deploy apps with those misconfigurations.



Some of the most common security misconfigurations include not disabling debugging processes on apps before deploying them to the production environment, not letting applications automatically update with the latest patches, forgetting to disable default features, as well as a host of other little things that can spell big trouble down the road.

The best way to combat security misconfiguration vulnerabilities is to eliminate them from your network before they are deployed to the production environment.

In this chapter, we will learn:

- How hackers find and exploit common security misconfigurations
- Why security misconfigurations can be dangerous
- Policies and techniques that can be employed to find and fix security misconfigurations

How Do Attackers Exploit Common Security Misconfigurations

There are a lot of common security misconfigurations. The most popular ones are well-known in hacker communities and are almost always searched for when looking for vulnerabilities. Some of the most common misconfigurations include, but are not limited to:

- Not disabling default accounts with well-known passwords.
- Leaving debugging features turned on in production that reveal stack traces or other error messages to users.
- Unnecessary or default features left enabled, such as unnecessary ports, services, pages, accounts, or privileges.
- Not using security headers, or, using insecure values for them.

Some misconfigurations are well-known and trivial to exploit. For example, if a default password is enabled, an attacker would only need to enter that along with the default username to gain high-level access to a system.

Other misconfigurations require a bit more work, such as when debugging features are left enabled after an app is deployed. In that case, an attacker tries to trigger an error, and records the returned information. Armed with that data, they can launch highly targeted attacks that may expose information about the system or the location of data they are trying to steal.

Why is Security Misconfiguration so Dangerous?

Depending on the exact security misconfiguration being exploited, the damage can range from information exposure to complete application or server compromise. Any security misconfiguration provides a hole in defenses that skilled attackers can leverage. For some vulnerabilities, such as having default passwords enabled, even an inexperienced hacker can exploit them. After all, it doesn't take a genius to look up default passwords and enter them!

Removing the Threat Posed by Security Misconfigurations

The best way to avoid security misconfigurations is to define secure settings for all apps and programs being deployed across an organization. This should include things like disabling unnecessary ports, removing default programs and features not used by the app, and disabling or changing all default users and passwords. It should also include checking for and dealing with common misconfigurations, such as always disabling debugging mode on software before it hits the production environment.

Once those are defined, a process should be put in place, one that all apps go through before they are deployed. Ideally, someone should be put in charge of this process, given sufficient power to enforce it, and also responsibility should a common security misconfiguration slip through.

More Information About Security Misconfigurations

For further reading, you can take a look at the OWASP list of the **most common** security misconfigurations. You can also put your newfound defensive knowledge to the test with a **free demo** of the Secure Code Warrior platform, which trains cybersecurity teams to become the ultimate cyber warriors.

3 Software Supply Chain Failures

With the much-anticipated arrival of the **2025 OWASP Top Ten**, enterprises have a couple of new threats to be extra wary of, including one that lurks near the top of the list. **Software Supply Chain Failures**, which debuts as a new category but isn't entirely new, sits at No. 3 on the Open Web Application Security Project's quadrennial list of the most serious risks to web application security. It's a risk that enterprises must take very seriously, if they aren't already.

Software Supply Chain Failures grew out of a category in the previous list from 2021, **Vulnerable and Outdated Components**, and now it includes a broader range of compromises across the software ecosystem of dependencies, build systems and distribution infrastructure. And its appearance on the list should come as no particular surprise, given the damage caused by high-profile supply chain attacks such as **SolarWinds** in 2019, the **Bybit hack** earlier this year, and the ongoing **Shai-Hulud campaign**, a particularly nasty, self-replicating npm worm wreaking havoc on exposed developer environments.

The OWASP Top Ten has generally been consistent, which befits a list that appears every four years, albeit with updates in between. There usually is some shuffling within the list—Injection, a longtime resident, drops from No. 3 to No. 5, for instance, and Insecure Design drops two places to No. 6, while Security Misconfiguration jumps from No. 5 to No. 2. Broken Access Control continues to stake out the top position. The 2025 edition has two new entries, the aforementioned Software Supply Chain Failures and Mishandling of Exceptional Conditions, which enters the list at No. 10.

Here, we take a close look at the new supply chain vulnerabilities entry.

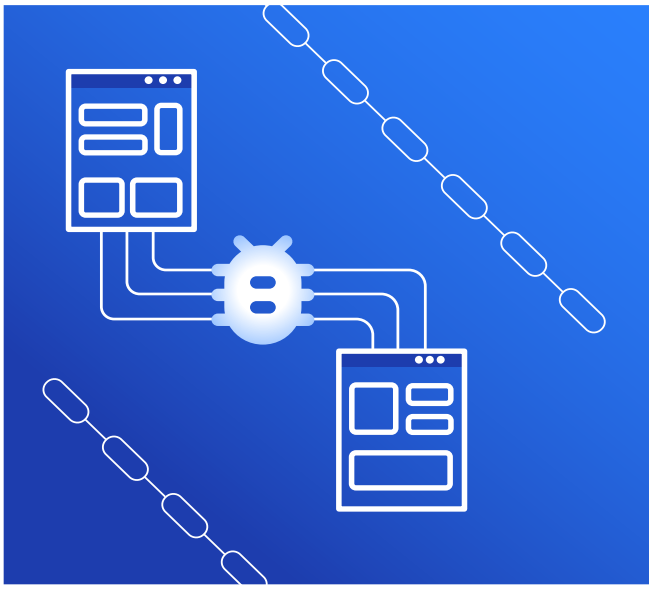
Vulnerabilities Can Crop Up Almost Anywhere

Software Supply Chain Failures is a somewhat unusual category on the list in that, among the 10 entries, it has the fewest occurrences in OWASP's research data, but it also had the highest average exploit and impact scores resulting from the five Common Weakness Enumerations (CWEs) in the category. OWASP said it suspects the category's limited presence is due to current challenges in testing for it, which could eventually improve. Regardless, survey respondents overwhelmingly named Software Supply Chain Failures as a top concern.

Most **supply chain vulnerabilities** grow out of the interconnected nature of doing business, involving upstream and downstream partners and third parties. Every interaction involves software whose components (aka dependencies or libraries) could be unprotected. An enterprise can be vulnerable if it doesn't track all versions of its own components (client side, server side or nested), as well as transitive dependencies (from other libraries) ensuring that they are not vulnerable, unsupported or out of date. Components typically have the same privileges as the application, so compromised components, including those that come from third parties or open-source repositories, can have a far-reaching impact. Timely patching and updates are essential—even regular monthly or quarterly patch schedules can leave an enterprise exposed for days or months.

Likewise, the lack of a change management process with your supply chain can create vulnerabilities if you are not tracking Integrated Development Environments (IDEs) or changes to your code repository, image and library repositories, or other parts of the supply chain. An organization needs to harden the supply chain by applying access control and least-privilege policies, ensuring that no individual can create code and deploy it to production without supervision, and that no one can download components from untrusted sources.

