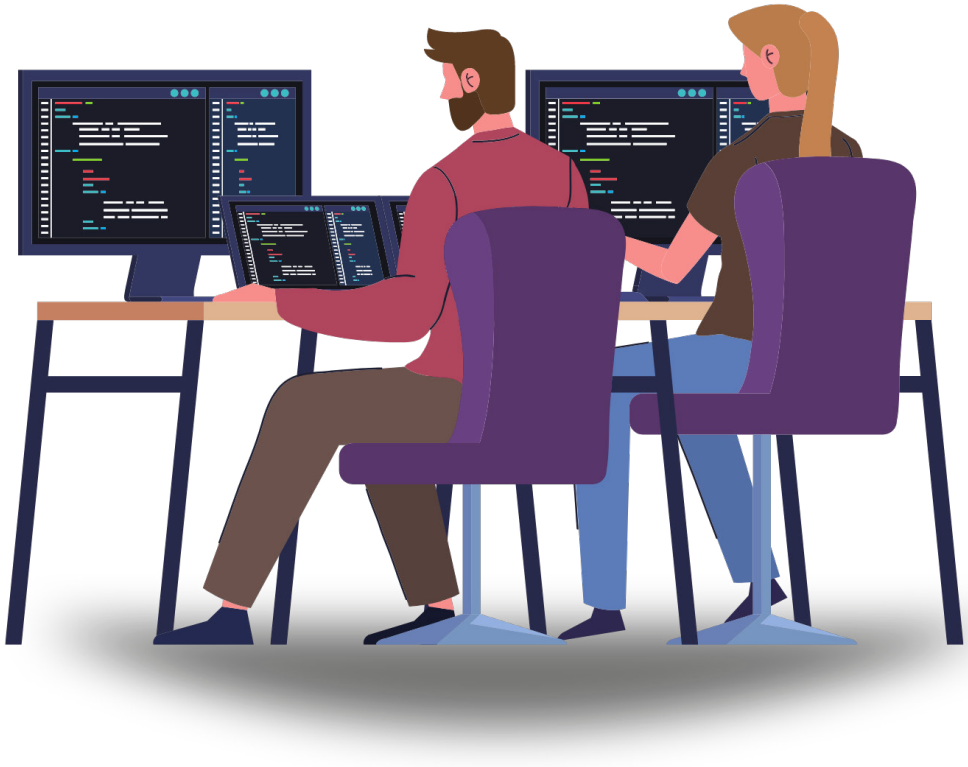
10

10 Essential App **Best Practices** for Developers:

A Complete Guide



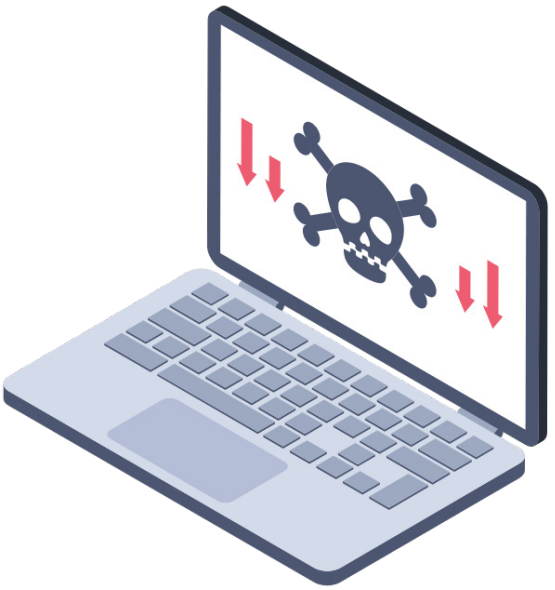
PreEmptive

INDEX

OVERVIEW

Developers must follow good practices to ensure that source code does not contain exploits and that applications are secure. Failing to do so can result in vulnerabilities that attackers exploit to compromise the system's security and users. This can lead to data loss, financial damage, or harm to the company's reputation or the individuals involved. Adhering to best practices helps developers write secure code and reduces the likelihood of introducing vulnerabilities to the application.

