# CODERS CONQUER SECURITY: YOUR BATTLE PLAN TO DEFEAT THE OWASP TOP 10

The ten most common security vulnerabilities don't stand a chance against secure development superheroes like you. This is your ultimate field guide to understanding each infamous entry in the OWASP Top 10 2017, gaining insight into how each bug 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

# INDEX

You have received a series of missions from Secure Code Warrior. Learn how your targets work, understand their motives, and finally, destroy them. You will have the opportunity to test your skills in our online battleground after reading about each vulnerability.



GOOD LUCK NEW WARRIOR

SECURE CODE WARRIOR

# INSUFFICIENT LOGGING & MONITORING

Insufficient logging and monitoring is one of the most dangerous conditions that can exist within a network defensive structure. If this vulnerability or condition exists, 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.

## IN THIS CHAPTER 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 Insufficient Logging and Monitoring?

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 is Insufficient Logging and Monitoring Dangerous?

Insufficient logging and monitoring 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) puts 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.

**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.**

## Eliminating Insufficient Logging and Monitoring

Preventing insufficient logging and monitoring requires two main things. First, all software 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.

TEST YOUR SKILLS

## MORE INFO ABOUT INSUFFICIENT LOGGING AND MONITORING

For further reading, you can take a look at what OWASP says about **insufficient logging and monitoring**. You can also put your newfound defensive knowledge to the test with the **free demo** of the Secure Code Warrior platform, which trains cybersecurity teams to become the ultimate cyber warriors.

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.

REQUEST A TEAM DEMO

SECURE CODE WARRIOR

# USING COMPONENTS WITH KNOWN VULNERABILITIES

What is the one thing inherent in all software? Components, also known as dependencies or libraries. There is very little code in the world that doesn't depend on other code at some point in time. You even start with a mountain of dependencies from the time you create the software!

Since all software uses components, most of which you haven't written, vulnerabilities within the components you use can become liabilities.

**LET'S DISCUSS WHAT USING COMPONENTS WITH KNOWN VULNERABILITIES MEANS, HOW DANGEROUS IT IS, AND HOW TO RESOLVE IT.**

## Understand Using Components with Known Vulnerabilities

All complex software has vulnerabilities. That's just the nature of the beast, unfortunately. You may never know if your components are 100% safe, however, when vulnerabilities are found in components, you can certainly arm yourself with the knowledge to fix them.

Problems arise most often when components are used past their useful life or after vulnerabilities are found. Most components and libraries release patches for security vulnerabilities at or before the same time the vulnerability is announced. Therefore, when vulnerabilities in components are discovered and announced, updating the components as soon as possible is of utmost importance. Don't leave vulnerable software in production.

Components can come from several sources. Sometimes you buy third-party vendor products which integrate directly with your custom code. These components become a part of your code and operate at the same level of privilege. Another origin is via open source projects hosted on sites such as GitHub. Open source can be dangerous since not all open source libraries have been carefully vetted or audited for vulnerabilities.

Attackers use component vulnerability information to their advantage. Since the vulnerabilities are announced publicly, attackers know about the vulnerabilities the same time you do. Attackers also have techniques they can use to find out what components you're using. Once they know this information, they'll know how to attack your software if it isn't patched.

**IN JULY OF 2017, EQUIFAX, A UNITED STATES CREDIT BUREAU, DISCOVERED A MASSIVE DATA BREACH WHICH LEAKED THE PERSONALLY IDENTIFIABLE INFORMATION OF OVER 147 MILLION PEOPLE.**

## Why Vulnerable Components are Dangerous

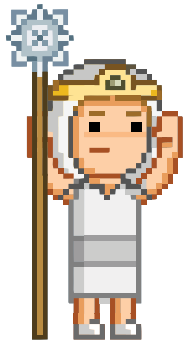
If you're looking for evidence of how dangerous using components with known vulnerabilities is, look no further than the Equifax breach of 2017.

In July of 2017, Equifax, a United States credit bureau, discovered a massive data breach which leaked the personally identifiable information of over 147 million people. The scope and impact of this data breach is unprecedented. Recently, news has come out about Equifax's lax security practices.

One of those lax practices was patch management. Equifax didn't have good patch management practices, which meant their components would go quite some time without getting patched. This was the direct cause of the breach.