Supply chain attacks can take many forms. The notorious SolarWinds attack began when Russian attackers injected malware into an update to the company's popular network management software. It affected about 18,000 customers. Although the number of enterprises actually impacted was closer to 100, that list included major corporations and government agencies. The $1.5 billion Bybit hack, traced to North Korea, involved compromised cryptocurrency apps. The recent **Glass Worm** supply chain attack involved an invisible, self-replicating code that infected the Open VSX Marketplace.



Preventing Supply Chain Exploits

Because supply chain attacks involve the interdependency of systems, defending against them involves an all-encompassing approach. OWASP offers tips for preventing attacks, including having patch management processes in place to:

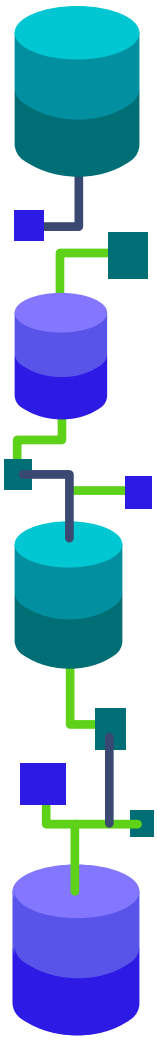
- Know your Software Bill of Materials (SBOM) for all software and manage the SBOM centrally. It's best to generate SBOMs during the build, rather than later, using standard formats, such as SPDX or CycloneDX, and to publish at least one machine-readable SBOM per release.
- Track all of your dependencies, including transitive dependencies, removing unused dependencies, as well as unnecessary features, components, files and documentation.
- Continuously inventory both client-side and server-side components and their dependencies using tools, such as **OWASP Dependency** Check or **retire.js**.
- Stay up to date on vulnerabilities, continuously monitoring sources such as the **Common Vulnerabilities and Exposures** (CVE) website and the **National Vulnerability Database** (NVD) and subscribe to email alerts for security vulnerabilities related to the components you use.

- Use components obtained only from trusted sources over secure links. A trustworthy provider, for instance, would be willing to work with a researcher to disclose a CVE the researcher discovered in a component.
- Deliberately choose which version of a dependency you will use and upgrade only when you need to. Work with third-party libraries that have had their vulnerabilities published in a well-known source such as NVD.
- Monitor for unmaintained or unsupported libraries and components. If patching is not possible, consider deploying a virtual patch to monitor, detect or protect against the discovered issue.
- Regularly update developer tooling.
- Treat components in your CI/CD pipeline as part of this process, hardening and monitoring them while documenting changes.

Change management or a tracking process should also apply to your CI/CD settings, code repositories, sandboxes, integrated developer environments (IDEs), SBOM tooling, created artifacts, logging systems and logs, third-party integrations such as SaaS, artifact repository and your container registry. You also need to harden systems, from developer workstations to the CI/CD pipeline. Be sure to also enable multi-factor authentication while enforcing strong identity and access management policies.

Protecting against software supply chain failures is a multi-faceted, ongoing endeavor in the face of our highly interconnected world. Organizations must employ strong defensive measures for the entire lifecycle of their applications and components in order to defend against this rapidly evolving, modern threat.

4 Cryptographic Failures



Cryptographic failures refers to a category of sensitive data exposure-related bugs. This kind of attack has been one of the most impactful breaches over the past few years. There is a medium level of sophistication required, and sometimes special equipment on the part of the attacker, but it's not overly hard for a hacker to pull off in many cases, and tools exist to automate some of the attack functions.

Sensitive data exposure occurs whenever information that is only meant for authorized viewing is exposed to an unauthorized person in a nonencrypted, unprotected, or weakly protected state. Most of the time this involves typical data that hackers want to steal such as credit card numbers, user identification, business secrets and personal information that might be protected by laws and industry regulations.

These days, nobody would store highly targeted information like that without encryption. But with sensitive data exposure, hackers can sometimes get at it anyway by indirectly attacking the encryption scheme. Instead of trying to decrypt strong encryption directly, they instead steal crypto keys, or attack data when it's moved to a non-encrypted state such as when it's being readied for transport.

In this chapter, we will learn:

- How attackers can trigger sensitive data exposure.
- Why sensitive data exposure is so dangerous.
- Techniques that can fix this vulnerability.

How Do Attackers Exploit Data Exposure?

Sensitive data exposure normally happens when sites don't employ strong end-to-end encryption to protect data, or when there are exploitable flaws in the protection scheme. It can also happen when the encryption used is particularly weak or outdated.

Hackers will often try and find ways to get around encryption if it's not extended everywhere. For example, if a password database stores information in an encrypted state, but automatically decrypts it when retrieved, a hacker might be able to use one of the attacks we previously covered in these blogs, such as SQL or XML injection, to order the database to perform the decryption process. Then the data would be decrypted for the hacker, with no additional effort. Why try and break down a steel door when you can just pickpocket the key?

Weak encryption is also a problem. For example, if credit cards are stored using an outdated encryption scheme, it could be an issue if a hacker is able to use something like a file upload attack to pull the entire database over to their computer. If the captured data was protected using something strong like AES-256 bit encryption, then it would still be unbreakable even if it landed in a hacker's possession. But if weaker or outdated encryption is used, something like the older DES standard, then a hacker with special equipment such as a rack of graphics processing units (GPUs) can task them to break the encryption in a relatively short time.

Why is Sensitive Data Exposure Dangerous?

Sensitive data exposure is dangerous because it lets unauthorized users see protected information. If the data wasn't important, it wouldn't be protected, so any breach of that protection is going to cause problems. It's never a situation that an organization wants to find itself facing.

How much trouble a sensitive data exposure can cause depends on the kind of data that gets exposed. If user or password data is stolen, then that could be used to launch further attacks against the system. Personal information exposure could subject users to secondary attacks such as identity theft or phishing. Organizations might even find themselves vulnerable to heavy fines and government actions if the exposed data is legally protected by statutes like the Health Insurance Portability and Accountability Act (HIPAA) in the United States or the General Data Protection Regulation (GDPR) in Europe.

Eliminating Sensitive Data Exposure

Stopping sensitive data exposure begins with ensuring strong, up-to-date and end-to-end encryption of sensitive data across an enterprise. This includes both data at rest and in transit. It's not enough to encrypt sensitive data while it sits in storage. If it is unencrypted before use or before transport, then it can be exposed using a secondary attack that tricks a server into unencrypting it.

Data in transit should always be protected using Transport Layer Security (TLS) to prevent exposure using man in the middle or other attacks against moving data. And sensitive data should never be cached anywhere in the network. Sensitive data should be sitting with strong encryption in storage or sent using TLS protection, giving attackers no weak points to exploit.

Finally, do an inventory of the kinds of sensitive data that is being protected by your organization. If there is no reason for your organization to store such data, then dump it. Why expose yourself to potential trouble for no possible benefit? Data that isn't maintained by an origination can't be stolen from it.

More Information About Cryptographic Failures Leading to Sensitive Data Exposure

For further reading, you can take a look at what OWASP says about **sensitive data exposure**.

