
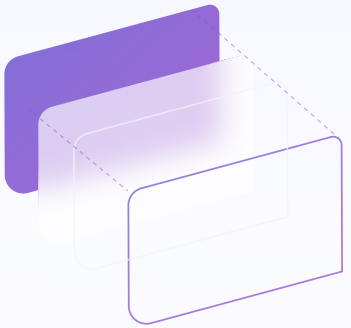

OpenZeppelin

Comprehensive Smart Contract Audit Readiness Guide

Version 2.0

February 28th, 2022


Table of Contents

Preparing for a Smart Contract Audit

Understanding Audits

Smart contract audits provide Web3 developers valuable feedback to address the novel security challenges of distributed systems. Web3 developers must contend with the extreme variability of code, as well as the ongoing evolution of both individual projects and the surrounding ecosystem. Moreover, accumulating value in smart contract systems and DAOs (decentralized autonomous organizations) only makes security considerations more salient. Audits allow a project's developers to demonstrate that their code has been thoroughly inspected and battle-tested, thereby fostering trust and encouraging the project's adoption by prospective users.

Put simply, a smart contract audit is a methodical inspection by advanced experts intended to uncover vulnerabilities and recommend solutions. Working with the client, an auditor defines the scope of the audit and systematically probes for weaknesses in the project's code. At the conclusion of the audit, the auditor provides the client with a report of findings. The client addresses these findings to strengthen the security and scalability of the project. Once any identified vulnerabilities are rectified, the client may choose to make the audit report public in order to demonstrate the project's commitment to security.

Optimal Audit Timing

Clients that benefit most from an audit do so by cultivating a dynamic partnership with the audit team. A good auditor's skillset goes well beyond the mere ability to read code and includes the knowledge and understanding that only come from extensive hands-on experience in auditing and smart contract development. Clients should make use of this expertise, working with the audit team collaboratively to determine the optimal timing for an audit.

Obviously, as on-chain security solutions for smart contracts remain limited, audits are vastly more useful if they take place before a project is deployed. At the same time, however, auditors should not be brought into a project too early. Audits tend to be most productive—and have the most interesting and useful findings—for systems that have achieved a certain level of maturity, i.e., code that is tested, documented, and ready for deployment. For these more mature projects, audits are a highly effective tool for assessing smart contract security.

As part of initial discussions to plan an audit engagement, a prospective client should review the audit checklist below and discuss the status of each item with the auditor.

Audit Readiness Checklist

While there are many aspects to a successful audit (and a successful project), it can be helpful to think of audit readiness in terms of three general categories: (1) The Team, (2) The Community, and (3) The Code. This checklist is intended to help to prepare a project for a successful audit and a promising future.

1.0 The Team

Once a project is solidly established, it will have hundreds of external contributors. Until that time, however, it will need a strong and diverse core group of maintainers. Here are some important things that any reputable auditor will consider about a prospective client's team.

1.1 Harness the Requisite Skills

The team will need people with all the skills and knowledge necessary to successfully implement the full scope of the project. Team members will need a strong understanding both of the project and any external dependencies.

Consider:

1.1.1. Does the team have the requisite competencies for the project? In addition to any project-specific skills, does the team collectively possess the skills listed below?

1.1.1.1. Software Development

1.1.1.2. Understanding of the Blockchain and Smart Contracts

1.1.1.3. DevOps

1.1.1.4. Testing

1.1.1.5. Agile Software Planning

1.1.1.6. Experience with a smart contract language, such as Solidity

1.1.1.7. Experience with Git and GitHub

1.1.1.8. Technical Writing

1.1.2. Are the core team members well-versed with—and, ideally, active participants in—neighboring communities that will be essential to the project?

A project's owners should consider how the project's needs and its team's skill set overlap. Any gaps should be identified and rectified.

