Security Guidance of auditing smart contract on Blockchain and Distributed Ledger system

Technical Report · April 2024

DOI: 10.13140/RG.2.2.21273.02405

Christophe Ozcan
C4A - CRYPTO4ALL

Security Guidance of auditing smart contract on Blockchain and Distributed Ledger system.

Author: Christophe André Ozcan (Crypto4All & Standard ISO/TC307 Expert from AFNOR)

Contact: christophe.ozcan@crypto4all.com

Date: 31 mars 2024

I. Scope

Auditing Smart Contract third party evaluate the security risks of deploying protocols using smart contracts. To review and verify the project specifications and source code with a detailed focus on weaknesses, potential vulnerabilities, and overall security the procedure of findings with solutions that may mitigate future attacks or loopholes must be provided by auditors.

The mission of this document is to define the different types of approaches and detections, ranging from manual, static, and dynamic analysis, as well as formal verification, to ensure that a protocol using smart contracts is checked against known attacks and common potential vulnerabilities.

A smart contract audit involves security experts to scrutinize the source code created to underwrite the functions of the smart contract often called a decentralized protocol.

Smart contract audits are usually conducted by a third-party company to ensure that the source code is reviewed as thoroughly as possible. Depending on the complexity of the smart contract, companies may choose to engage the services of a specialist smart contract team to conduct the audit without being sure that the auditing process is well conducted.

The importance of getting the smart contract code correct and secure before it is deployed is very important even more due to the immutability of blockchain and distributed ledger system. The implications of activating a smart contract that has not been properly audited could be severe for any projects.

The contribution helps to the emerging literature on audit data analytics (ADA) by proposing a new approach involving audit methodology, audit analytic tools and smart audit procedures which are enabled by blockchain technology. Besides, this contribution presents a discussion regarding the effect of smart audit procedures on audit quality and the public/private interest regarding the role of emerging technologies in the traditional system audit process bring by a new emerging cybersecurity market.

TABLE OF CONTENTS

II. Methodology

A Smart Contract Audit (SCA) is an audit of a distributed ledger system involving smart contract operations and related control processes. It aims to ensure the security, reliability, compliance, performance, and interoperability of smart contracts. Auditing a distributed system like a smart contract differs from auditing a centralized system due to the unique characteristics of peer-to-peer networks, distributed ledgers, Distributed Virtual Machines (DVMs), and consensus mechanisms.

The objectives of a smart contract audit include evaluating the reliability of data from smart contracts that impact financial statements, assessing the effectiveness of smart contract governance controls, ascertaining compliance with applicable laws, policies, and existing standards, and ensuring the performance and interoperability of the smart contract within the broader ecosystem.

The audit process follows the following steps:

❖ Evaluate the reliability of data from smart contract which have an impact on financial statements.

❖ Evaluate effectiveness of Smart contract governance controls to ensure the distributed systems are functioning as intended.

❖ Ascertain compliance with applicable laws, policies, and existing standards.

1. Audit Preparation: Agreeing on a specification document.

The audit process begins by agreeing on a detailed specification document. This document outlines the project's functional requirements, smart contract architecture, structural choices, and logical build process. It serves as a reference for auditors to ensure that the smart contract works as intended. The specification should include information on variables, functions, and their interactions within the contract.

- Gather all relevant information: Provide auditors with the smart contract code, technical specifications (readme file ...), and any relevant documentation. The more information the audit has, the better he can understand the code purpose and potential vulnerabilities.
- Define the scope of the audit: Specify which parts of the smart contract to audit (e.g., entire contract, specific functionalities). Focus on critical areas like financial transactions, access control, governance, performance and sensitive data handling.
- Set clear expectations: Outline the desired deliverables from the audit (e.g., detailed report, severity levels of vulnerabilities). This ensures both parties are on the same page about the audits.
- Auditors review the project's documentation, including specifications, design documents, test plans, and implementation details. They verify that the documentation accurately reflects the implemented code and that it aligns with industry best practices.

2. Security Level of References

Severity levels in audit reports typically classify security findings into distinct tiers, commonly identified as Critical, High, Medium, Low and Informational.

To standardize the security evaluation made by auditors, we define the following terminology:

◊ Likelihood represents how likely a particular vulnerability is to be uncovered and exploited.
◊ Impact measures the technical loss and damage of a successful attack.
◊ Severity demonstrates the overall criticality of the risk.

Likelihood and impact are categorized into four ratings: High, Medium, Low, Informational respectively.

When determining severity, auditor is considering various factors, including:

1) **Potential Impact:** Assessing the worst-case outcome of a vulnerability, such as the potential for catastrophic loss of all funds or unintentional compromise of user security, which may be deemed as low severity.

2) **Scope of Impact:** Evaluating whether the vulnerability poses a risk to the overall security of the entire application (critical) or if it only affects individual users (less medium).