Neglecting best practices can introduce security vulnerabilities in the code, making the application more susceptible to attacks. Unfortunately, it doesn't seem that developers have yet to eliminate vulnerabilities from code. In fact, according to HackerOne's 2022 6th Annual Hacker-Powered Security Report, ethical hackers found 65,000+ software vulnerabilities in 2022 — up 21% over 2021.



The problem, however, is not ethical hackers. It's the many unethical hackers out there searching for vulnerabilities to exploit so they can access sensitive information, which leads to data breaches, financial losses, and reputational damage. There have been numerous examples of hacks or data breaches that have resulted from vulnerabilities that could have been eliminated at the development level. For instance:

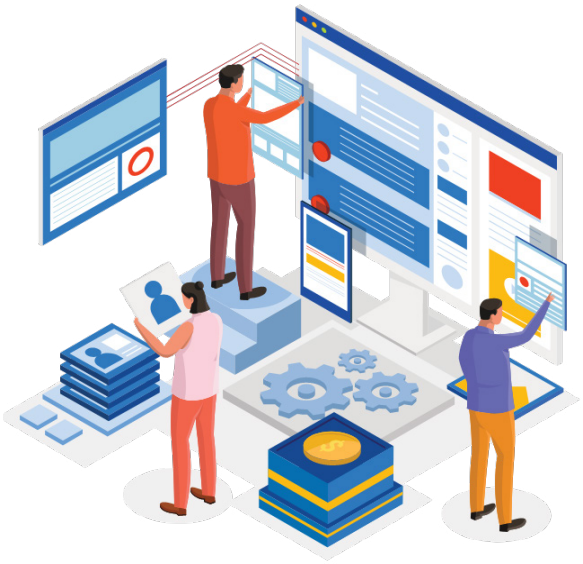
- **SolarWinds Hack**: In 2020, attackers installed malware in software updates distributed to SolarWinds' customers. The exploit exfiltrated sensitive data from the compromised systems. The cause was traced to a vulnerability in the Orion software's source code.
- **Marriott International Data Breach**: In 2020, a vulnerability in a guest services application exposed the personal information of 5.2 million guests.
- **Heartbleed Bug**: In 2014, an error in the OpenSSL source code allowed attackers to steal sensitive data like passwords and private keys from servers that were using affected versions of OpenSSL.

These incidents will only continue. After all, Cybercrime costs are projected to reach $10.5 trillion annually by 2025, and the United States will account for at least a third of it. Businesses and organizations that don't want to find themselves included in that figure must prioritize security — and developers are perfectly positioned to do so.

The 10 Essential Secure *Development Practices*

10

Software developers are the builders who turn ideas into the applications we all use. They coordinate the components, connections, and everything in between that enables you to place an order from your favorite store from your phone, or manage all your banking, or schedule an appointment with your doctor, or any of the other million things we use apps for.



The thing about those apps is that they often coexist next to our personal sensitive data, and oftentimes they even interact with and use it. That's fine – and typically what consumers want. After all, it's hard to view your checking account balance without access to that data. But with that access comes a big responsibility in keeping that data safe and secure.

It might sound like a tall order, but it's actually not an impossible task. In fact, with the right combination of tools and processes, dev teams can produce secure apps that aren't exploited by hackers. But to create these secure apps, developers must adopt certain habits when coding. So let's explore ten of the most essential.

1. *Always* Validate All Input

Check user input to make sure it's exactly and only what's expected.

It's not uncommon for users to interact with an app and provide input of some sort. Text in a form field, an image, a connection to a server — whatever. And then the app processes that information. But what if the information was malicious data or an attempt to get the app to share private information? That's what input validation is for.

Trying to get applications to behave in unexpected ways is a basic hacking tactic, and unexpected input is an easy way to accomplish that. Validating that any input received is the input desired can help to prevent a wide range of attacks including buffer overflow attacks, SQL injection attacks, cross-site scripting (XSS) attacks, and more.

