SafeStack Academy

CHECKLIST

Code review security checklist

Including quick implementation guide and detailed implementation manual

SafeStack Academy

Code review security checklist

Quick implementation guide

Purpose

This checklist is designed to help you improve your code review culture by consistently applying secure coding practices. It's not intended as a standalone teaching tool, an accountability mechanism, or as a complete guide to secure development.

How to run the checklist

The checklist has three phases: before code is pushed, during the code review, and before the code review is marked complete. You can include the checklist itself as a template in a code review request, with the review tools configured to require its completion. It may still be helpful to have physical copies visible around your teams' workstations.

The first phase happens before the original author shares their code with the team. It consists of the author verifying they haven't included any real passwords, keys, tokens, or other secrets in their code.

The next phase happens during review and each item may be completed by any of the reviewers besides the original author. The reviewers confirm the right people have been engaged and that they all understand the intended change.

They then check for the presence of debug code, the handling of untrusted data and response information, the correct use of tools, and that there's sufficient log and test coverage.

Finally, if the code review has raised risks beyond the scope of the review to fix, the reviewers raise the risk to their team and make sure it's logged somewhere it will be reviewed. Any of the reviewers can do this.

Modifying the checklist

Though teams should modify the checklist to suit their needs, they shouldn't remove safety steps because they're unable or unwilling to perform them. The entire team should be involved in decisions to modify the checklist, and the modified checklist should be tested on a single system to check it works as intended. Any changes should result in a checklist that's **focused, brief, actionable, collaborative, tested, and integrated.**

Introducing the checklist

Identify a core group of people who are enthusiastic about improving their code review culture. Start with a single system and expand incrementally. Identify key quality metrics and measure them. Integrate the checklist directly into your build-test-release workflow.

SafeStack Academy

Code review security checklist

Before pushing code to the team repository	
Have all secrets been removed from the committed code?	☐ Yes
During the code review (with author, reviewers, testers)	
Have the right people been engaged to review the code?	☐ Yes
Is the purpose of the change stated and understood by the reviewers?	☐ Yes
Is there debug functionality in the code?	☐ No ☐ Yes, and it can only run in test environments
What's the status of user-supplied data?	☐ Validated before it's used or stored ☐ Escaped when it's passed to an interpreter
How are log entries set up?	☐ They cover all key events and states ☐ They include enough information to uniquely identify the event ☐ They exclude secrets and customers' personally identifiable information (PII)
What's been considered for frameworks, libraries, tools, and other dependencies?	☐ They're being used effectively ☐ New dependencies have been vetted ☐ They're up-to-date
How are response messages set up?	☐ They use appropriate status codes ☐ They exclude information that should remain internal to the system ☐ They limit information to the correct level of authorization
What's been considered for testing?	☐ Test coverage is sufficient ☐ Misuse cases are represented
Before completing the code review	
Have unresolved risks been raised and documented?	☐ Yes

SafeStack Academy doesn't intend this checklist to be comprehensive, and we encourage additions and modifications to fit your practice.

Code review security checklist

Detailed implementation manual

SafeStack Academy

Code review security checklist

Detailed implementation manual | Page 1

Introduction

This checklist was created to improve security culture in software development teams and help them consistently check their code for common security risks.

We made it because the earlier vulnerabilities are discovered, the cheaper and easier they are to fix — so the more we can do to support this, the better.

We intend for this checklist to be used as part of a software team's code review process. Our hope is that it will improve teams' overall security posture and the quality of the code they release.

How to use this manual

The "reviewing team" is understood to include the original author, a primary reviewer who approves the change, testers, business analysts, and any secondary reviewers that are tagged in. Everyone in the reviewing team plays a role in enabling safe code reviews.

This manual provides guidance on using the checklist, suggestions for implementation, and explanations of each entry. Teams should adapt the checklist to their own circumstances. Each check has been included based on our expert opinion that its inclusion will reduce the likelihood of security risks and that it is unlikely to introduce risk to a system or add unmanageable cost.