Equifax's website used the Apache Struts web framework. Several months before the attackers hacked into the network, a vulnerability was found in the Struts framework, Apache Struts CVE-2017-5638. However, Equifax didn't patch the vulnerability. Attackers used this vulnerability to gain access to Equifax's network. From there, they gained access to a treasure trove of personal information.

Many websites are based on web frameworks not written by the company. This is standard practice since building in all of the necessary functionality from scratch would be too large of an undertaking. However, depending strongly on a framework can open you up to vulnerabilities. **Don't become the next Equifax.**



## How You Can Defeat Vulnerable Components

There is no silver bullet to protecting against using vulnerable components. However, there are policies and controls you can use to mitigate the risk of vulnerable components being used to compromise your systems.

You need to know what components and which version of each component you're using to build your software. Dependency management tools such as OWASP's Dependency Check help you to get a handle on what dependencies you're using. Dependency Check will also tell you if any of those components has a publicly disclosed vulnerability.

A patch management methodology is also essential. When vulnerabilities are discovered, have a system in place for the patches to be downloaded, tested, and released into production smoothly. Keeping your software patched prevents months-old vulnerabilities from being used by attackers.

Finally, have policies in place governing the use of open source and third-party components. Developers don't like to have red tape, and that is understandable. However, there has to be a vetting process for code not written by your organization. It doesn't have to be heavyweight, but is a must to prevent unknown components from being used. At the least, an inventory of components used must be kept up-to-date.

## Don't Get Bitten by a Third-Party Bug

Components will have vulnerabilities. You business software will use components, whether from a vendor or from an open source library. This doesn't mean your organization need be vulnerable to attack.

Even though attackers know what vulnerabilities exist at the same time you do, patches are usually made available at the same time as the public announcements. Be sure to keep track of what your software is using, what is vulnerable and keep your components patched.

# THINK YOU'RE READY TO FIND AND FIX VULNERABLE COMPONENTS RIGHT NOW?

CHALLENGE ME

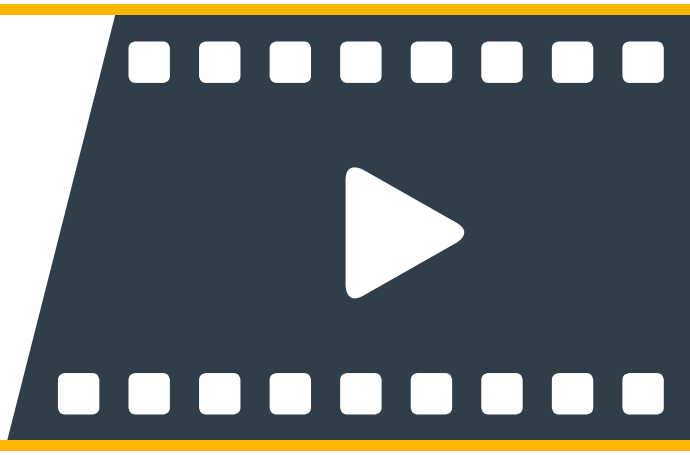
# INSECURE DESERIALIZATION

Depending on the software, the process of serialization can happen all the time. It's the term used to describe whenever data structures or object states are translated into a format that can be stored or possibly sent as a communication. Deserialization is the opposite of this process, taking the now structured data and turning it back into the object or data string that it was before storage.

Insecure deserialization can happen whenever software treats data being deserialized as trusted. If a user is able to modify the newly reconstructed data, they can perform all kinds of malicious activity such as code injections, denial of service attacks or simply changing the data to give themselves some advantage within the software like lowering the price of an object or elevating their privileges.

## IN THIS EPISODE WE WILL LEARN:

- ✔ How attackers can exploit insecure deserialization
- ✔ Why insecure deserialization is dangerous
- ✔ Techniques that can fix this vulnerability



## How do Attackers Exploit Insecure Deserialization?

These days, the most popular data format for serializing data is JSON, though XML is a close second. Quite a few programming languages also offer their own methods for serializing data which often contains more features than JSON or XML. In any case, problems can occur if developers program apps to treat deserialized data as trusted input, as opposed to following the old mantra you probably learned from other blogs in this series, specifically: "Never trust user input!"

User input is never to be trusted because the user can then insert things like code into those strings, which might accidentally be executed by the receiving server. And since raw deserialized data can also sometimes be accessed and exploited, it needs to fall into that same untrusted category.

For example, if forum software uses PHP object serialization to save a cookie containing a user's identification, role and password, then that can be manipulated. A malicious user might change their "user" role to "admin" instead. Or, they can use the opening provided by the data string to inject code, which might be misinterpreted and run by the server as it processes the "trusted" data.

