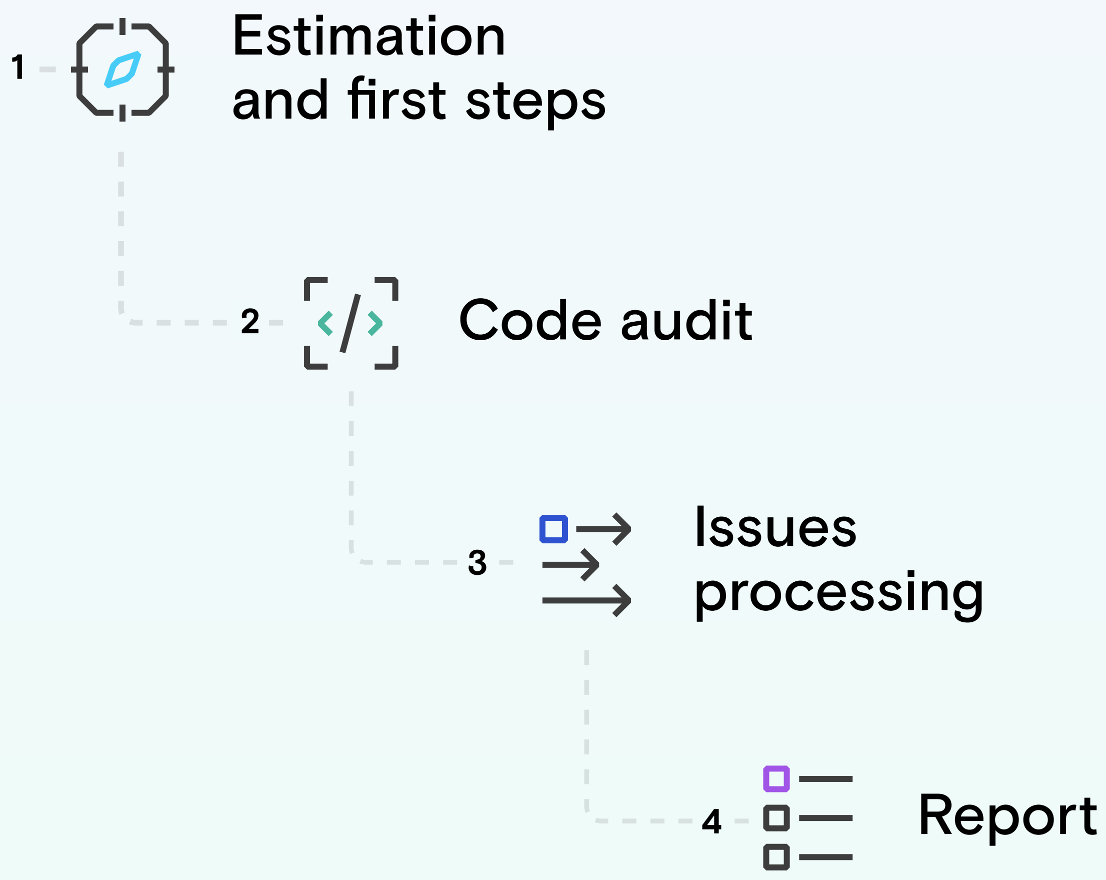
Audit Methodology

— ABDK



Introduction

This methodology applies to various codebases, including smart contracts, regular programs, and more sophisticated constructs like circuits for zero-knowledge frameworks. It's important to note that different strategies may apply to a specific type of code.

1. PHASE

Estimation and first steps

We begin by engaging with Client to explain our process and how it operates. Ultimately, we aim to provide a clear picture of the expected duration and cost.

In order to estimate the overall price and time needed for the audit, we require specific data and information from Client. For an initial estimation we need the size of the codebase, typically measured in lines of code. From there, we apply a default price per line of code. We may adjust this price depending on factors such as the density of the code, which is calculated from the code sample provided by Client. Once Client agrees preliminarily to our terms regarding the price per line and approximate timeline, we discuss the delivery date for the code.

To secure a timeslot, we require an advance payment amounting to 50% of the initial estimate, typically paid 1-1.5 months in advance. If the estimate is revised during the preparation for the audit, the advance payment remains the same. The advance payment is refundable before the audit commences.

We can also offer a Statement of Work, detailing code size, timeline, milestones, and payment schedules. We commit to completing the audit by a set deadline if Client provides the code and requires advance payment by the agreed-upon date. We also ask for supplementary materials to be provided:

- White paper (required)
- Description of the business logic or Yellow paper
- Software requirements specification
- Description of the code architecture

Occasionally we arrange a call close to the audit's starting date to clarify any issues found in the documentation. During this meeting Client can also elaborate on the intended functionality of the code. Once we get the information, we start with the audit.