So how can developers make sure that their apps are only processing the type of data they're designed for? There are a few things to remember with input validation and the first — as with most of the strategies in this ebook — reinventing the wheel may not be necessary. Many languages have built-in input validation libraries to check things like emails and credit cards.

Additionally, make sure that inputs match the expected data type. For example, if the input is expected to be a number, the application should validate that the information is a number and not a string. Check the length of user inputs to ensure they are within the expected range. Use input filters to remove unwanted or malicious content from user inputs. And while javascript can quickly check form data on the client-side, make sure to help any client-side validation with server-side validation to prevent attackers from bypassing client-side validation.

2. *Use* Encryption

Protect sensitive data and communication with strong encryption.

Remember that sensitive information we mentioned? Encryption is the standard when it comes to protecting personal and financial information from unauthorized access or theft. Encryption ensures the confidentiality, integrity, and authenticity of the information being transmitted and stored, which helps to reduce the risk of data breaches and cyber-attacks.

Encryption can mitigate many security risks by ensuring that sensitive information is only readable with the correct decryption key — even if it falls into the hands of unauthorized individuals. This is important for preventing man-in-the-middle attacks where attackers intercept communications. Rules and regulations such as General Data Protection Regulation (GDPR), require encryption. Failure to comply can result in fines.

The specific encryption method used will depend on the security requirements and the nature of the data being encrypted but a few standards include:

- Transport layer security and secure sockets layer (SSL) protect network communications.
- Source code obfuscation can make source code unreadable and typically utilizes encryption.
- Asymmetric encryption secures communication between two parties.
- Hashing algorithms make it difficult for an attacker to view a plaintext password.
- Encrypted file storage is a secure storage solution that prevents improper access.

3. Implement *Tight Access Controls*

10

Restrict access to sensitive data and functionality.

Ever heard about a party where being on the guest list was required to get in? Access controls work the same way in that they restrict who can view what data, files, or network services, based on pre-defined rules. It's an important security mechanism that protects sensitive information, financial information, and confidential business information.

Access control keeps track of who has access to resources and what actions they have taken, which can identify the source of security incidents. It may also be a requirement of certain regulations, such as GDPR and HIPAA. By implementing access control in an application, developers can reduce the risk of security threats, hacking, and data breaches.

Fortunately, keeping sensitive information protected is easily achievable. The following are a few examples of how developers can implement access control measures.

- **Role-Based Access Control (RBAC)** is basing access on user roles. For example, an administrator might have access to all information, while a regular user has less access.
- **Attribute-Based Access Control (ABAC)** restricts access to resources based on the attributes of the user, such as their location, time of day, or device.
- **Access Control Lists (ACLs)** are lists of permissions associated with a resource, such as a file or a network service, that specify which users or groups can access it.
- **Multi-Factor Authentication (MFA)** requires the user to provide multiple forms of authentication, such as a password and a security token, to access resources.
- **Token-Based Access Control** generates a token when the user logs in and uses it to authenticate requests for access to resources.

4. *Store* Sensitive Data Securely

Protect sensitive information at all times.

Storing data securely protects sensitive information like personal data, financial information, and confidential information. By implementing secure data storage, developers can reduce the risk of security threats, such as data breaches, hacking, and data theft. Not to mention that secure data storage is often a requirement for compliance with various regulations.

Secure data storage can be achieved by following procedures that store data securely and protect sensitive information from data breaches and unauthorized access. Use solutions that protect data at rest, such as disk encryption and secure file systems. And encrypt sensitive data before storing it in a database or other storage mediums.

5. *Review* Code Regularly

Make systematic code review part of the application development process.

Code review — systematically going through the code, line-by-line, weeding out defects — is an opportunity for developers to learn best practices for secure coding and improve their coding skills. And when it's done in a collaborative environment where developers work together, it promotes a shared understanding of security requirements.

Implement a comprehensive code review procedure by first establishing standards by which all code will be evaluated. Document those standards and train the team accordingly. Empower developers with tools to locate security issues automatically and put processes in place so that issues can be addressed promptly.