The ultimate goal of this checklist is to prompt a security mindset at code review time, making it safer and easier for everyone to discuss possible security issues in code.

SafeStack Academy

Code review security checklist

Detailed implementation manual | Page 2

How to run the checklist

Before pushing code to the team repository

These checks are done by the author of the change before any code is pushed to a shared area. If continuously integrating, that may mean performing this check with every push.

Have all secrets been removed from the committed code?

Once code has been shared, it's annoying and error-prone to remove passwords, keys, certificates, and other secrets from a branch of commits. It can also be difficult to review, as a secret might have been added in an early commit and removed in a subsequent one.

Secrets that do make it into a shared repository should be replaced. It may be possible to automatically catch some secrets (for example, by searching for variables named "password") but only the author can really confirm they haven't included sensitive information.

Tests may contain secrets only if they are set and consumed by the tests themselves. A couple of useful tips are to aim for generated secrets rather than hand-created ones, and to clearly mark fake secrets as such to make reviewing easier.

During the code review

These checks are done during the review of the code by the primary reviewer. The goal is to confirm safety before merging into a shared branch.

Have the right people been engaged to review the code?

Choosing an effective group of reviewers can be tough as the team needs to balance their workload with the desire to have everyone's eyes on changes. Aim for two to three reviewers for most changes, and tag in additional people when the change meets the following requirements.

- It involves deeper knowledge of the system
- It interacts with another team's system
- It could benefit from other specialised knowledge — for example, subject matter experts or security experts for changes to authentication or authorisation components
- It's considered risky or tricky.

Consider including people who are expected to test the change, and folks less familiar with the system who can ask outsider questions.

SafeStack Academy

Code review security checklist

Detailed implementation manual | Page 3

Is the purpose of the change stated and understood by the reviewers?
Everyone on the reviewing team should have a clear grasp of what the change is trying to do. Teams with issue tracking systems should require all proposed changes to reference the related issue and vice versa. The one irreplaceable feature of manual code reviews is judgement: does this change do what is intended?

Be on the lookout for functionality that isn't related to the core purpose. While a refactor that enables the change is fine, additional changes and fixes should have their own review. At the very least, they should be highlighted in the change's summary.

Is there debug functionality in the code?
Sometimes we write code that makes our system easier to debug by emitting extra information or enabling actions outside of the normal workflow. It's easy to forget to remove this code before pushing it into production. Always remove debug code if possible. If debug code is needed it must be guarded to prevent it running outside of specific test environments, and this check should fail closed. Consider writing tests that confirm this.

What's the status of user-supplied data?

Is it validated before it is used or stored? Is it escaped to suit the context in which it is interpreted?
Injection attacks remain an evergreen threat wherever data is passed to an interpreter — for example, browsers, database query engines, or shells. Any untrusted data should be validated on receipt and be escaped before being used in an interpreter.

SafeStack Academy

Code review security checklist

Detailed implementation manual | Page 4

How are log entries set up?

Do they cover all key events and states?
Investigating performance and security issues is greatly assisted by systems that emit helpful logs. At a minimum, log the following: process start, process end, any exceptions thrown, audit events, input and output events.

Do they include enough information to uniquely identify the event?
Events like user requests should generate a unique identifier that's included in all related messages to enable correlation of individual requests across multiple components. This is critical to understanding how an event plays across the entire system. Include high cardinality fields in log events, and consider using structured logging.

Do they exclude secrets and customers' personally identifiable information (PII)?
As logs cross trust boundaries by their nature, they should limit the amount of customer information they record, ideally only to identifying tokens. Where PII logging is necessary for debugging, take care to limit these entries to diagnostic tools rather than auditing or archival stores.

What's been considered for frameworks, libraries, tools, and other dependencies?

Are they being used effectively?
Frameworks often provide tools to defend against injection attacks, to provide authentication and access control, and to emit logs. Your team must be familiar with how to use these tools safely, and identify when they are making a risky choice.