3) **Attacker Incentive:** Analyzing the cost-benefit ratio for an attacker. If the effort required to exploit a vulnerability outweighs the potential gains, severity may decrease from critical to high. However, this doesn't negate the existence of a vulnerability, as non-economic motives for attacks (e.g., showcasing skills, personal vendettas) may still be at play.

4) **Complexity of Exploitation:** Distinguishing between vulnerabilities exploitable by unsophisticated attackers versus those requiring deep knowledge of sophisticated attacks involving a specific attack scenario. The more intricate the attack, the less likely it is for an attacker to exploit it.

The final rate is determined by likelihood and impact and can be classified into severity categories accordingly to the following classification table. Every finding is assigned a severity level from the following classification table.

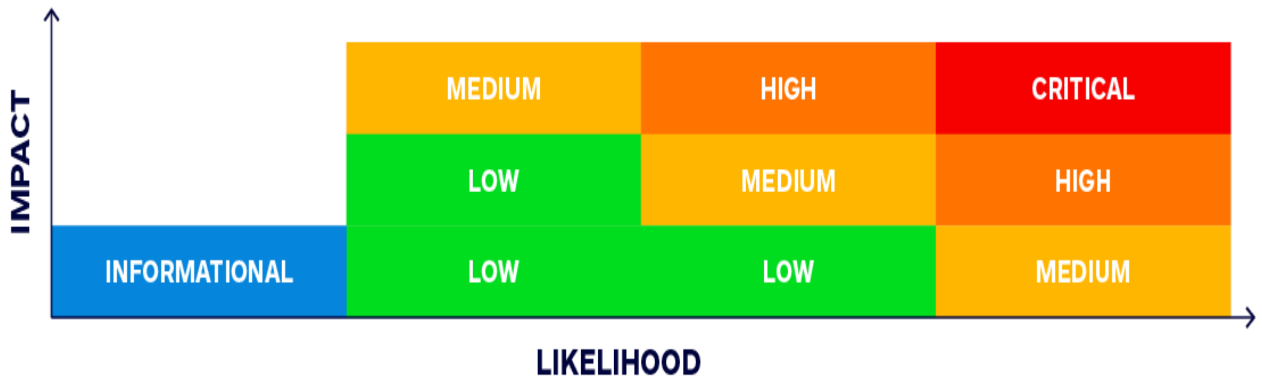

Figure 1 - Security Severity Classification Table

1. List of Vulnerabilities and checkpoints:

Vulnerability	Category	Checkpoints
Arithmetical function	Functional	Integer underflow/overflow
		Floating Points and Decimal Precision
Access & Privilege Control	Structural	Administrative functionality for control and emergency handling
		Restriction access for sensitive functions and data
		Ownership management of the contract
		Rate limit for critical operations, permission to contract state changes, and delay operations for malicious/sensitive actions
		Variable Limiting
		Check Effect Interaction Pattern
Denial of Service	Network	Unexpected Revert
		Unbounded operations or block stuffing
		Unbounded Loops
Miner or Node manipulation	Network	Block Number Dependence
		Timestamp Dependence
		Transaction Ordering Or Front-Running
External Referencing	Structural	Correct usage of the pull over push favor for external calls
		Correct usage of checks-effects-interactions pattern to minimize the state changes after external contract or call referencing
		Avoid state changes after external calls
		Error handling and logging events
		Fallback function security
Race Conditions	Structural	Reentrancy - unexpected state changes
		Cross-function racing - attacks that using different functions while share the same state
Low-level Call	Structural	Code Injection by delegate call
		Unsuited adoption on assembly code
Visibility	Functional	Specify the correct external visibility of variables and functions.

		Interface implementation facilitating the external smart contract interaction.
Proxy	Structural	Specify the upgradeability management of the smart contract code in the case of an emergency patch or version should be deployed.
Incorrect Interface	Functional	Ensure the defined function signatures match with the contract interface and implementation
Execution consumption	Structural	Ensure that the execution of the smart contract is not consuming a lot of energy or gas to not impact the efficiency of the smart contract and the entire distributed network.
Dependencies	Structural	Using updated smart contract dependencies for which the smart contract is referring
Transaction Ordering	Structural	Reception of Transactions calling different functions which could interfere on the state of the smart contract causing unexpected results and potential security exploits
Unexpected reception of funds	Structural	Smart contract could receive unexpected cryptoassets without having the possibility to withdraw them if it was not implemented causing potential fund loss.
Ownership key management	Human Management	A owner of smart contract having some privileged to interact with private functions have a hug risk to lost his private key impacting the ownership and more often the state of the smart contract itself depending on his manual external call actions to change the state of a value.
Old Compiler	Functional	A good practice is to use always before coding a smart contract the last updated compiler version allowing to resolve some common issues and helping the developer to use some last recommendations (depreciated function …)

2. Manual or Static Analysis Review

Auditors perform a manual or static analysis of the smart contract's source code. This involves reading and analyzing the code line by line to identify potential bugs, vulnerabilities, and areas where defensive programming practices can be applied. The analysis focuses on security-oriented code review, checking for common vulnerabilities and flaws. It also includes a review of code dependencies to ensure their security and relevance to the audited smart contract.