## Why is Insecure Deserialization Dangerous?

It's true that this kind of attack requires some modicum of skill on the part of a hacker, and sometimes trial and error while the attacker learns what kinds of code or exploits the server will accept from their manipulated, deserialized data. That said, this is a commonly exploited vulnerability because of the potential power it gives to hackers skilled enough to use it.

Depending on how the deserialized data is supposed to be used, any number of attacks, including many that we covered in previous blogs, can be employed. Insecure deserialization can be a gateway to remote cross code injection, cross site scripting, denial of service, access control hijacking, and of course SQL and XML injection attacks. It basically opens up a launching point, declares all the data being deserialized to be trusted, and lets the attackers try and exploit it.

## Eliminating Insecure Deserialization

The safest thing that organizations can do to prevent insecure deserialization is to restrict software from accepting deserialized data. That may not be possible or realistic however, but no worries, because there are other techniques that can be employed to defend against this kind of attack.

### MORE INFORMATION ABOUT USING COMPONENTS WITH KNOWN VULNERABILITIES

For further reading, you can take a look at what OWASP says about insecure deserialization. You can also put your newfound defensive knowledge to the test with the **FREE DEMO** of the Secure Code Warrior platform.

If possible, data can be sanitized to something like numeric values. This might not totally stop an exploit, but would prevent code injections from occurring. Even better is simply requiring some form of integrity check against deserialized data such as a digital signature, which could ensure that data strings have not been manipulated. And all deserialization processes should be isolated and run in a low privilege environment.

Once you have those protections in place, be sure to log all failed deserialization attempts, as well as network activity coming from containers or servers that deserialize data. If a user triggers more than a couple deserialization errors in the logs, it's a good indication that they are either a malicious insider or have had their credentials hacked or stolen. You might even consider things like automatic lockouts for users that constantly trigger deserialization errors.

Whichever of these tools you employ to fight insecure deserialization, remember that at the core, this is data that might have been touched or manipulated by a user.

**Never trust it.**

CHALLENGE ME

# CROSS-SITE SCRIPTING (XSS)

Web browsers might be our gateway to all that great stuff online, but sadly, it's not all good news. The inherent behavior of web browsers can be a catalyst for security vulnerabilities. Browsers began by characteristically trusting the markup it saw and executing it without question. That's all fine and dandy until that functionality is exploited for unsavory purposes... and naturally, attackers eventually found ways to exploit this tendency to further their evil ends.

Cross-site scripting (XSS) uses the trust of browsers and ignorance of users to steal data, take over accounts, and deface websites; it's a vulnerability that can get very ugly, very quickly.

Let's take a look at how XSS works, what damage can be done, and how to prevent it.

**WHEN ATTACKERS ADD THEIR MALICIOUS SCRIPT TO A VULNERABLE SITE, IT IS EXECUTED WITHOUT QUESTION. THERE IS NO DEEPER INVESTIGATION, NOR DETECTION MEASURES IN PLACE.**

## How does XSS work?

XSS occurs when untrusted input (often data) is rendered as output on a page but misinterpreted as executable code. An attacker can place malicious executable code (HTML tags, JavaScript, etc.) within an input parameter, which -- when returned back to the browser -- is then executed instead of displayed as data.

As mentioned above, the vulnerability appeared due to the core functioning behavior of browsers, where it is difficult to distinguish data from executable code. The operating model of the web is as follows:

- ✔ User visits a web page
- ✔ The page tells the browser what files to load and what to execute
- ✔ The browser executes what is on the page, no questions asked

This functionality has led to some of the most awesome, interactive experiences we enjoy on the web. The other side of the coin is that it has also led to costly exploits and vulnerabilities. When attackers add their malicious script to a vulnerable site, it is executed without question. There is no deeper investigation, nor detection measures in place.

# CROSS-SITE SCRIPTING (XSS) CONT'

There are three types of XSS:

1. Stored XSS
2. Reflected XSS
3. DOM XSS

**XSS can be used to redirect users to malicious sites, steal cookies and hunt for session data. Basically, whatever JavaScript can do, XSS attacks are capable of as well.**


**Stored XSS** occurs when an attacker can persistently store the malicious script in a data field of the software (e.g. in a field that stores the user's mobile phone number). This sketchy script is then sent to a user's browser every time that data field is displayed in the software.