5 Injection

Injection vulnerabilities take many forms, and they have enjoyed the top spot in the OWASP Top 10 for many years. However, they've now been knocked all the way down to number three. Despite this, SQL injection in particular remains a popular and dangerous bug (despite its old age) and this chapter will focus on finding and fixing that one in particular. Scroll to the bottom of this chapter for links to comprehensive guides on defeat NoSQL, OS Command, Email Header and XQuery injection.

Now, let's get started.

In simple terms, SQL (or Structured Query Language) is the language used to communicate with relational databases; it's the query language used by developers, database administrators and applications to manage the **massive amounts of data being generated every day**.

Our data is fast becoming one of the world's most valuable commodities... and when something is valuable, bad guys will want to get their hands on it for their benefit.

Attackers are using SQL injection -- one of the oldest (**since 1998!**) and peskiest data vulnerabilities out there -- to steal and change the sensitive information available in millions of databases all over the world. It's insidious, and developers need to understand SQL injection (as well as how to defend against it) if we are to keep our data safe.

We'll discuss three key aspects of SQL injection:

- Why SQL injection works.
- Why it's so dangerous.
- How to defend against it.

Eliminating SQL Injection

SQL injection can be understood by using one word: context. Within an application, two contexts exist: one for data, the other for code. The code context tells the computer what to execute and separates it from the data to be processed.

SQL injection occurs when an attacker enters data that is mistakenly treated as code by the SQL interpreter. One example is an input field on a website, where an attacker enters "' OR 1=1" and it is appended to the end of a SQL query. When this query is executed, it returns "true" for every row in the database. This means all records from the queried table will be returned.

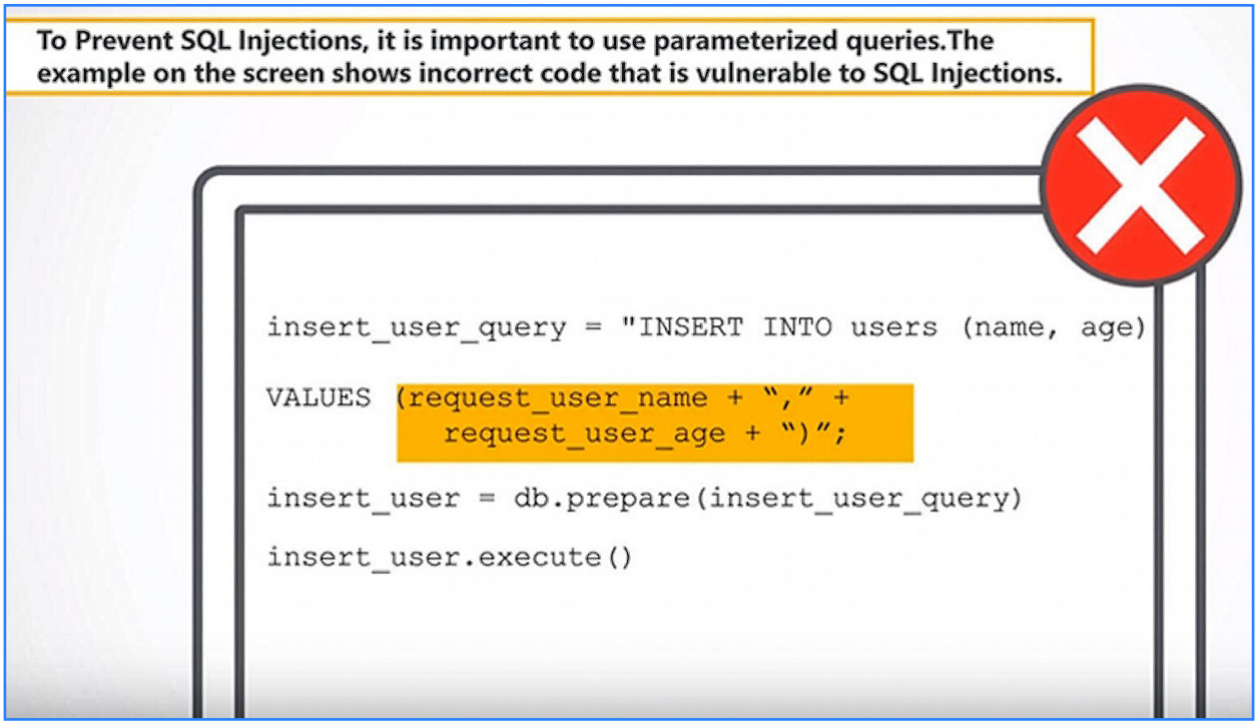
The implications of SQL injection can be catastrophic. If this occurs on a login page, it could return all user records, possibly including usernames and passwords. If a simple query to take data out is successful, then queries to change data would too.

Let's take a look at some vulnerable code so that you can see what an SQL injection vulnerability looks like in the flesh. Check out this string of code:

```
String query = "SELECT account balance FROM user_data WHERE
user_name = "+ request.getParameter("customerName");

try {
        Statement statement = connection.createStatement( ... );
        ResultSet results = statement.executeQuery( query );
}
```

The code here simply appends the parameter information from the client to the end of the SQL query with no validation. When this happens, an attacker can enter code into an input field or URL parameters and it will be executed.

The key thing is not that attackers can only add "' OR 1=1" to each SELECT query but that an attacker can manipulate any type of SQL query (INSERT, UPDATE, DELETE, DROP, etc.) and extend it with anything the database supports. There are great resources and tools available in the public domain that show what is possible.



We'll learn how to correct this issue. First, let's understand how much damage can be done.

Why SQL Injection is So Dangerous

Here are just three examples of breaches caused by SQL injection:

- Illinois Board of Election website **was breached** due to SQL injection vulnerabilities. The attackers stole the personal data of 200,000 U.S. citizens. The nature of the vulnerability found meant that the attackers could have changed the data as well, although they didn't.
- Hetzner, a South African website hosting company, **was breached** to the tune of 40,000 customer records. A SQL injection vulnerability led to the possible theft of every customer record in their database.
- A Catholic financial services provider in Minnesota, United States, **was breached** using SQL injection. Account details, including account numbers, of nearly 130,000 customers were stolen.

Sensitive data can be used to take over accounts, reset passwords, steal money, or commit fraud.

Even information not considered sensitive or personally identifiable can be used for other attacks. Address information or the last four digits of your government identification number can be used to impersonate you to companies, or reset your password.

When an attack is successful, customers can lose trust in the company. Recovering from damage to systems or regulatory fines can cost millions of dollars.