Have new dependencies been vetted? Are they up-to-date?
Supply-chain attacks are becoming increasingly common and out-of-date dependencies pose a major risk if they have known vulnerabilities. Your team should have processes for assessing new dependencies and for keeping their ecosystem current. In general, software composition analysis tools can (and should) be integrated into build pipelines to automate identification of vulnerable dependencies. Use judgement and coordination to decide when to update a version.

SafeStack Academy

Code review security checklist

Detailed implementation manual | Page 5

How are response messages set up?

Do they use appropriate status codes? Do they exclude information that should remain internal to the system? Do they limit information to the correct level of authorization?
Clear, consistent status codes enable the team to monitor operational issues quickly and effectively. Specifically, error responses that differentiate between authentication failures, server errors, and client errors can be vital.

That said, it's important not to accidentally leak information. For example, returning a different status code if a record exists than if it doesn't gives the user information they're not entitled to know. All sensitive responses should pass through access controls that prevent unauthorized leaking of information.

What's been considered for testing?

Is the test coverage sufficient? Are misuse cases represented?
Tests can identify failing edge cases, bug regressions, and more. While the developer may have written low-level and even integration tests, a tester's perspective is crucial to identifying risks and unexpected inputs.

A misuse case is a representation of something that shouldn't happen and should be tested for. While test cases should be identified earlier in the development process, this is the last chance to check if they're missing, or if further manual testing is needed.

Before completing the code review

This check takes place immediately before completing the review. Concerns that are mitigated or found not to qualify as risks don't need to be escalated beyond the review.

Have unresolved risks been raised and documented?
It may not be possible to fully mitigate an identified risk during review. The risk may even be integral to the nature of the change. Given this, identified risks should be communicated to the wider team and formally accepted by the system owners.

SafeStack Academy

Code review security checklist

Detailed implementation manual | Page 6

Additional notes

Modifying the checklist

Teams should modify the checklist to account for differences in terminology, technological platforms, and team composition. They shouldn't remove safety steps because they're unable or unwilling to perform them.

The entire team should be involved in decisions to modify the checklist, and the modified checklist should be tested on a single system to ensure it works as intended.

Here are the key principles to take into account when modifying the checklist.

- Focus on critical issues that aren't adequately covered by other controls.
- Make it quick to run through.
- Have clearly actionable items to prevent confusion.
- Have collaborative ownership.
- Test changes in small increments.
- Integrate it into the team's workflow.

Teams may consider adding extra safety checks, especially if they're needed in their domain.

Take care not to make the checklist too complex or time-consuming — the last thing we want is for it to feel like a chore people want to avoid.

SafeStack Academy

Code review security checklist

Detailed implementation manual | Page 7

Introducing the checklist to a code review process

Leadership support goes a long way to making the introduction of the checklist successful.

Here are some ways to help that happen.

- Identify and work with a core group of people who are enthusiastic about the checklist. Try to include business leaders and other leaders from outside of your development team.
- Start small, with one team and one system. Wait until they're comfortable with the process and have worked out the kinks.
- Track changes and improvements to metrics like vulnerabilities that make it to release. You can include the checklist items in a code review template (for example, in GitHub, Azure DevOps, BitBucket).
- We advise teams to establish consensus around using a checklist before integrating it into the review process.

Evaluating code review quality

It's helpful to understand your baseline quality and failure metrics before putting a checklist in place.

Ultimately, a code review process should aim to reduce the number of defects that are released to production. It needs to balance this with the cost of the code review process itself.

Here are some measurements to consider.

- Mean time between failures (MTBF)
- Overall equipment effectiveness (OEE)
- First pass yield
- Cost of changes

Identifying the most effective measure of success in code review quality is a work in progress for us — we'd love to hear your thoughts and ideas!

About SafeStack Academy

SafeStack Academy is a community-centric online training platform that takes a flexible, people-focused approach to ongoing cyber security education at a time when it's never been more needed.

By teaching software development teams to weave in security from idea to maintenance, as well as providing cyber security and privacy awareness training for the wider workforce, SafeStack Academy's training programmes offer a comprehensive way of protecting people, systems, and data in an ever-changing world.

academy.safestack.io hello@safestack.io