This type of attack is often seen on forum sites or commenting engines. An attacker enters the malicious script in a comment, and bam - every user who views that comment unknowingly executes the script.

**Reflected XSS** occurs when user input is reflected back to the user's browser as-is. An example is a search box that displays, "You searched for ..." to the user while fetching search results.

Now, imagine that the search works by placing the search term in the URL as query parameters. A malicious attacker could send the victim a link with the malicious script embedded in those very same parameters and truthfully, most web users would barely notice it. The victim clicks the link and is redirected to a phishing site where he/she unwittingly enters their password for the site. Little do they realize, an attacker has just stolen the key to their account.

**DOM XSS** is a relatively new variety of this vulnerability. It takes advantage of complex templating structures found in many UI frameworks such as Angular and React. These templates allow for dynamic content and rich UI software. If used incorrectly, they can be used to execute XSS attacks.

So, there you have it. You've got the scope of XSS in a nutshell. Let's dive deeper into how it can be used destructively.

## Why is XSS so dangerous?

XSS can be used to redirect users to malicious sites, steal cookies and hunt for session data. Basically, whatever JavaScript can do, XSS attacks are capable of as well.

### HERE ARE THREE EXAMPLES OF XSS ATTACKS:

1. Yahoo email users had their session cookies stolen using XSS in 2015.
2. The Samy worm was distributed via an XSS vulnerability in MySpace. It is still the fastest-spreading malware of all time, affecting one million users in just 20 hours.
3. eBay allowed malicious scripts to be included in product descriptions. This led to XSS attacks against eBay users.


XSS attacks are deceptively simple and very serious. They can lead to the theft of sessions, user credentials or sensitive data. Reputational damage and decreased revenue are major pitfalls of these attacks. Even just defacing a website can lead to undesirable consequences for a business.

However, XSS can be defeated by a savvy security warrior just like you. The fix is not complicated and the industry has come a long, long way since XSS became a commonly used exploit.

## You can defeat XSS

The key to defeating XSS is understanding the context. Specifically, the context in which your user input will be rendered back to the client and where it will be rendered back. Inside the HTML code, or inside a JavaScript snippet.

If user input doesn't have to be sent back to the browser, so much the better. But if it is, it often should be HTML-encoded. HTML encoding the output will tell the browser to render the content as-is and not to execute it.

Input validation is important as well. However, validation and whitelisting are not foolproof solutions. Encoding goes a few steps further and stops browsers from executing a malicious script. Whatever is not caught with validation and whitelisting strategies, encoding will pick up.

Many frameworks are now encoding HTML output automatically. Angular, ASP.NET MVC, and React.js are frameworks where default HTML encoding is used. You have to specifically tell these frameworks not to encode by calling a special method.

Most other frameworks, (i.e. Django and Spring) have standard libraries for XSS prevention that you can easily incorporate into your code.

# CROSS-SITE SCRIPTING (XSS) CONT'

The biggest challenge is teaching yourself to analyze all of the ways that user input can enter a system so you can keep your eyes peeled for it. Query parameters can carry attacks, as can post parameters. Follow the flow of data throughout your software and do not trust any data that comes from outside.

Think like border patrol. Stop every piece of data, inspect it, and don't allow it in if it looks malicious. Then encode when rendering to ensure that any bad stuff that was missed still won't cause problems.

Execute these strategies, and your users will be safe from attack via XSS. Take a look at the OWASP Cheat Sheet for even more tips to keep your data under control.


**STOP. INSPECT. ENCODE.**

## Thwart XSS and level up your security skills.

XSS resides at number seven on the OWASP Top 10 2017 list of web security risks. It has been around for a while, but it can still appear and cause problems with your software if you're not careful.

Training is so important for developers in building a security-first mindset as they craft code. And, that training is always at its most effective when it simulates real software, in the languages developers are actively using. With that in mind, why not check out our Learning Resources to learn more about XSS? After that, you can begin the training and practice that leads to your mastery.

## THINK YOU'RE READY TO FIND AND FIX XSS VULNERABILITIES RIGHT NOW?

## CHALLENGE YOURSELF ON THE SECURE CODE WARRIOR PLATFORM.

PLAY NOW

SECURE CODE WARRIOR

# SECURITY MISCONFIGURATION

The term security misconfiguration is a bit of a catchall that includes common vulnerabilities introduced due to the software'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 software 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?