Audit Readiness Checklist

1.0 The Team

1.2 Establish Effective Development Processes

Every successful team needs a way of working together that structures how it makes progress—and guarantees that it does. Appropriate project management controls are an essential part of a successful project and protect against “brain drain” from inevitable turnover.

Consider:

1.2.1. Does the team have an established way to set goals, plan its work, and check on its progress? If so, how long have these controls been in place and how have they been working?

1.2.2. How would the team assign responsibility for interacting with auditors during an audit?

1.2.3. Do the team’s business processes ensure that all team members are aware of the work other team members are doing and the issues they are facing? If a team member left the project, would another team member be able to step in and keep the project moving forward without a steep learning curve?

1.2.4. Is the project’s Git process well organized? Does the team follow specific and established procedures for branching and pull requests that maintain version control in the code?

A project’s owners should consider the ways in which their team plans work, ensures progress, and hedges against turnover. If they have not already done so, they should establish business processes that promote progress and create resiliency.

Audit Readiness Checklist

1.0 The Team

1.3 Manage for Success

The team will require a leader with the skills and temperament necessary to set priorities, manage team members, and resolve problems and conflicts. This role could be an official Project Manager, but might also be a Lead Developer, Chief Technology Officer, or anyone who will take responsibility for managing the project.

Consider:

1.3.1. Does the team have a strong leader with experience leading complex developments?

1.3.2. Does the team's leader have the judgment necessary to prioritize wisely among competing goods?

1.3.3. Does the team's leader have the temperament necessary to engage and motivate team members and other stakeholders?

A project's owners should consider what steps can be taken to enhance the strengths and mitigate the weaknesses of the team's leader. The Manager's Path by Camille Fournier is a good resource on the art and science of technical management.

Audit Readiness Checklist

2.0 The Community

An active and vibrant community is an essential part of the development, adoption, and evolution of any project. In a decentralized ecosystem, all projects build upon and interact with other projects. Not only is community engagement a requisite for a project's adoption, but it is also essential to its security.

2.1 Use a Free Software License

Smart contract coding without community is inherently insecure. If a web3 project's prospective users cannot inspect it, study it, hack it, and experiment on it, they are not going to trust it, either. Closed code makes users vulnerable to a project's owners—and to anyone more capable than those owners. When would-be users have money at stake, closed code is an insurmountable obstacle to project adoption. Transparent and legally open-sourced code is one of the most important steps in fostering community trust in the project.

Consider:

2.1.1. What free software license works best for the project's needs?

2.1.2. What battle-tested open-source code might facilitate the needs of the project?

Project owners should review the Free Software Definition and ensure that the project is using the license that best suits its needs. OpenZeppelin Contracts—the community-vetted standard for smart contract development—are also an excellent resource; they can be easily imported into a project's codebase using the Node Package Manager. (See item 3.1.5.)

2.2 Conduct Community Outreach

Whether or not the code is ready to deploy, it is good to consider how a project will attract and inform its community

Consider:

2.2.1. How will the team market the project?

2.2.2. Who will serve as the project's community manager? Should this be a new hire?

By considering these issues in advance of deployment, a project's owners can create buzz around the project and build its momentum.

Audit Readiness Checklist

2.0 The Community

2.3 Channel Community Input

An audit can build public trust in a project's code and facilitate its adoption. Community growth will create both challenges and opportunities.

Consider:

2.3.1. How will the team channel input from the community?

2.3.2. Should the project establish a community code of conduct and contribution guidelines?

2.3.3. Should the project establish a bug bounty program to incentivize its community to look out for vulnerabilities?

The project's owners should consider how the project team would leverage the advantages of an engaged community. The varied and insightful writings and videos of Jono Bacon are a good resource for thinking about this.

Audit Readiness Checklist

3.0 The Code

Obviously, the most critical aspect of audit-readiness is the code itself. It should be clean, well-tested, and well-documented. Below are some important aspects to consider.

