SYNOPSYS®

Anatomy of an Application Security Weakness

What It Is and How to Deal With It

Table of contents

In software development, there are only slightly more weaknesses than there are ways to define them. But as Shakespeare might ask, what's in a name? That which we call a software weakness by any other name would pose as much risk.

What is an application security weakness?

Vulnerabilities, weaknesses, flaws, faults, bugs, holes: They're all names for the errors, mistakes, and poor design choices in software and systems that attackers can exploit to get into (or get information out of) a system. Perhaps the most commonly used term in software development is "vulnerability." Open Web Application Security Project (OWASP) defines a vulnerability as "a hole or a weakness in the application, which can be a design flaw or an implementation bug, that allows an attacker to cause harm to the stakeholders of an application."[1]

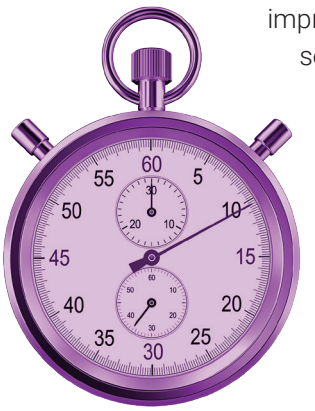
Then what is a software or application security weakness? According to The MITRE Corporation, which maintains the Common Weakness Enumeration (CWE) website detailing over 800 software errors, "Software weaknesses are flaws, faults, bugs, vulnerabilities, and other errors in software implementation, code, design, or architecture that if left unaddressed could result in systems and networks being vulnerable to attack."[2]

As you can see, there's a lot of overlap between the definitions of a vulnerability and a weakness. But some experts use "vulnerability" to refer to a known instance of a weakness in a specific piece of software (i.e., the kind of vulnerability that gets a CVE entry) and "weakness" to refer to generic issues. So to avoid confusion, for the rest of this eBook, we'll discuss application security weaknesses. We'll talk about where they come from, why developers introduce them, what makes them exploitable, how you can mitigate them, and how to keep them out of your code.

Where do application security weaknesses come from?

Most application security weaknesses result from a mistake in how software is written.[3] But they can also come from a flaw in user management or how the system responds to data input.[4]

Sometimes these software "errors" are not accidents but conscious decisions made by development teams. Factors that influence these decisions include (1) the pressure on developers to quickly deliver new and improved features customers are willing to pay for and (2) the disincentive for developers to build security in, especially if the organization doesn't promote a culture that makes security everyone's responsibility.[5]



Others posit that software organizations' release schedules prevent teams from striking a balance between achieving aspired-to security levels and hitting deadlines. This is primarily because higher levels of security result in longer testing hours, higher development costs, delayed deliveries, and higher selling prices.[6] But why can't developers write secure code at the same time they are creating new or improved features? Isn't there a way to write and test code at the same time?

Why don't developers write secure code?

Some argue that when developers learn to code, their instructors emphasize speed and total lines of code (LOC) completed, in spite of whatever application security weaknesses may occur. For example, even the best coding bootcamps follow compressed learning agendas crammed into as little as nine weeks,[7] raising the question of how much focus goes to secure coding. And code marathons that last an entire weekend obviously place all their bets on speed and show little to no concern for coding errors or technical debt that can lead to application security weaknesses.

So while some question the secure coding practices of coding bootcamps,[8] given the paucity of data, formal institutions of higher learning have to answer to the analysts.

Unfortunately, the statistics aren't promising. For example, none of the top 40 ranked U.S. universities that offer undergraduate degrees in computer science require even a single course in secure coding or secure application development, according to Forrester.[9] Internationally, things are not much better, with only one university (Cambridge) in the top five computer science programs offering a course that contains elements of secure code design.[10]

None of the **top 40 ranked U.S. universities** that offer undergraduate degrees in computer science **require even a single course** in secure coding or secure application development

How do attackers exploit application security weaknesses?

The CWE/SANS Top 25, which highlights the most dangerous software errors from the CWE, lists SQL injection as its No. 1 application security weakness. The OWASP Top 10 2017, which focuses on web application security risks, also names injection risks of all types (including SQL and LDAP) as its top application security weakness.

And SQL injection has more than earned its reputation for putting applications in critical condition. For example, Magento recently revealed that its e-commerce platform, used by more than 300,000 websites, contained an SQL injection weakness. Attackers could potentially execute their own SQL commands to extract credit card numbers and other personally identifiable information (PII) and transfer them to a remote server.[11]

So let's take a closer look at SQL injection as a use case for how injection weaknesses typically function.