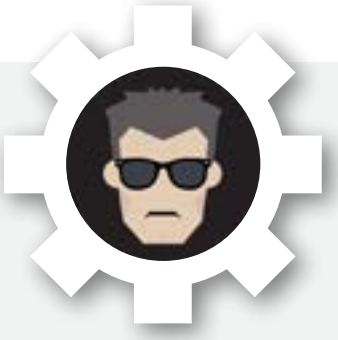
There are a lot of common security misconfigurations. The most popular ones are well-known in hacker communities and are almost always searched for when looking for vulnerabilities. Some of the most common misconfigurations include, but are not limited to:

- 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.

# SECURITY MISCONFIGURATION CONT'

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 are Security Misconfigurations so Dangerous?

Depending on the exact security misconfiguration being exploited, the damage can range from information exposure to complete software 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. To learn more about defeating this vulnerability, and a rogues' gallery of other threats, visit the Secure Code Warrior blog.

## READY TO THWART A SECURITY MISCONFIGURATION RIGHT NOW?

## HEAD TO OUR PLATFORM AND CHALLENGE YOURSELF

PLAY NOW

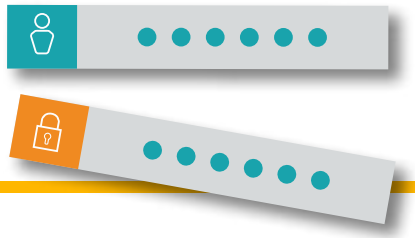
SECURE CODE WARRIOR

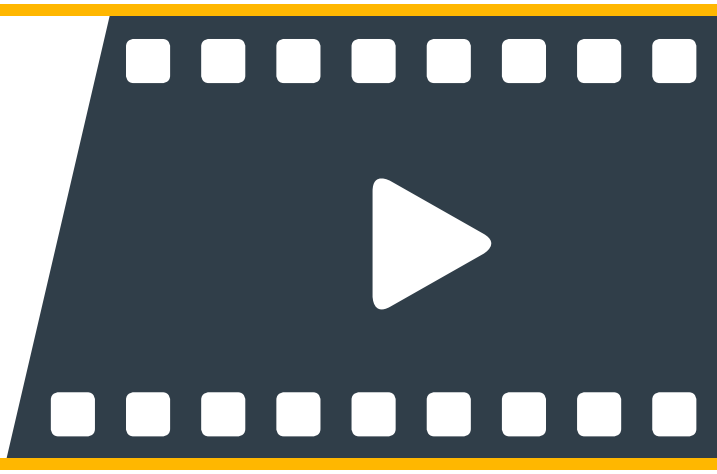
# BROKEN ACCESS CONTROL

When you build business software, 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 software has 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 WHAT BROKEN ACCESS CONTROL IS, WHY IT'S SO DANGEROUS, AND HOW TO FIX IT.





## Understand Broken Access Control

Broken access control occurs when software code does not have the proper security or access checks in place. It can also occur when software 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 software framework which users are allowed to access which pages. This ensures that only the right people are seeing the right functions.

SECURE CODE WARRIOR

## 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 software 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. Software works 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 software. Research your chosen software framework to see what options exist for robust access control.

**CUSTOMER DATA THAT IS NOT PROTECTED PROPERLY COULD LEAD TO A MASSIVE DATA BREACH, HURTING YOUR REPUTATION AND YOUR REVENUE.**

## Protect Your Sensitive Functions

Broken access control can leave your data and your software 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 software safe from malicious attackers and even accidental insiders. Then, you'll know that your data and your functions are safe and secure.

**DEFEAT BROKEN ACCESS**

SECURE CODE WARRIOR

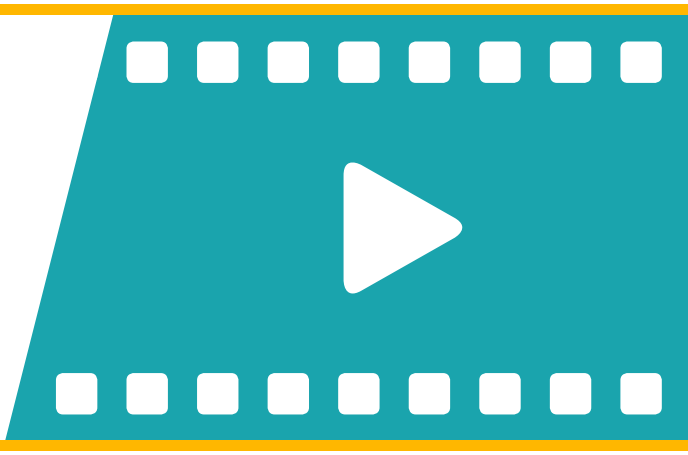
# XML EXTERNAL ENTITIES (XXE)

XML injection attacks are nasty little exploits invented by hackers to help them compromise systems hosting XML databases. This includes the kinds of things that come to mind when one thinks about traditional databases - detailed stores of information about anything from medicines to movies. Some XML datastores are also used to check for authorized users, so injecting new XML code into them can create new users that the host system will accept from that point forward.