6. *Use* Static Application Security Testing

Vulnerability detection and remediation should be regularly done.

Static application security testing (SAST) is done with tools that analyze an application's source code, configuration files, and other components for security vulnerabilities, design flaws, and coding errors that could be exploited. The main advantage of SAST is that it can be performed early so vulnerabilities can be found and fixed instead of carried through to deployment.

SAST can also be automated so it's often a cost-effective security solution that doesn't require a lot of additional resources. It's valuable at every stage, including the build and release pipeline, allowing for constant testing and timely feedback. Some SAST tools, like Kiuwan, also find defects that impact code quality, maintainability, performance, and security.

To add SAST to your processes, first choose a SAST tool by evaluating options compatible with the languages, frameworks, and libraries you use. Configure it and integrate into your build pipeline. Once in place, you can scan code, generate a report, review, and prioritize fixes based on severity and other goals (*this is super easy with our partner product Kiuwan*).

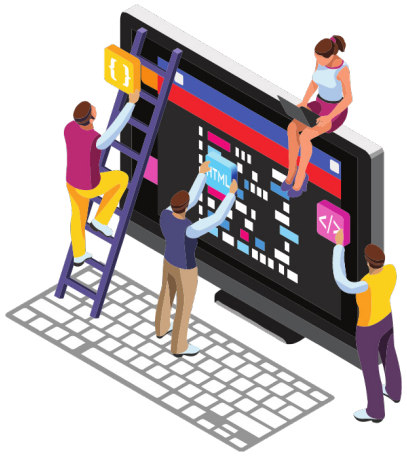
Once vulnerabilities are fixed, re-run the SAST tool to ensure the updates are satisfactory. Repeat the SAST process as part of the regular development cycle, such as during every code commit.

7. *...And* Dynamic Application Security Testing

Simulate an attack on an application to identify security vulnerabilities.

Dynamic application security testing (DAST) is testing that simulates an attack on an application to identify security vulnerabilities. The tester does not know the application's code or internal structure. Rather, DAST tests the application from the outside by sending different inputs and analyzing the output to detect security issues.

DAST tools can emulate attacks, such as brute force, session hijacking, and injection. They may use a combination of automated scanning and manual testing to identify vulnerabilities like SQL injection, cross-site scripting, and broken access controls.



The main advantage of DAST is that it can identify vulnerabilities that other tests like static analysis or manual code reviews may not detect. DAST is also helpful for testing applications that are difficult to test with other methods, such as web and mobile applications. Moreover, DAST gives a glimpse of the application's security posture to find vulnerabilities that may only appear during runtime.

Test all ends of the app, including the web server, application server, and database, providing a complete view of the security status and get immediate feedback on application security so developers can identify and address vulnerabilities quickly.

The process to implement a DAST tool is much like a SAST tool. Weigh the options and select one, configure it to your needs, run it against your apps, analyze the results, and make the fixes. Repeat until you're satisfied with the final results. Repeat the DAST process regularly, such as before each release or on a scheduled basis, to ensure ongoing security.

8. *Harden & Shield* Your Applications

Get additional armor against attacks and unauthorized access.

Application hardening is configuring and securing an application, its operating environment, and its infrastructure to reduce its attack surface and improve its resistance to exploitation. Application...

...shielding puts a layer of protection around the application with techniques such as obfuscation, anti-tampering, and anti-reverse engineering. Together, they provide a multi-layered defense that makes it difficult for attackers to analyze or modify an application.

Multilayered protection like encrypting data, access controls, and security tools to monitor and detect suspicious activity. Reduce risk and stop data breaches that damage an organization's reputation, cause financial losses, and expose sensitive data.
.

9. *Add* Runtime Application Self-Protection

Continue protection even after deployment.

Runtime application self-protection (RASP) is a security technology designed to protect applications from attacks by monitoring the application at runtime and detecting and blocking malicious activity. It is an additional layer of security that can help to mitigate security risks that may have been missed during development or testing and other security benefits.