Application security weakness case study: SQL injection

What is SQL injection?

SQL is a programming language that allows users to interact with a database. Users can execute SQL statements to perform tasks such as finding, adding, modifying, and deleting data in database tables. For example, a web application with a search function takes a visitor's search term, runs an SQL statement that queries the database to find matches, and displays the results in the visitor's browser.

SQL injection occurs when attackers insert their own malicious code into an SQL statement that the application sends to the database. By injecting this code, they can manipulate sensitive data and even retrieve information that lets them gain more control over the system. For example, an attacker might tell the database to retrieve administrator usernames and passwords, or even the whole table of users.

Many developers are unaware that SQL statements can be manipulated,[12] so the applications they create can be overly trusting of user-supplied data inputs.[13] Attackers can successfully exploit SQL injection weaknesses when applications allow user text input without parameters, permit dynamic queries, don't validate, filter, or sanitize user-supplied data, or give hostile data direct access to the database.[14]

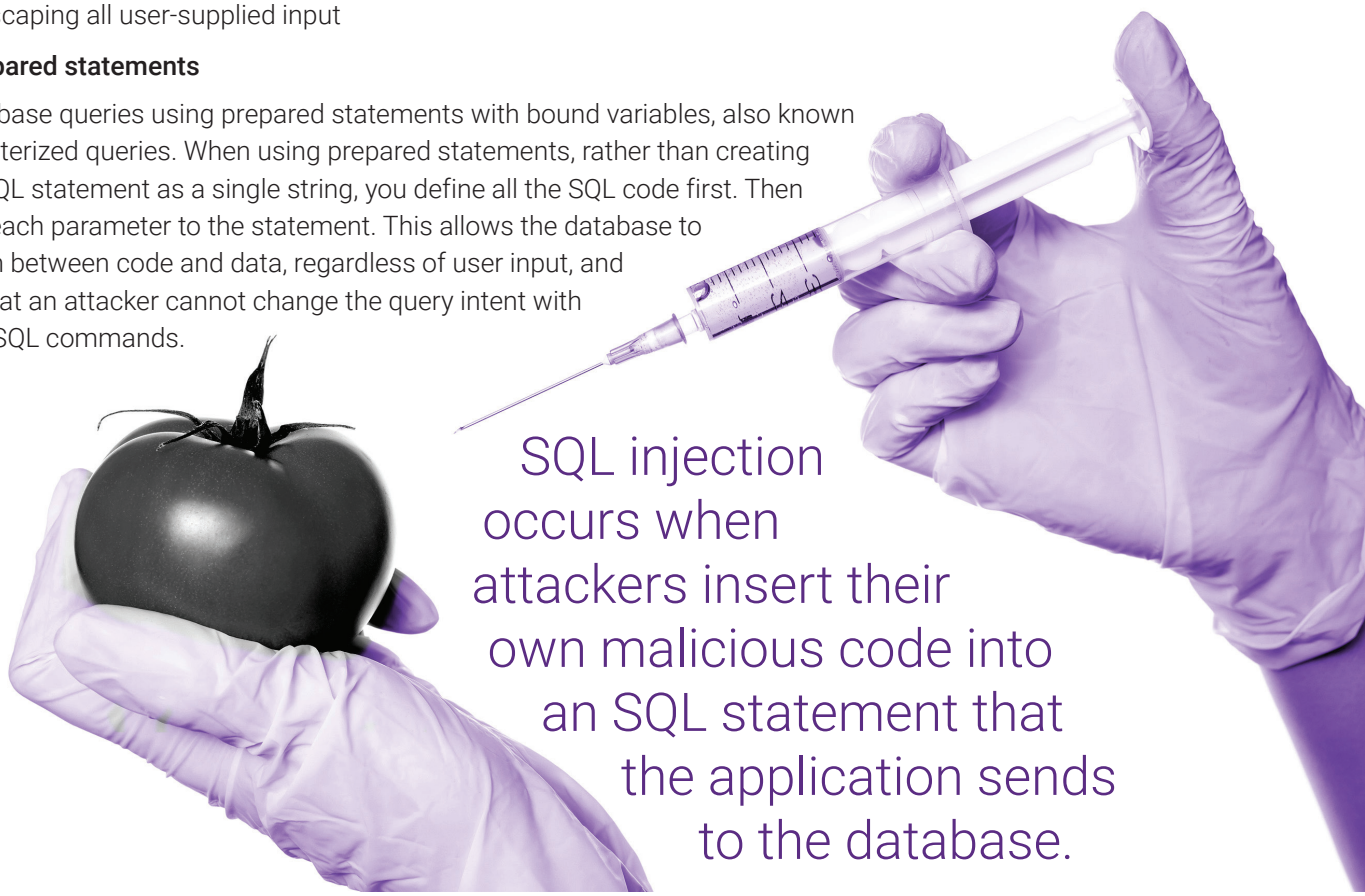
How can I protect against SQL injection?