For an attacker to implement an XML injection, there needs to be software which relies on, or at least accesses, an XML database. Whenever that happens, and user input is not properly vetted, new XML code can be added to the datastore. Depending on the skill of the attacker, adding new XML code can do quite a lot of damage, or even provide access to the entire database.

## IN THIS CHAPTER WE WILL LEARN:

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



As you read on, you might discover that XML injection is closely related to the SQL injection attacks that we previously covered. That's because even though they target different types of databases, they are extremely similar. And thankfully, the fixes are similar as well. Learning how to defeat one type of attack will put you well ahead of the game when working to prevent the other.

**XML INJECTIONS ARE SUCCESSFUL WHENEVER AN UNAUTHORIZED USER IS ABLE TO WRITE XML CODE AND INSERT IT INTO AN EXISTING XML DATABASE**

## How do Attackers Trigger XML Injections?

XML injections are successful whenever an unauthorized user is able to write XML code and insert it into an existing XML database. This only requires two things to work: software that relies on, or connects to, an XML database and an unsecured data pathway for the attacker to launch their attack.

XML injections are almost always successful if user input isn't sanitized or otherwise restricted before being sent to a server for processing. This can allow attackers to write their own code, or inject it, at the end of their normal query string. If successful, this tricks the server into executing the XML code, allowing it to add or delete records, or even reveal an entire database. Hackers implement an XML injection attack by adding XML code to the end of a string for a normal query. This can be anything from a search field to a login page. It might even include things like cookies or headers.

**For example, in a login field, a user might add the following code after the name or password field:**

```
</user>
<user>
<role>administrator</role>
<username>John_Smith</username>
<password>Jump783!Tango@12</password>
```

In this example, a new user named John_Smith would be created with administrator access. At least the new user is employing good password density rules. Too bad they are actually an attacker.

Hackers don't necessarily need to always hit a homerun like that to be successful with XML injections. By manipulating their queries and recording the various error messages that the server returns, they may be able to map out the structure of the XML database. And that information can be used to enhance other types of attacks.

## Why are XML Injections so Dangerous?

The level of danger involved in an XML injection attack depends on what information is stored within the targeted XML database, or how that information is being used. For example, in the case of an XML database being used to authenticate users, an XML injection can give an attacker access to the system. It might even allow them to become an administrator on the targeted network, which of course is an extremely dangerous situation.

For XML injections levied against more traditional databases, the danger is in having that information stolen, having incorrect data added to the store, or possibly having good data overwritten. XML code is not very difficult to learn, and some of the commands can be extremely powerful, overwriting entire information fields or even displaying the contents of a datastore.

Generally, nobody builds a database unless the information stored there has value. Hackers know this, which is why they often target them. If that data includes things like the personal information on employees or customers, having it compromised can lead to reputation loss, financial consequences, heavy fines or even lawsuits.

## Stopping XML Injection Attacks

XML Injections are fairly common due to the low degree of difficulty in pulling one off, and the prevalence of XML databases. But these attacks have been around for a long time. As such, there are several ironclad fixes that will prevent them from ever executing.

One of the best methods for stopping the attacks is to design software to only use precompiled XML queries. This limits the functionality of the queries to an authorized subset of activities. Anything that comes in with extra arguments, or commands that don't match the precompiled query functions simply won't execute. If you don't want to be quite so restrictive, you can also use parameterization. This restricts user input to specific types of queries and data, for example only using integers. Anything that falls outside those parameters is considered invalid and forces the query to fail.

It's also a good idea to pair precompiled or parameterized queries with customized error messages. Instead of sending back the default, descriptive error messages from a failed query, software should intercept those responses and replace them with a more generic message. Ideally you will want to tell a user why the query failed, but not give them any information about the database itself. If you restrict those custom messages to just a few choices, hackers will not be able to compile any useful reconnaissance from failed queries.