3. Automated Analysis

Security engineers and researchers use automated analysis tools to increase the chances of detecting flaws and critical risksplaying a crucial role in auditing smart contracts. These tools utilize various techniques, including static analysis, symbolic execution, and dynamic analysis, to scan the smart contract code for known patterns of vulnerabilities.

Such tools are using by auditors to evaluate the security code without having a standard to evaluate them. However, a comparative evaluation of automated analysis tools for solidity smart contracts[1] research paper can help auditors to select the relevant tools depending on their audit scope.

4. Formal Verification Analysis

Formal verification involves performing an automated mathematical proof that the source code fulfills a certain formal specification. It helps ensure the correctness of core components and identifies any discrepancies between the technical specification and the actual implementation. By using formal mathematical proofs, auditors can validate critical properties of the smart contract, such as correctness, safety, and security.

5. Dynamic Analysis:

Smart contracts are tested in a simulated or controlled environment to observe their behavior during runtime. Different transactions and inputs are executed to identify any unexpected outcomes, vulnerabilities, or performance issues. Dynamic analysis helps assess the contract's behavior under various scenarios and validate its intended functionality.

6. Security Testing:

Apart from the source code review, security testing techniques such as penetration testing and vulnerability scanning are employed. These tests simulate real-world attack scenarios to identify weaknesses or vulnerabilities that could be exploited by attackers. Penetration testing involves actively trying to exploit vulnerabilities to gain unauthorized access or manipulate the contract's behavior. Vulnerability scanning scans the contract for known security vulnerabilities and weaknesses.

7. Integration and Interoperability Testing:

[1] A Comparative Evaluation of Automated Analysis Tools for Solidity Smart Contracts

Smart contracts often interact with external systems, APIs, or other smart contracts. Integration and interoperability testing assesses how the smart contract integrates and interacts within the broader ecosystem to identify any compatibility issues or vulnerabilities. This testing ensures that the contract can effectively communicate and exchange data with other systems or contracts without compromising security or functionality.

8. Performance Evaluation:

The performance of the smart contract is evaluated in terms of response times, transaction throughput, resource consumption, and scalability. This evaluation helps identify bottlenecks, performance limitations, and inefficiencies. It ensures that the contract can handle the expected workload and operates efficiently within the distributed network.

9. Continuous Monitoring:

After the initial audit, continuous monitoring of the smart contract is essential to identify and address any emerging security threats or vulnerabilities. Regular reviews and updates to the contract's security controls, codebase, and compliance with evolving standards are crucial to maintaining the contract's security over time.

The use of monitoring security tools should be recommended allowing companies to integrate it by default.

10. Compliance Review:

The smart contract is evaluated for compliance with applicable laws, regulations, and industry standards. Auditors ensure that the contract meets specific legal requirements, data protection regulations, financial industry standards, or other relevant regulations. This review helps ensure that the smart contract aligns with the regulatory landscape and mitigates legal risks.

11. Remediations and Recommendations trough a report

The audit process concludes with the issuance of a final report. This report outlines all critical, medium, and low findings and provides actionable items and upgrade suggestions. Security engineers, with their expertise in software engineering and security, outline ways to mitigate vulnerabilities and enhance the overall security of the smart contract. The report also includes recommendations for improvements in areas such as code quality, error handling, access control, and compliance with industry standards.

III. External publications references

- ReJection: A AST-Based Reentrancy Vulnerability Detection Method, Rui Ma, Zefeng Jian, Guangyuan Chen, Ke Ma, Yujia Chen - CTCIS 19
- MPro: Combining Static and Symbolic Analysis forScalable Testing of Smart Contract, William Zhang, Sebastian Banescu, Leodardo Pasos, Steven Stewart, Vijay Ganesh - ISSRE 2019
- ETHPLOIT: From Fuzzing to Efficient Exploit Generation against Smart Contracts, Qingzhao Zhang, Yizhuo Wang, Juanru Li, Siqi Ma - SANER 20
- Verification of Ethereum Smart Contracts: A Model Checking Approach, Tam Bang, Hoang H Nguyen, Dung Nguyen, Toan Trieu, Tho Quan - IJMLC 20
- Smart Contract Repair, Xiao Liang Yu, Omar Al-Bataineh, David Lo, Abhik Roychoudhury - TOSEM 20
- A Comparative Evaluation of Automated Analysis Tools for Solidity Smart Contracts, Zhiyuan Wei, Jing Sun, Zijian Zhang*, Xianhao Zhang, Meng Li*, Liehuang Zhu