According to leading GitHub contributors, it's simple to keep SQL injection weaknesses out of your code. Developers should use caution when allowing dynamic queries that include user-supplied text, using these four primary defenses:[15]

1. Using prepared statements (with parameterized queries)
2. Implementing stored procedures
3. Whitelisting input validation
4. Escaping all user-supplied input

Using prepared statements

Write database queries using prepared statements with bound variables, also known as parameterized queries. When using prepared statements, rather than creating the final SQL statement as a single string, you define all the SQL code first. Then you pass each parameter to the statement. This allows the database to distinguish between code and data, regardless of user input, and ensures that an attacker cannot change the query intent with their own SQL commands.



SQL injection occurs when attackers insert their own malicious code into an SQL statement that the application sends to the database.

Implementing stored procedures

Stored procedures contain predefined SQL statements saved in the database. Instead of building and executing the same SQL statements repeatedly, you can simply call a stored procedure and pass in the required parameters. But beware of using dynamic queries within stored procedures, because there is some chance that they could be used for SQL injection.

Whitelisting input validation

Some parts of SQL statements, such as names of columns or tables, disallow bound variables. If you want to allow your users to choose a column to search, for example, you should validate their input and map it to a list of expected column names (e.g., by means of a switch statement) so that unexpected values don't end up in the SQL statement.

Escaping all user-supplied input

Use the proper escaping scheme to escape all user-supplied input before putting it into an SQL statement. This way, the database won't confuse input with developer-written SQL code. Note that this technique should be a last resort when none of the above seem feasible. In general, OWASP recommends this technique only to retrofit legacy code when implementing input validation isn't cost-effective.

How about other types of application security weaknesses?

Not all application security weaknesses are created equal, and not all warrant the same level of concern. But the other application security weaknesses listed by OWASP beneath injection weaknesses still require your attention. So let's take a quick look at three other representative examples and approaches for remediation.

Broken authentication and session management

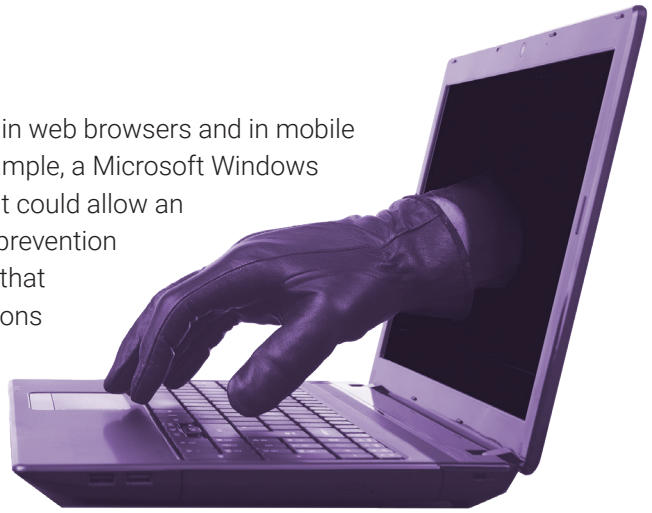
This weakness allows an attacker to impersonate other users through leaks or flaws in authentication or session management procedures (e.g., exposed accounts, passwords, session IDs). For example, real estate title company First American was found to have exposed 885 million files, including sensitive financial documents, to any user via unauthenticated website access.[16] Among other preventive steps, OWASP recommends implementing multifactor authentication to prevent automated, credential stuffing, brute force, and stolen credential reuse attacks, as well as using a server-side, secure, built-in session manager to generate a new random session ID with high entropy after login. Also, don't put session IDs in the URL, but do securely store them, and invalidate them after logout, idle, and absolute timeout.