3.1 Develop Clean, Readable, and Modular Code

Clean, readable, and modular code is absolutely essential to success. Good code can be understood simply by reading it. By establishing and following certain rules, developers can promote good, clean code that draws a community of contributors.

Matters to consider:

3.1.1. Does the code follow reasonable naming conventions? Naming should be as straightforward as possible. If a project's naming is not straightforward, it should be explained in a glossary.

3.1.2. Is the code's style consistent throughout the codebase? It should be. It is a good idea to follow Solidity's style guide, but at the very least a project's style should be documented somewhere. Moreover, developers should consider running a linter on every new line of code.

3.1.3. Is the code modular? Ideally, code should be in short and straightforward segments. If functionality is split into multiple modules (such as libraries or other contracts), it should be well encapsulated and the codebase should be navigable without too much jumping around.

3.1.4. Are various operations organized similarly? For both security and readability, it is a good idea to follow similar steps in operations performed. Code should follow the Checks-Effects-Interactions convention. Moreover, failing early and loudly is a recommended practice.

3.1.5. When the project imports external dependencies and libraries, does it do so in a clear, obvious, and well-documented way? When using industry-standard libraries and interfaces, it is always best to import these directly whenever possible, using tools such as the Node Package Manager (NPM), rather than copying and pasting.

Audit Readiness Checklist

3.0 The Code

3.2 Build a Fast, Thorough Test Suite

The soundness of a project's code has a direct relationship with the security of the money involved. For this reason, testing is an essential component of secure development, and project developers should conduct unit tests for all—or nearly all—of a project's code. Failing to test code may be leaving the door open to a costly exploit.

Developers should follow a process of test-driven development, which uses short Agile development cycles to minimize the number and impact of assumptions. The Red-Green-Refactor approach is a recommended best practice. Developers should begin by testing at the unit level, then working their way up to testing at the level of a user, and finally to testing at the level of interactions with other systems.

Reviewing a project's test suite helps an auditor understand the intent behind a project's code, thereby assisting in successful auditing. It should come as no surprise that auditors frequently find code vulnerabilities by calling exposed functions that are absent from the test suite. For this reason, the existence of a comprehensive test suite is a good proxy for the overall quality of a project, and auditors consider this factor when assessing a project's audit-readiness.

Developers should construct a readable, well-structured, and fast test suite. Matters to consider include:

3.2.1. Are all tests executable and successful? Developers should ensure that the test suite and its dependencies are up to date, such that all tests execute and pass on a fresh installation

3.2.2. Does the test suite test for edge cases? Developers should keep in mind that malicious actors often succeed in attacking code by considering ways it is not intended to be run.

Audit Readiness Checklist

3.0 The Code

3.2.3. Does the test suite test against local forks? Developers integrating with multiple projects should run a set of integration tests against local forks of mainnet. Leveraging forking capabilities of existing tools can be a useful approach to testing edge cases. However, developers should note this approach may require more set up and wiring, demanding stronger maintenance efforts. To mitigate, projects should consider a limited set of smoke tests before deployment against a local fork of mainnet.

3.2.4. Does the test suite cover at least 90 percent of the code? While high coverage does not guarantee that the test suite can detect all vulnerabilities, it does indicate that an appropriate level of effort is being devoted to testing.

3.2.5. Does the test suite include helpers? Developers should spend time building and using helpers and utilities. These make their tests more legible and easier to write.

3.2.6. Does the test suite include more advanced tests? For more advanced projects, developers may incorporate a range of more advanced testing tools to achieve a greater degree of confidence in their code's correctness. Such tools may include those for fuzzing and property based testing. These advanced testing approaches are becoming increasingly common.

3.2.7. Does the test suite meet well-known standards? Developers should strive to meet generally accepted testing standards, such as the Smart Contract Security Verification Standards (SVSCS) for G12: Test Coverage.