By monitoring and controlling access to the application at runtime, RASP can help reduce an application's attack surface. RASP can detect and respond to attacks in real-time, helping to prevent the exploitation of vulnerabilities in an application. And it provides total protection for applications, covering a wide range of threats.

RASP is relatively easy to deploy, with minimal impact on application performance. Not to mention that RASP is a valuable addition to an organization's application security strategy by providing real-time threat detection and comprehensive protection.

Many developers integrate runtime application self-protection into the software development lifecycle by selecting a RASP solution that meets their needs, adding it to the application architecture, and testing and optimizing it for consistent and reliable performance.

10. Play By the *Rules*

Be compliant with your industry's regulations, and standards.

Compliance is an important consideration for software developers, especially when developing software used in regulated industries such as finance, healthcare, or government. Typically, compliance and security go hand-in-hand as regulations exist to ensure security, procedural, and...

...other standards are met. So being compliant will often have the benefit of security.

10

But compliance requires proactivity. Stay up-to-date with your industry's regulations, perform routine risk assessments to identify compliance issues, and train staff so that everyone knows their role in ensuring compliance. And start by familiarizing yourself with the regulations that affect your industry and region.

General Data Protection Regulation (GDPR) outlines the collection, storage, and processing of data for EU residents. Software must obtain user consent for data collection, have certain data protection measures, and allow users to request or delete data.

Health Insurance Portability and Accountability Act (HIPAA) is a U.S. regulation that requires the protection and privacy of patient data in the healthcare industry and affects developers who work on software in the healthcare industry.

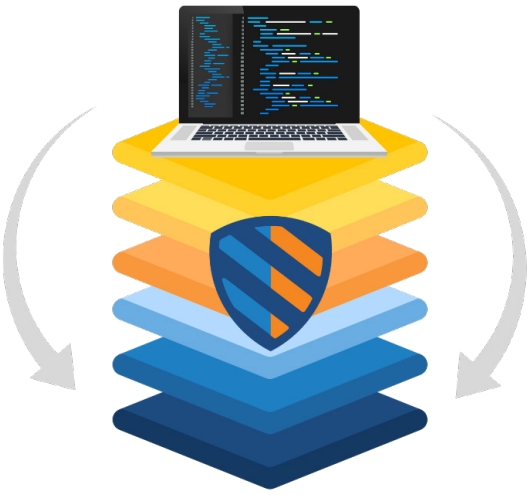
Payment Card Industry Data Security Standard (PCI DSS) outlines how payment card data must be handled and it includes provisions for testing security systems and maintaining documentation and risk assessments.

Sarbanes-Oxley Act (SOX) is a U.S. regulation that applies to publicly traded companies and aims to prevent financial fraud. Software that supports financial processes must comply with SOX requirements like controls over financial data, auditing controls, and documentation.

Federal Risk and Authorization Management Program (FedRAMP) provides a standardized approach to security for cloud products and services for the U.S. Government including a security assessment package and authorization process.

Secure Your Organization's Application Development Process

Do you want peace of mind in knowing that your applications are safe, vulnerability-free, and able to withstand the attacks of hackers and other malicious actors? Implement the strategies described in this guide and make PreEmptive part of your software development lifecycle.



PreEmptive provides obfuscation, instrumentation, and runtime protection for .NET, Java, Android, and JavaScript applications. It's the industry standard for obfuscating and hardening source code so hackers can't snoop around. Plus it adds runtime protection for security controls that protect the application while it is running.

Contact us today to learn how PreEmptive can help safeguard your software development lifecycle from security threats.

Smart App Protection for an ***Unsafe*** *World!*

GET IN TOUCH:

Headquarters in USA
10801 N Mopac Expressway
Building 1, Suite 100
Austin, TX, 78759
phone: **+1 (512) 226-8080**
email: **solutions@preemptive.com**

European Sales
140 bis rue de Rennes
75006 Paris
France
Tel: **+33 01.83.64.34.74**
email: **EuroSolutions@preemptive.com**

JAPAN
AG-Tech Corp
Tel: **+81-3-3293-5300**
Email: **info@agtech.co.jp**