Sensitive-data exposure

This weakness is principally due to failure to encrypt sensitive data. Without properly validated SSL certificates and encryption for mobile and cloud solutions, data in transit is exposed and subject to exploits such as man-in-the-middle (MITM) attacks to eavesdrop and steal sensitive information. The recent breach of American Medical Collection Agency (AMCA), which affected Quest and LabCorp customers, resulted from a successful MITM attack on the agency's payment pages.[17] To remediate, according to OWASP, at a minimum, you should use up-to-date and strong standards for algorithms, protocols, and keys (with proper key management), and encrypt all data in transit with secure protocols such as TLS with perfect forward secrecy (PFS) ciphers, cipher prioritization by the server, and secure parameters.

Cross-site scripting (XSS)

This weakness happens when an attacker executes their own code in web browsers and in mobile apps that display webpages (e.g., FAQs, help pages). In a recent example, a Microsoft Windows administrative interface was found to contain several XSS flaws that could allow an attacker to victimize a computer and access its network.[18] OWASP prevention methods include using frameworks like Ruby on Rails and ReactJS that automatically escape XSS by design. But you must learn the limitations of each framework's XSS protection and appropriately handle use cases that are not covered. OWASP also suggests that escaping untrusted HTTP request data based on the context in the HTML output—body, attribute, JavaScript, CSS, or URL—will resolve reflected and stored XSS vulnerabilities.



How can I teach my developers to create more secure code?

The steps described above to prevent application security weaknesses seem straightforward and easy enough to implement. Yet they are not currently part of most developers' training or second nature. Developers' first inclination is to produce functional feature code as quickly as possible and worry about security afterward, but this is not sustainable. Something has got to give, as security experts estimate that 5–20 application security weaknesses exist in every 1,000 lines of software code.[19] As suggested by the Forrester report, encouraging developers to instinctively acquire secure coding skills will go a long way toward reducing the rate at which they introduce new application security weaknesses into existing codebases.[20] But how can developers acquire these skills?

> Security experts estimate that 5–20 application security weaknesses exist in every 1,000 lines of software code.

Formal education, bootcamps, and e-learning

You've already read that none of the top 40 American universities require courses in secure coding. As for coding bootcamps, a Google search in August 2019 showed that there were about 10 times as many results for "cyber security bootcamp" as there were for "secure coding bootcamp."

A third option, in the form of real-time e-learning in the integrated development environment (IDE), appears to be the best alternative for software developers and software development managers and directors. An inline secure coding tutorial, accessible from an IDE plugin, can show developers potential errors in their code and instantaneously deliver clickable, brief training modules on how to remediate vulnerabilities.[21]

What about existing application security weaknesses in my code?

The most important benefit of secure coding training may be its capacity for keeping any ***new*** weaknesses from being introduced into your codebase. And that might be the best you can do, because there are hundreds—if not thousands—of unremediated flaws in commercial codebases, and it's impossible to revisit them all. Many of these weaknesses present low impact to confidentiality, low impact to integrity, and low availability. They're worth keeping

an eye on, but perhaps not worth fixing. You're better off focusing your efforts on fixing high-risk existing weaknesses and preventing new ones.

If developers spend too many cycles fixing defective code, they'll never ship any new code. So they need a way to remediate weaknesses while developing. Alternately, you could accept the risk of introducing weaknesses into your code without knowing about them—that is, until you test your application just before deployment, when issues are more expensive to fix.

You could also cross your fingers and hope that no one finds your weaknesses. But it's more likely that after deployment, threat actors and bounty hunters will "test" your application in the wild and exploit any weaknesses they find. Then not only will you have to pay to fix the code—if you even find out—but you'll also have to answer for any exposure of protected data that happened as a result.

When should I focus on finding application security weaknesses?

The best time for developers to find and remediate application security weaknesses is when they are writing code. But what will enable them to do this?

The "shift left" movement is a popular strategy for finding and removing application security weaknesses without throwing a monkey wrench into the software development life cycle (SDLC). The idea is that it's faster and less costly to find weaknesses early in the SDLC. The earlier development teams find weaknesses, the less rework they'll have to do later.

For this reason, many organizations now make their developers responsible for application security.[22] For example, 41% of security decision makers say their software organizations plan to shift the responsibility for static application security testing (SAST) into the development phase of the SDLC by 2020, on top of the 28% who have already implemented/are implementing SAST in the development phase.[23] All these organizations need tools to support their efforts.