There are a number of helpful test suite resources. For a short overview, consult Test Driven Development: By Example by Kent Beck. In developing a comprehensive test suite, developers may find that some scenarios are difficult to test; for these situations, it can be valuable to refer to Gerard Meszaros's xUnit Test Patterns. Of course, there is often value in following the examples of others; developers should review testing guidelines shared by others in the ecosystem.

Audit Readiness Checklist

3.0 The Code

3.3 Write Clean, Comprehensive Documentation

While it may not be any developer's favorite part of a project, good documentation is essential to success—and it is the first place auditors will look to understand a project's purpose, features, and inner workings.

Project documentation should be clear, comprehensive, and up-to-date. Specifically, auditors will consider:

3.3.1. Does the documentation achieve all of the following objectives:

3.3.1.1. Does it tell users, contributors, and auditors the intention behind the project?

3.3.1.2. Does it provide detailed explanations of what the code is doing—as well as statements of what it is not doing in any place where omitting such information might cause confusion?

3.3.1.3. Whenever applicable, does it explain why one way of achieving a particular aim was chosen over another way?

3.3.1.4. Whenever applicable, does it make explicit any assumptions made by the developers?

3.3.2. Does the project have a Readme file? A Readme should serve as a straightforward index of its project. It should follow the simple and effective Standard Readme style. It should also include a specific section that states how members of the community should disclose any security vulnerabilities found in the project.

3.3.3. Does the code have rich documentation in every contract, function, event, and variable that documents every function included in the public API? While it is tempting to think that clearly written code does not require documentation, the fact is that protocols for decentralized applications will be called by a wide variety of external agents. For this reason, good docstrings are important. They should follow the NatSpec format. If functions are private/internal, but implement sensitive logic, they should be documented as well.

3.3.4. Does the project have inline comments? In addition to NatSpec, it is a good idea to have inline comments to explain complex functionality and the current status of certain sections of code. It is also important to check for "TODOs" and ensure that they are resolved prior to an audit or large release.

Audit Readiness Checklist

3.0 The Code

3.3.5. Does the project have automated tools to parse and generate external documentation based on the contracts' inline documentation? Updating documentation for protocols that regularly change can be time-consuming. It is always a good idea to make documentation changes part of an automated deployment pipeline, so that when code documentation changes, public-facing documentation is automatically updated as well.

3.3.6. Is there external documentation on the project's website? Such documentation might include more expansive discussion of the project's system architecture, economic incentives, roles, and other relevant design considerations.

3.3.7. Is there documentation on the deployment process for on-chain code changes? For smart contract protocols that make on-chain upgrades, a clear deployment process is essential. (This is true whether upgrades are initiated from a privileged account or a DAO.) The deployment process can have implications for both the security and the functionality of the code. OpenZeppellin Upgrades Plugins are a great resource to help manage upgradeable contracts on Ethereum.

What Audit Clients Can Expect

- Client and audit team agree on audit terms and timing
- Client reviews and executes the audit readiness checklist
- Audit team reviews final version of code prior to commencing audit to ensure the code is ready and fits agreed upon timeline
- Audit team conducts audit
- Audit team provides report of findings
- Client fixes any issues identified
- Audit team reviews fixes, if requested
- Client and audit team publish audit report, if requested

Conclusion

A smart contract audit is a valuable tool in helping Web3 developers to secure their code, but—unlike formal verification or vulnerability testing—an audit is about more than just the code itself. It is about building the trust necessary to attract and engage a thriving community that is willing to invest its time and resources into a project. Fostering such a community requires thoughtful planning and disciplined execution in all aspects of the project, and the checklist above is intended to assist project owners in being comprehensive in their preparations for an audit. A good audit team will discuss each item on the checklist, and more, with a prospective client in order to gauge that client's level of audit-readiness.

Prospective clients who wish to engage OpenZeppelin's team of smart contract experts should fill out the form here. The OpenZeppelin team will review the code submitted and provide a quote and timeline. In the meantime, other OpenZeppelin tools and action items on this checklist will help developers improve a project's code and security posture.