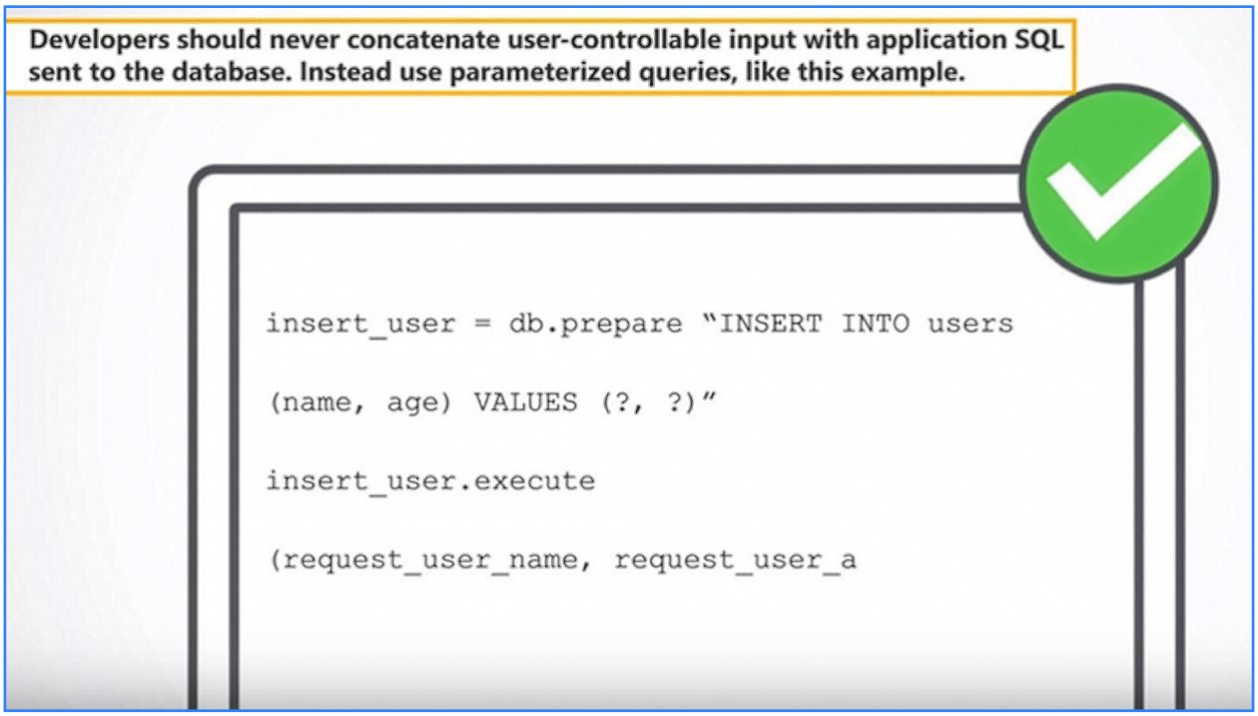
But it doesn't have to end that way for you.

Defeat SQL Injection

SQL injection can be defeated by clearly labeling parts of your application, so the computer knows whether a certain part is data or code to be executed. This can be done using parameterized queries.

When SQL queries use parameters, the SQL interpreter will use the parameter only as data. It doesn't execute it as code.

For example, an attack such as "' OR 1=1" will not work. The database will search for the string "OR 1=1" and not find it in the database. It'll simply shrug and say, "Sorry, I can't find that for you." An example of a parameterized query in Java looks like this:



Most development frameworks provide built-in defenses against SQL injection. Object Relational Mappers (ORMs), such as **Entity Framework** in the .NET family, will parameterize queries by default. This will take care of SQL injection without any effort on your part.

However, you must know how your specific ORM works. For example, **Hibernate**, a popular ORM in the Java world, can **still be vulnerable** to SQL injection if used incorrectly.

Parameterizing queries is the first and best defense, but there are others. Stored procedures also support SQL parameters and can be used to prevent SQL injection. Keep in mind that the stored procedures **must also be built correctly** for this to work.

```
// This should REALLY be validated too
String custname = request.getParameter("customerName");
// perform input validation to detect attacks
String query = "SELECT account_balance FROM user_data
WHERE user_name = ? ";

PreparedStatement pstmt = connection.preparedStatement(
query );
pstmt.setString(1, custname);
ResultSet results = pstmt.executeQuery( );
```

Always validate and sanitize your inputs. Since some characters, such as "OR 1=1" are not going to be entered by a legitimate user of your application, there's no need to allow them. You can display an error message to the user or strip them from your input before processing it.

In saying that, don't depend on validation and sanitization alone to protect you. Clever humans have found ways around it. They're good Defense in Depth (DiD) strategies, but parameterization is the surefire way to cover all bases.

Another good DiD strategy is using 'least privilege' within the database and whitelisting input. Enforcing least privilege means that your application doesn't have unlimited power within the database. If an attacker were to gain access, the damage they can do is limited.

OWASP also has a great **SQL Injection Cheat Sheet** available to show how to handle this vulnerability in several languages and platforms.

The Journey Begins

You've made some great progress towards understanding SQL injection, and the steps needed to fix it. Awesome! We've discussed how SQL injection occurs; typically with an attacker using input to control your database queries for their own nefarious purposes.

We've also seen the damage caused by the exploitation of SQL injection vulnerabilities: Accounts can be compromised and millions of dollars lost...a nightmare, and an expensive one at that. We've seen how to prevent SQL injection:

- Parameterizing queries
- Using object relational mappers and stored procedures
- Validating and whitelisting user input

Now, it's up to you. Practice is the best way to keep learning and building mastery, so why not check out our **Learning Resources** on SQL injection? You'll be well on your way to becoming a Secure Code Warrior.

Discover more types of injection:

NOSQL Injection

OS Command Injection

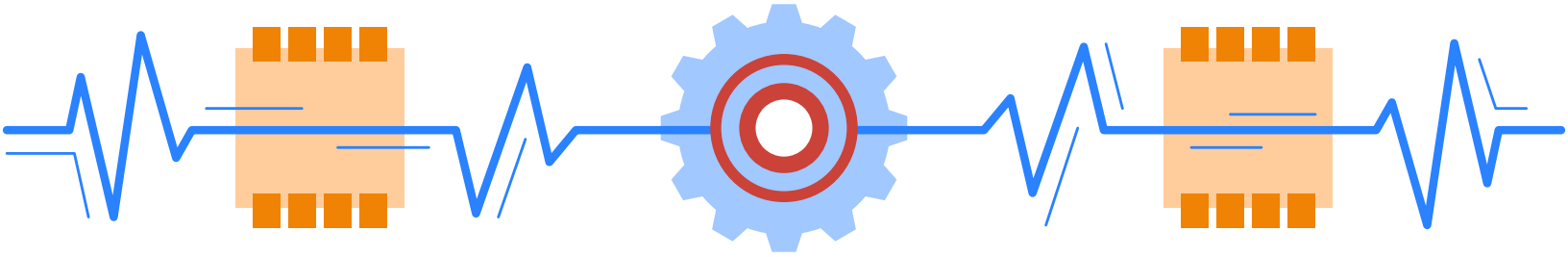
Email Header Injection

XQuery Injection

6 Insecure Design

Adding the **insecure design category** to the OWASP list represents a shift in how the rankings designate and represent threats and vulnerabilities. Instead of calling out single vulnerabilities, many of the OWASP top ten entries now represent broad families of weaknesses. And among them, insecure design is among the most encompassing.