2. PHASE

Code audit

While each codebase is unique, we typically follow several approaches that work for most of our clients. We look into the supplementary materials for the code, including the documentation and the white paper, and assess their readability. If there are any problems with it, we ask for clarification. Then we proceed to the code analysis. It can be split into two parts: dynamic code analysis and static code analysis.

The static code analysis deals with simpler issues like documentation errors, comments, etc. We offer recommendations to enhance code practices specific to the programming language, suggesting simplification and more. These comments aim to improve code readability and enhance the overall audit quality.

The dynamic code analysis is crucial as it involves simultaneous examination of documentation and code to identify and analyze issues comprehensively. We initially focus solely on the code, followed by a process akin to reverse engineering to compare our findings with specified requirements, including access controls. We meticulously examine access controls to ensure proper implementation for internal and external access, evaluate control levels, user roles, permissions, and asset protection mechanisms. Special attention is given to objects handling user assets, ensuring alignment with intended business logic. We then delve into the internal logic of individual functions, assessing correctness and efficiency. Occasionally, we offer code snippets to assist clients. We compile a comprehensive report documenting various issues, including object definition, access levels, and code readability, as well as concerns related to access control, efficiency, and correctness. We also evaluate the use of appropriate data structures, algorithms, and external libraries for relevance to the code's tasks.

If any part of the code is unclear or if a critical issue is discovered, we promptly request further details via a private chat. Usually, this isn't for an immediate fix, as there may be numerous other issues to address, but to enable Client to investigate any related mistakes or potential implications. This is especially important if a similar issue exists in a previously deployed contract, as it allows for mitigation of any associated risks.

The code is audited by at least two independent auditors who cross-review each other's work to enhance error detection. This ensures that we always have a secondary opinion on the most critical code issues.

As part of our process, Client is provided with a spreadsheet containing a list of identified issues.

3. PHASE

Issues processing

Each issue comes with a code snippet extracted from a file where we identified an issue.

Sometimes, we expand on the most important issues by providing additional context where we deem it appropriate. This reflects examples of potentially malicious behaviors by adversaries. We may provide a code snippet to help improve the efficiency of the code. Certain issues are explained in greater detail, and they are ranked from the most critical to the least severe.

Critical issues directly affect the code functionality and may cause a significant loss of assets.

Major issues are either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behavior which should be double checked.

Moderate issues are not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.

Recommendations contain code style, best practices and other suggestions.

Then we request Client to apply all necessary fixes, after which we review them collectively and address **critical** and **major** issues at no additional cost. Once the fixes are implemented, we examine the updated code and compare it with the previous version to assess how critical and major issues were resolved.

If there are minor fixes to review or new functionality added to the code during or after the audit process, we request an additional, appropriately priced, smaller audit to address these changes.

At times, Client may disagree with the severity level we have assigned to an issue. This may happen if we have overestimated the importance of a bug or overlooked an additional check that Client is aware of and informs us about. For several issues, ongoing discussion about severity and importance may be necessary.

There are 3 possible outcomes:

- Client acknowledges the issue and its importance, then fixes it
- Client provides evidence that the issue exists but its effect is much less severe. Then the severity is downgraded
- Client provides evidence that the issue is non-existent. Then it is removed

Clients are also welcome to accompany any non-fixed issue with a comment providing an additional context.

Finally we send an invoice for the second part of the payment. It is necessary to pay this invoice in order to get the final report. Once the comments are prepared and all issues are resolved, we are ready to proceed with the final audit report.

4. PHASE

Report

The report is provided free of charge.

We typically require 2-3 days to craft a report. It comprises existing issues (some of those elaborated) and recommendations, client comments, and code snippets. Should we need more time to produce a report, we promptly inform Client to facilitate their planning.

Client is free to share the report publicly or keep it private. In the former case, which is most typical, we ask to add it to our public report database and possibly announce it on social networks.

Our most captivating audit cases for smart contracts:

- Github.com/abdk-consulting/audits/blob/main/uniswap/ABDK_Uniswap_UniversalRouter_v_2_0.pdf
- Github.com/abdk-consulting/audits/blob/main/contango/ABDK_Contango_CoreV2_v_2_0.pdf

ZK circuits:

- Github.com/abdk-consulting/audits/blob/main/nil_foundation/ABDK_NilFoundation_BlockchainVerifier_v_2_0.pdf
- Github.com/abdk-consulting/audits/blob/main/blockswap/ABDK_Blockswap_RPBS_v_3_0.pdf

Combined case:

- Github.com/abdk-consulting/audits/blob/main/zklink/ABDK_zkLink_CircuitsSmartContracts_v_6_0.pdf