( ) = ‘ [ ] : , * /

If you can't parameterize your queries, you can also restrict special characters by hand. This will prevent attackers from forcing an XML database to consider code injected into a query. In general, you will need to restrict the following characters as well as blanks or whitespaces.

XML injections were highly successful when they were first developed. But given how long ago that was, today we can easily construct defenses that can no longer be breached.

## MORE INFORMATION ABOUT ABOUT XML INJECTIONS

For further reading, you can take a look at the OWASP writeup on XML injections. You can also put your newfound defensive knowledge to the test with the **FREE DEMO** of the Secure Code Warrior platform, which trains cybersecurity teams to become the ultimate cyber warriors.

CHALLENGE ME NOW

SECURE CODE WARRIOR

# SENSITIVE DATA EXPOSURE

The sensitive data exposure 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.

## 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

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.

**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.**

## How do Attackers Exploit Sensitive 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 required. 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 amount of time.

**HACKERS WILL OFTEN TRY AND FIND WAYS TO GET AROUND ENCRYPTION IF IT'S NOT EXTENDED EVERYWHERE**

## 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 either 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.


**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**

## MORE INFORMATION ABOUT SENSITIVE DATA EXPOSURE

For further reading, you can take a look at what OWASP says about sensitive data exposure. You can also put your newfound defensive knowledge to the test with the **FREE DEMO** of the Secure Code Warrior platform, which trains cybersecurity teams to become the ultimate cyber warriors.

**FIX SENSITIVE DATA EXPOSURE**

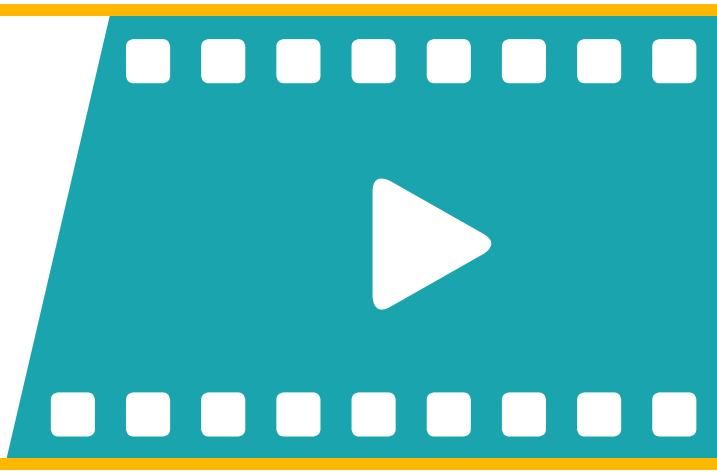
SECURE CODE WARRIOR

# BROKEN AUTHENTICATION

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.

## IN THIS CHAPTER, 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



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.

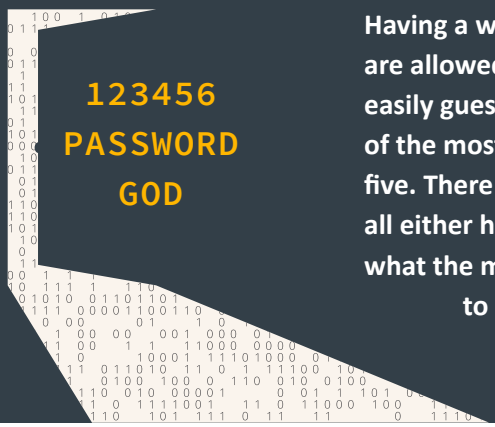
## How do Attackers Exploit Authentication Vulnerabilities?

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.

# 08 BROKEN AUTHENTICATION CONT'

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 & spaces,
- 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.

## MORE INFORMATION ABOUT AUTHENTICATION VULNERABILITIES

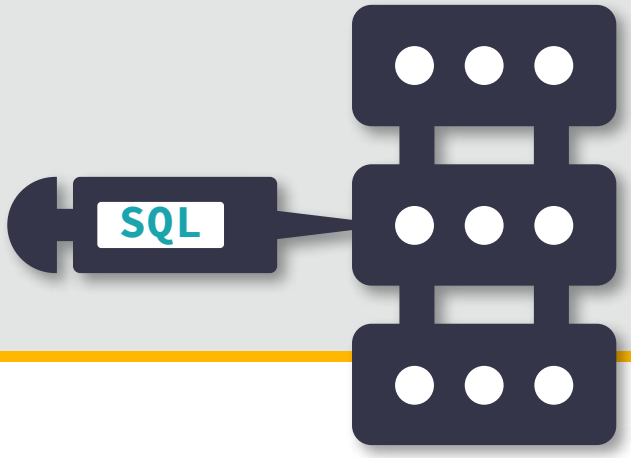
For further reading, you can take a look at the OWASP authentication cheat sheet. You can also put your newfound defensive knowledge to the test with the **FREE DEMO** of the Secure Code Warrior platform, which trains cybersecurity teams to become the ultimate cyber warriors.