OWASP explains that insecure design is different from insecure implementation because insecure design is at the heart of the code. You can have a completely secure design that introduces vulnerabilities as it is implemented and put into a production environment. But if the design is insecure to begin with, then even a perfect implementation won't do any good. The vulnerabilities will still exist.



They further explain that the biggest factor contributing to the insecure design category is a lack of business risk profiling of the software or system being developed, which results in a failure to determine what level of security design is required. The entire category seems to have been created to help push better threat modeling and the use of secure reference architectures when creating code, something that Secure Code Warrior wholeheartedly supports.

Some notable common weakness enumerations (CWEs) cited by OWASP that are directly created by insecure design include CWE-209: Generation of Error Message Containing Sensitive Information, CWE-256: Unprotected Storage of Credentials, CWE-501: Trust Boundary Violation, and CWE-522: Insufficiently Protected Credentials.

How to Defeat the Insecure Design Family of Vulnerabilities

Eliminating insecure design vulnerabilities involves many of the tactics and techniques that Secure Code Warrior always advocates on our web pages and through our video series. Because the new category is so broad, most efforts to improve secure coding will have a big impact on removing this threat from your environment.

There are several key areas to concentrate on that will go a long way to securing code. One of the most effective is to establish a secure library of code for common tasks like providing authentication for apps, programs, and APIs. Instead of asking developers to write new code every time one of those common tasks is required, they should pull from the secure library and directly add it to their code.

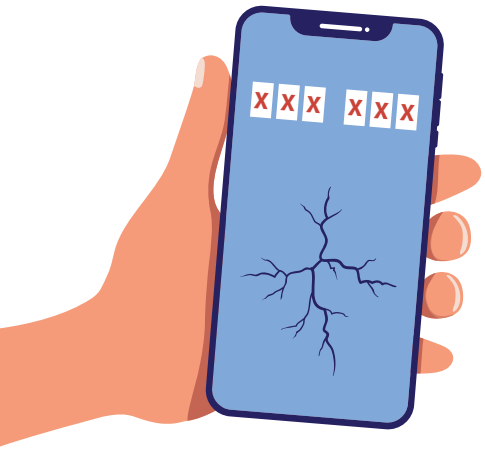
Another important strategy when eliminating insecure design is embracing threat modeling, which OWASP also highly recommends. A good threat modeling program should focus on critical areas where vulnerabilities can cause the most damage. This includes authentication processes, access control, and key flows. The best threat modeling programs are fully integrated into the development process so that vulnerabilities are found and eliminated as the code is still being created.

Programs and applications should also be tested for areas that typically bypass threat modeling. These include things like business logic flaws. For example, you need to know what happens when a user tries to order billions of items from an eCommerce site, and the app needs to have specific instructions for handling any strange and unpredictable exceptions.

Finally, training developers in secure coding techniques and best practices will be critical in eliminating insecure design from your environment. Developers need to be supported in their efforts to learn about and deploy secure code. That means offering training to your developers and also rewarding those who excel at secure coding. It might also mean asking the best coders to become security champions who support the rest of the team, with appropriate recognition and rewards for doing so.

The new insecure design category is both encompassing and broad, so there is no one specific silver bullet that can eliminate it. However, organizations that embrace best practices, better threat modeling, and security training (plus rewards) for developers will immediately start to mitigate it. As those efforts start to experience continued success, organizations will find that the problems of insecure design will naturally wane at the same time. And once you have established a robust secure coding practice, insecure design vulnerabilities will hardly be a threat at all.

7 Authentication Failures



In this chapter, we will cover one of the most common problems faced by organizations that either run websites or which allow employees to remotely access computer resources - which is pretty much everyone. And yes, you probably guessed that we are going to be talking about authentication.

While authentication vulnerabilities are not exploits themselves, having them as part of a login or user authorization process makes an attacker's work easy. If a hacker can simply log into a system as an administrator with a valid user name and password, then there is no need to deploy advanced techniques to battle network defenses. The system simply opens the door and lets the attacker inside. Worse yet, if the attacker doesn't do anything too outlandish, their presence is almost impossible to detect since most defenses will simply see them as a valid user or administrator doing their job.

The category of authentication vulnerabilities is quite large, but we will go over the most common problems that tend to get accidentally baked into user login processes. By shoring up these holes, you can eliminate the vast majority of authentication problems from your organization.

In this episode, we will learn:

- How some common authentication vulnerabilities are exploited.
- Why they are so dangerous.
- What policies and techniques can be used to eliminate authentication vulnerabilities.

How Do Attackers Exploit Authentication Vulnerabilities?

There are quite a few authentication vulnerabilities that might creep into a login or user authorization system, so hackers exploit each one a little bit differently. First, lets go over the most common vulnerabilities and then give examples demonstrating how a couple of them might be exploited.

The most common authentication vulnerabilities include:

- Having weak or inadequate password policies,
- Allowing unlimited login attempts,
- Providing information back to an attacker on failed logins,
- Sending credentials over insecure channels,
- Weakly hashing passwords,
- And having an insecure password recovery process.

Having a weak password policy is likely the most common vulnerability. If users are allowed to create passwords with no restrictions, far too many of them will use easily guessable ones. Every year various computer news organizations put out a list of the most used passwords, and "123456" and "password" are always in the top five. There are others. Administrators like to use "God" quite a lot. True, those are all either humorous or easy to remember, but also very easy to guess. Hackers know what the most common stupid passwords are, and try them first when attempting to breach a system. If those kinds of passwords are allowed in your organization, you will get breached eventually.

A less obvious but still dangerous vulnerability is providing information back to a user regarding a failed login. This is bad because if you return one message when a user name does not exist and another when a user name is correct but the password is bad, it allows attackers to map out valid users on a system and concentrate on guessing passwords. If this is combined with the authentication vulnerability that allows unlimited password guessing, it would enable attackers to run dictionary attacks against whatever valid users they have found, which might get them into a system fairly quickly if the password they are trying to guess is simply a word or well-known phrase.

Why Are Authentication Vulnerabilities So Dangerous?

There is a classic tale from the American Old West about a paranoid homesteader who installed triple locks on his front door, boarded up his windows and slept with lots of guns in easy reach. In the morning he was found dead. His attackers got to him because he forgot to lock the back door. Authentication vulnerabilities are a lot like that. It really doesn't matter what kind of cybersecurity platform you are running or how many expert analysists you employ if an attacker can simply submit a valid user name and password to enter your network unopposed.

Once inside, there are very few restrictions on what that attacker can do. So long as they act within their user permissions, which can be quite extensive if they have compromised an administrator account, there is very little chance that they will be caught in time to prevent serious problems. This makes the authentication class of vulnerabilities one of the most dangerous to have on any system.

Eliminating Authentication Vulnerabilities

One of the best ways to eliminate authentication vulnerabilities from a network is to have good, globally enforced password policies. Not only should users, even administrators, be restricted from using passwords like "password" but should be forced to add in a level of complexity that would make it unfeasible for an attacker to apply a dictionary or common phrases type of attack.