The benefits of in-IDE application security testing

With application security testing (AST) tools integrated into the IDE, developers can receive real-time updates on weaknesses they've introduced into their code branch before check-in. Through an intelligent AST tool seamlessly integrated into the IDE via a native plugin, developers can receive immediate security checking of their code as they program. Then, as the tool finds coding errors, it can direct them to just-in-time training on how to remediate the weakness.

An example of this type of solution is the Code Sight™ IDE plugin, which is part of the Synopsys Polaris Software Integrity Platform™. Code Sight enables developers to integrate security testing into their primary development tool in a matter of minutes.

With in-IDE application security testing and just-in-time e-learning showing your developers where the weaknesses live in your codebase and giving them the steps for remediation, your SDLC can accelerate. You'll produce higher-quality software faster. In turn, you'll achieve faster time to market and make more efficient use of your developer resources.

To learn more about the Code Sight IDE plugin with just-in-time e-learning, visit the Polaris Software Integrity Platform page.

1. OWASP, Category:Vulnerability, updated June 6, 2016.
2. The MITRE Corporation, CWE—Frequently Asked Questions (FAQ), updated April 29, 2019.
3. Latest Hacking News, Where Do Software Vulnerabilities Come From?, July 17, 2017.
4. Thomas Holt, What Are Software Vulnerabilities, and Why Are There So Many of Them?, Scientific American, May 23, 2017.
5. Raymond Pompon, Where Do Security Vulnerabilities Come From?, Dark Reading, Sept. 22, 2017.
6. P.K. Kapur, V.S.S. Yadavali, and A.K. Shrivastava, A Comparative Study of Vulnerability Discovery Modeling and Software Reliability Growth Modeling, Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), International Conference on (IEEE Xplore), Feb. 25–27, 2015.
7. SwitchUp, 2019 Best Coding Bootcamps: 50 Top Schools: Rankings, Reviews, and Courses, retrieved July 11, 2019.
8. Steven Zimmerman, Coding Bootcamps Need to Get Real About Secure Coding Practices, Synopsys Software Integrity Blog, April 2, 2019.
9. Amy DeMartine, Trevor Lyness, et al., Show, Don't Tell, Your Developers How To Write Secure Code, Forrester Research, April 19, 2019.
10. Ibid.
11. Jay Jay, Update: Critical Flaw in Magento E-commerce Platform Exposes 300,000 E-commerce Sites to SQL Injection, SC Magazine UK, March 29, 2019.
12. The PHP Group, PHP Manual, Security, Database Security, SQL Injection, retrieved July 3, 2019.
13. Pompon, Where Do Security Vulnerabilities Come From?.
14. OWASP, OWASP Top 10 – 2017: The Ten Most Critical Web Application Security Risks, 2017.
15. Dave Wichers, Jim Manico, et al., SQL Injection Prevention Cheat Sheet, GitHub, updated March 30, 2019.
16. Brian Krebs, First American Financial Corp. Leaked Hundreds of Millions of Title Insurance Records, Krebs on Security, May 19, 2019.
17. Scott Ikeda, Third Party Data Breach Hits Quest Diagnostics With 12 Million Confidential Patient Records Exposed, CPO Magazine, June 11, 2019.
18. Tara Seals, Microsoft Management Console Bugs Allow Windows Takeover, Threatpost, June 18, 2019.
19. Latest Hacking News, Software Vulnerabilities.
20. DeMartine, Lyness, et al., Show, Don't Tell.
21. Ibid.
22. Charlie Klein, How to "Shift Left" With Application Security Tools, and How Not To, Synopsys Software Integrity Blog, Jan. 30, 2019.
23. Amy DeMartine, Stephanie Balaouras, et al., The State of Application Security, 2019, Forrester Research, Feb. 27, 2019.

THE SYNOPSYS DIFFERENCE

Synopsys helps development teams build secure, high-quality software, minimizing risks while maximizing speed and productivity. Synopsys, a recognized leader in application security, provides static analysis, software composition analysis, and dynamic analysis solutions that enable teams to quickly find and fix vulnerabilities and defects in proprietary code, open source components, and application behavior. With a combination of industry-leading tools, services, and expertise, only Synopsys helps organizations optimize security and quality in DevSecOps and throughout the software development life cycle.

For more information, go to www.synopsys.com/software.

SYNOPSYS®

185 Berry Street, Suite 6500

San Francisco, CA 94107 USA

U.S. Sales: 800.873.8193

International Sales: +1 415.321.5237

Email: sig-info@synopsys.com