# INJECTION

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 software 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.




## TO THAT END, WE'LL DISCUSS THREE KEY ASPECTS OF SQL INJECTION:

1. How it works
2. Why it's so dangerous
3. How to defend against it

## Understand SQL Injection

SQL injection can be understood by using one word: context.

Within software, 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.

# INJECTION CONT'

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 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 SOON. FIRST, LET'S UNDERSTAND HOW MUCH DAMAGE CAN BE DONE.**

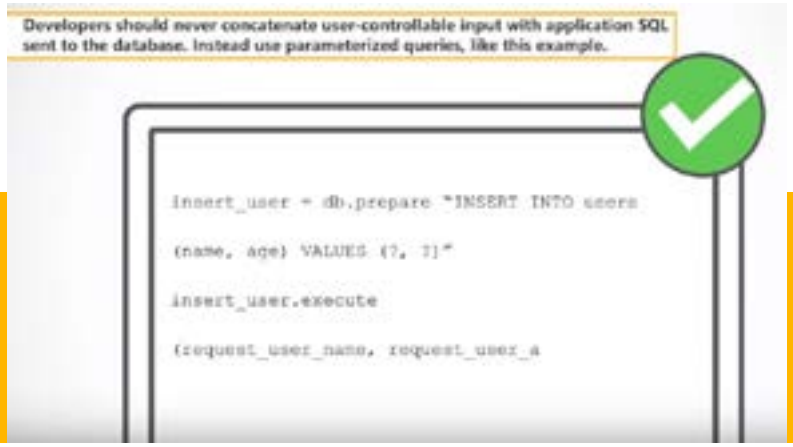
## Defeat SQL Injection

SQL injection can be defeated by clearly labeling parts of your software, 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( );
```

# INJECTION CONT'

</>_[;"=*}

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 software, 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 software doesn’t have unlimited power within the database. If an attacker were to gain access, the damage they can do is limited.

OWASP has a great SQL Injection Cheat Sheet available to show how to handle this vulnerability in several languages and platforms... but if you want to go one better, you can play a SQL injection challenge in your preferred language on our platform right now; here’s some of the more popular ones to get started:

- SQL injection in C#
- SQL injection in Node.js
- SQL injection in Python Django
- SQL injection in Java Spring

## 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:

- [x] Parameterizing queries
- [x] Using object relational mappers and stored procedures
- [x] 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, then try our **FREE DEMO** of the platform? You’ll be well on your way to becoming a Secure Code Warrior.

PLAY NOW

SECURE CODE WARRIOR

SECURE CODE WARRIOR

# BLAST OFF!

- Learn security best practice while having fun in a highly engaging, gamified environment using real code
- Contribute to the security strength of your organizaton, helping to reinforce a positive, secure-by-design culture
- Be empowered with the tools and knowledge to build secure code from the start.

## THE TEN MOST COMMON SECURITY VULNERABILITIES DON'T STAND A CHANCE AGAINST SECURE DEVELOPMENT SUPERHEROES LIKE YOU.

**Want more information on how to conquer security?**

**DISCOVER OUR RESOURCES**

**Want to see how we can benefit you and your team at work?**

**REQUEST A DEMO**

### ABOUT SECURE CODE WARRIOR

Secure Code Warrior is a global security company that makes software development better and more secure. Our vision is to empower developers to be the first line of defense in their organization by making security highly visible and providing them with the skills and tools to write secure code from the beginning. Our powerful platform moves the focus from reaction to prevention, training and equipping developers to think and act with a security mindset as they build and verify their skills, gain real-time advice and monitor skill development. Our customers include financial institutions, telecommunications providers and global technology companies in Europe, North America and Asia Pacific.


securecodewarrior.com

@securecodewarrior
info@securecodewarrior.com
insights.securecodewarrior.com
@seccodewarrior
/securecodewarrior
/company/secure-code-warrior