You can come up with your own rules for password creation based on the importance of the system being protected, but a good standard is to force at least three of the following complexity rules on password creation. Passwords must contain:

- At least 1 uppercase character (A-Z),
- At least 1 lowercase character (a-z),
- At least 1 digit (0-9),
- At least 1 special character including punctuation marks and spaces,
- And be at least 10 characters long.

Optionally, passwords should also be no more than 128 characters long and not have more than two identical characters grouped together.

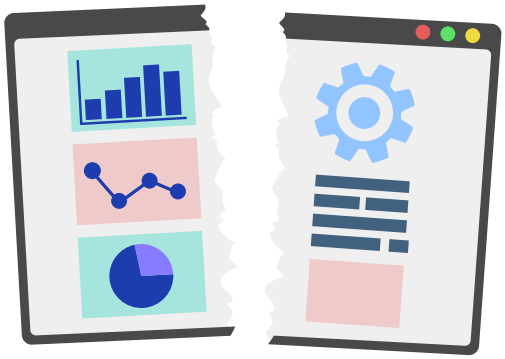
Doing that will prevent attackers from guessing passwords. You should also restrict the number of failed password attempts so that if an incorrect password is entered more than, say three times, the user is locked out. The lockout can be temporary as even a few minutes delay will prevent dictionary attacks from continuing. Or it can be permanent unless the account is unlocked by an administrator. In either case, security personnel should be alerted whenever such a lockout occurs so they can monitor the situation.

Another good way to prevent attackers from gathering information is to craft a generic message whenever either a bad user name or password is entered. It should be the same for both cases so that hackers wont know if they have been rejected because a user does not exist or due to having the wrong password.

Authentication vulnerabilities are among the most common and dangerous on most systems. But they are also fairly easy to find and eliminate.

For more information about authentication vulnerabilities, you can take a look at the OWASP **authentication cheat sheet**.

8 Software and Data Integrity Failures



The **OWASP Top Ten** list has enjoyed a bit of an overhaul, and now includes categories of vulnerabilities grouped together into single listings. This better represents the threat landscape, and ties many variants of each vulnerability back to a specific cause.

What Are Software and Data Integrity Failures?

The **Software and Data Integrity Failure** category was likely added to the OWASP list because of several high-profile breaches that recently used it in order to compromise otherwise secure networks.

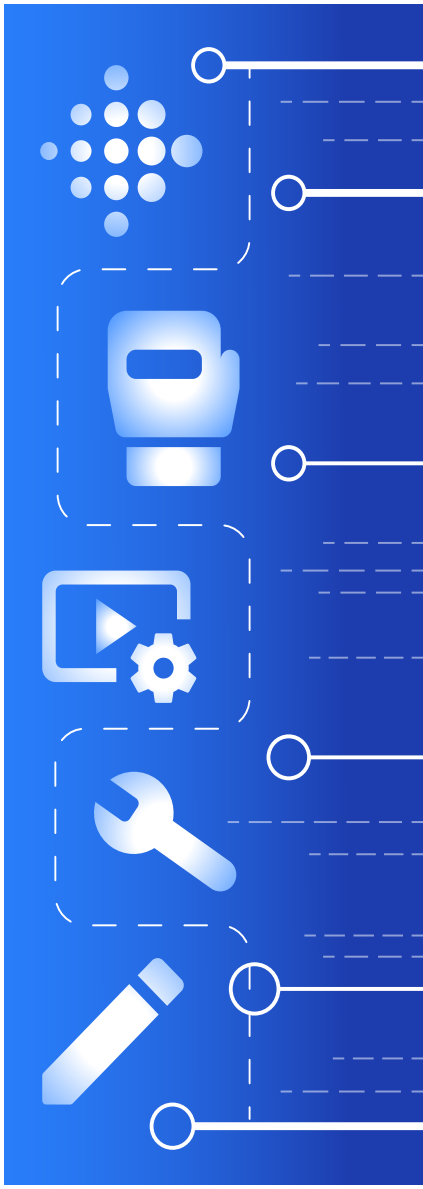
The vulnerability happens when developers or IT staff make assumptions about the security related to software updates and other components of programs being used in their environment. It can occur if, for example, an application calls on or pulls APIs or code snippets from insecure libraries. It can also happen if software update processes are fully trusted and an attacker is able to compromise a program or application farther down the supply chain.

Notable Common Weakness Enumerations (CWEs) include CWE-829: Inclusion of Functionality from Untrusted Control Sphere, CWE-494: Download of Code Without Integrity Check, and CWE-502: Deserialization of Untrusted Data. But the biggest example of a software and data integrity failure in action was the so-called **SolarWinds attack**.

SolarWinds is a company that provides system management tools to help with network monitoring and other technical services. It was installed at many of the world's top companies as well as government organizations and agencies. Attackers were able to compromise the SolarWinds software update process, installing a back door that was added to every organization that used SolarWinds when they received their next product update.

Because the SolarWinds software update was a trusted process, it was not put under much scrutiny at the 18,000 organizations that deployed it. Once the back door was installed, hackers were able to pick and choose which organizations they wanted to infiltrate. About 100 were ultimately compromised, including some otherwise highly secure government agencies. The attack was so subtle that it went unnoticed for months. And when it was revealed, the scope of the compromise was front-page news around the world. So we are certainly talking about a very dangerous vulnerability.

How to Defeat the Software and Data Integrity Failures Family of Vulnerabilities



Making sure that this new vulnerability does not affect your organization requires a few extra steps beyond what is required when combatting others on the OWASP list. Of course, you want to maintain the best practices that Secure Code Warrior always advocates via our content and video series. But once you have secure code in place, or if you are using software or applications from other vendors, you need to also take steps to secure your supply chain.

First off, you should only accept updates that are digitally signed so that you can ensure that they are coming from the vendor themselves and not a third party who compromised the update channel. That would not have helped in the SolarWinds incident because the vendor itself was compromised and sending out the malicious code.

To fully protect your organization, you should add a software supply chain security tool. These tools specifically look at update data and scan it for vulnerabilities. There are many to choose from, and OWASP offers two including the OWASP Dependency Check and the OWASP CycloneDX. But whatever tool is used, it's absolutely critical that all software update processes are scrutinized and checked for vulnerabilities. Even if the program being patched is trusted, you should never trust an update process, and always scrutinize it.

Beyond just checking update processes using tools, you should also implement a review process for all code and configuration changes to minimize the chance that malicious code or configurations can be snuck into your software pipeline. For extra security, the entire CI/CD pipeline should be segregated to ensure that no untested code slips into your production environment.

SolarWinds showed the danger of software and data integrity failures. It also forced organizations to look at their software update processes as a possible avenue of attack. But now that we know about it, steps can be put in place to close this overlooked loophole in many organizations' cybersecurity defenses. This new vulnerability certainly deserves a spot on the OWASP list, but now that we are fully versed about the dangers, it's a vulnerability that can be mitigated and ultimately defeated.

9 Logging and Alert Failures

Logging & Alerting Failures are among the most dangerous conditions that can exist within a network defensive structure. If vulnerabilities or conditions in this category are present, then almost any advanced attack made against it will eventually be successful. Having insufficient logging and monitoring means that attacks or attempted attacks are not discovered for a very long time, if at all. It basically gives attackers the time they need to find a useful vulnerability and exploit it.

OWASP's **comprehensive logging cheat sheet** delivers an extensive overview of the category-wide basics, and below, we will step through the core concepts.

We will learn:

- How attackers can use insufficient logging and monitoring.
- Why insufficient logging and monitoring is dangerous.
- Techniques that can fix this vulnerability.

How Do Attackers Exploit Security Logging and Alerting Failures?

At first, attackers don't know if a system is being properly monitored, or if log files are being examined for suspicious activity. But it's easy enough for them to find out. What they will sometimes do is launch some form of inelegant, brute force type of attack, perhaps querying a user database for commonly used passwords. Then they wait a few days and try the same kind of attack again. If they are not blocked from doing it the second time, then it's a good indication that nobody is carefully monitoring the log files for suspicious activity.

Even though it's relatively simple to test a network's defenses and gauge the level of active monitoring happening, it's not a requirement of successful attacks. In fact, hackers don't need to do anything to actively exploit the situation caused by insufficient logging and monitoring. They can simply launch their attacks in such a way as to make as little noise as possible. More often than not, the combination of too many alerts, alert fatigue, poor security configurations or simply a plethora of exploitable vulnerabilities means that they will have plenty of time to complete their goals before defenders even realize that they are there.

Why Are Authentication Vulnerabilities So Dangerous?

Insufficient logging, monitoring and alerting is dangerous because it gives attackers time to not only launch their attacks, but to complete their goals long before defenders can launch a response. How much time depends on the attacked network, but different groups like the Open Web Application Security Project (OWASP) put the average response time for breached networks at 191 days or longer.

Think about that for a moment. What would happen if robbers held up a bank, people called the police, and it took them half a year to respond? The robbers would be long gone by the time police arrived. In fact, that same bank can be robbed many more times before the police even respond to the first incident.

It's like that in cybersecurity too. Most of the high-profile breaches that you hear about on the news were not smash and grab type of operations. Often times the targeted organization only learns about a breach after the attackers have had more or less full control over data for months or even years. This makes insufficient logging and monitoring one of the most dangerous situations that can happen when trying to practice good cybersecurity.

Eliminating Logging and Alerting Failures

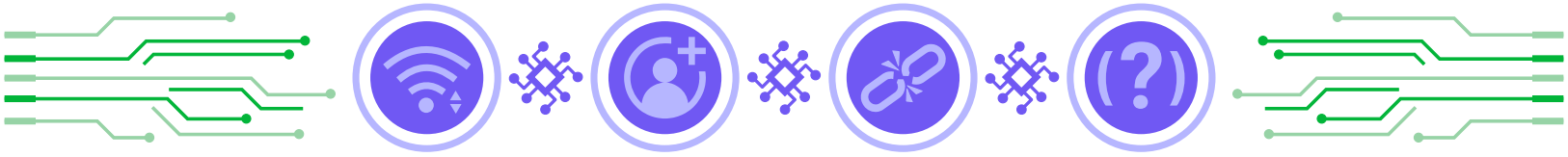
Preventing insufficient logging and monitoring requires two main things. First, all applications must be created with the ability to monitor and log server-side input validation failures with enough user context for security teams to identify the tools and techniques, if not the user accounts, that attackers are using. Or, such input should be formatted into a language like STIX (Structured Threat Information eXpression) which can be quickly processed by security tools to generate appropriate alerts.

Secondly, it's not enough to simply generate good alerts, though that is a start. Organizations also need to establish roles and responsibilities so that those alerts are investigated in a timely fashion. Many successful breaches actually triggered alerts on the attacked networks, but those warning were not heeded because of questions of responsibility. Nobody knew whose job it was to respond, or assumed that someone else was looking into the problem.

A good place to start when assigning responsibilities is adopting an incident response and recovery plan like the one recommended by the National Institute of Standards and Technology (NIST) in **special publication 800-61**. There are other reference documents, including ones specific to various industries, and they don't have to be followed to the letter. But forming a plan defining who within an organization responds to alerts, and how they go about doing that in a timely fashion, is critical.

For further reading, you can take a look at what OWASP says about **logging and alerting failures**.

10 Mishandling of Exceptional Conditions



With OWASP's release of its **2025 Top Ten**, enterprises have a couple of new risk categories to pay special attention to, including a brand-new category that proves, once and for all, that what you don't know can indeed hurt you.

The new category, **Mishandling of Exceptional Conditions**, outlines what can go wrong when organizations aren't prepared to prevent, detect and respond to unusual or unpredictable situations. According to OWASP, this vulnerability can trigger when an application doesn't prevent something unusual from occurring, fails to identify a problem when it crops up and/or responds poorly or not at all when an unexpected situation rears its head.

The idea that enterprises need to be ready for what they didn't see coming reflects the reality of today's highly distributed, interconnected systems. And it's not like OWASP is talking about problems that are unheard of—the Mishandling of Exceptional Circumstances contains 24 **Common Weakness Enumerations** (CWEs). It's just that those CWEs, which involve improper error handling, failure to open events, logical errors and other scenarios, can occur under abnormal conditions. This can result, for instance, from inadequate input validation, high-level errors in handling functions and the inconsistent (or non-existent) handling of exceptions. As OWASP states, "Any time an application is unsure of its next instruction, an exceptional condition has been mishandled."

Those exceptional circumstances can cause systems to fall into an unpredictable state, resulting in system crashes, unexpected behavior and long-lasting security vulnerabilities. The key to preventing this kind of disruption is to, essentially, expect the worst and plan to be prepared for whenever the unexpected happens.

Exceptional Circumstances and the Evolution of the Top Ten

The makeup of the quadrennial Top Ten list of the most serious risks to web application security has been fairly stable through the years, with some categories moving around the list and maybe one or two additions every four years. The 2025 iteration has two new entries, including Mishandling of Exceptional Conditions coming in at No. 10. The other, **Software Supply Chain Failures**, which sits at No. 3, is an expansion of an earlier category, **Vulnerable and Outdated Components** (No. 6 in 2021), that now includes a broad range software compromises. (For those keeping score, **Broken Access Control** still rules the roost as the No. 1 risk).

The exceptional conditions that constitute the newest category can create a host of vulnerabilities, from logic bugs to overflows, and fraudulent transactions to issues with memory, timing, authentication, authorization and other factors. These types of vulnerabilities can impact the confidentiality, availability and integrity of a system or its data. They allow an attacker to manipulate an application's flawed error handling, for example, to exploit the vulnerability, OWASP said.

One example of failing to handle unexpected conditions is when an application identifies exceptions while files are being uploaded during a denial-of-service attack, but then fails to release resources afterward. When that happens, resources remain locked or unavailable, resulting in resource exhaustion. If an attacker intrudes on a multi-step financial transaction, a system that doesn't roll back the transaction once an error is detected could allow the attacker to drain the user's account. If an error is detected part of the way through a transaction, it's very important to "fail closed"—that is, roll back the entire transaction and start over. Trying to recover a transaction in midstream can create unrecoverable mistakes.

Fail Closed vs. Fail Open

So, what's the difference between these two actions? Let's clarify:

Fail Open: If a system "fails open," it continues to operate, or remains "open" when something goes wrong. This is useful when keeping things running is very important, but it can be risky for security.

Fail Closed: If a system "fails closed," it automatically shuts down or becomes secure when there's a problem. This is safer from a security perspective because it helps prevent unauthorized access.

Handling Unforeseen Errors

Preventing this kind of risk starts with planning for the unknown. And that involves being able to detect any possible system error when it occurs and taking steps to solve the problem. You need to be able to properly inform the user (without revealing critical information to the attacker), log the event and, if necessary, issue an alert.

Here's an example of the disclosure of a SQL query error, along with the site installation path, that can be used to identify an injection point:

> Warning: odbc_fetch_array() expects parameter /1 to be resource, boolean given in D:\app\index_new.php on line 188

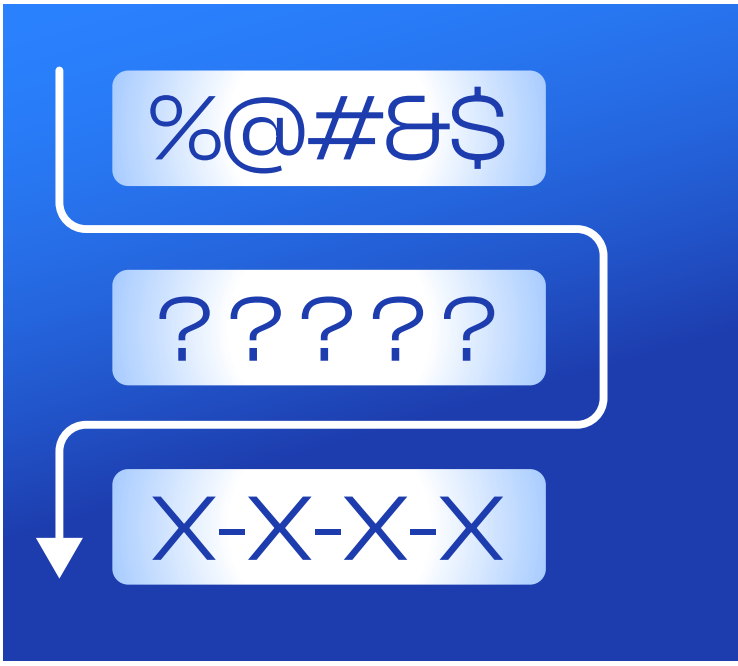
A system ideally would have a global exception handler in place to catch overlooked errors, along with features such as monitoring or observability tooling, or a feature that detects repeated errors or patterns that could flag an ongoing attack. This can help defend against attacks that are intended to take advantage of any weaknesses the enterprise may have in handling errors.

For a standard Java web application, for instance, a global error handler can be configured at the web.xml deployment descriptor level—in this case, a configuration used from Servlet specification version 2.5 and above.

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"

ns="http://java.sun.com/xml/ns/javaee"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"

version="3.0">
...
  <error-page>
    <exception-type>java.lang.Exception</exception-type>
    <location>/error.jsp</location>
  </error-page>
...
</web-app>
```

This little block of code tells a Java web application what to do when something goes wrong behind the scenes. Instead of showing users a confusing error message or a blank screen, it quietly catches the problem and sends them to a custom error page. In this case, that page would be error.jsp.

Because it's set to handle the general java.lang.Exception types, it acts as a master error handler for the whole app. That means no matter where an error happens, users will be redirected to the same friendly, consistent error page instead of seeing raw technical details.



Preventing the Unexpected

Ideally, organizations should work to prevent—as much as possible—exceptional conditions from even occurring. Implementing rate limiting, resource quotas, throttling and other limits can help against denial-of-service and brute force attacks, for example. You may want to identify identical repeated errors and include them only as statistics, so they don't interfere with automated logging and monitoring. A system should also include:

Strict input validation, to ensure that only properly formed and sanitized data is entering the workflow. It should be early in the data flow, ideally as soon as any data is received.

Error handling best practices, to catch errors right where they happen. They should be dealt with efficiently: Tell users clearly that they must keep a log and send alerts if needed. A global error handler is also ideal to catch anything that was missed.

General transaction safety, is also a must. Always "fail closed": if something goes wrong, roll back the entire transaction. And don't try to fix a transaction halfway—it can cause bigger problems.

Centralized logging, monitoring and alerting, along with a global exception handler, which allows for quick investigation of incidents and a uniform process of handling events, while also making it easier to meet compliance requirements.

Threat modeling and/or secure design review, performed in the design phase of projects.

Code review or static analysis, as well as stress, performance and penetration testing performed on the final system.

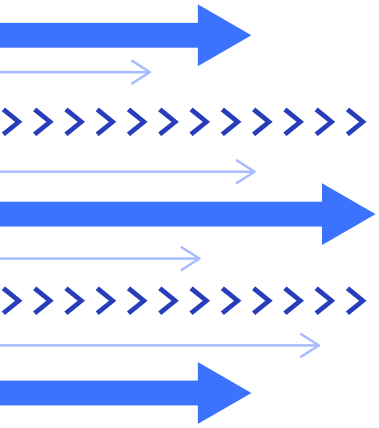
Mishandling of Exceptional Conditions may be a new category, but it involves some basic principles of cybersecurity and emphasizes why enterprises need to be prepared for what they aren't necessarily anticipating. You may not know what form exceptional conditions will take, but you know they will happen. The key is in being prepared to handle all of them in the same way, which will make it easier to detect and respond to those conditions when the inevitable crops up.

Links and Resources

For more information on navigating common vulnerabilities, check out our free **Secure Code Coach**.

Want more traceability and observability of the AI tech stack being used by your developers? Don't miss **SCW Trust Agent: AI**.

Want to see how Secure Code Warrior can help transform the security culture of your organization, and turn your developers into the heroes we need to fight the complex threat landscape? **Book a demo**.



What's Next?

Your security foundation should start with the OWASP Top 10 2025, and we've got interactive, hands-on, and job-relevant **Quests** and so much more to really boost developer-led security awareness and action. And that's just the beginning.

Explore the next level of security-skilled software development with **Secure Code Warrior**.

Note to SCW Trust Score™ Users:

As we update our Learning Platform content to align with the OWASP Top 10 2025 standard, you may observe minor adjustments in the Trust Score for your Full Stack developers. Please reach out to your Customer Success representative if you have any questions or require support.