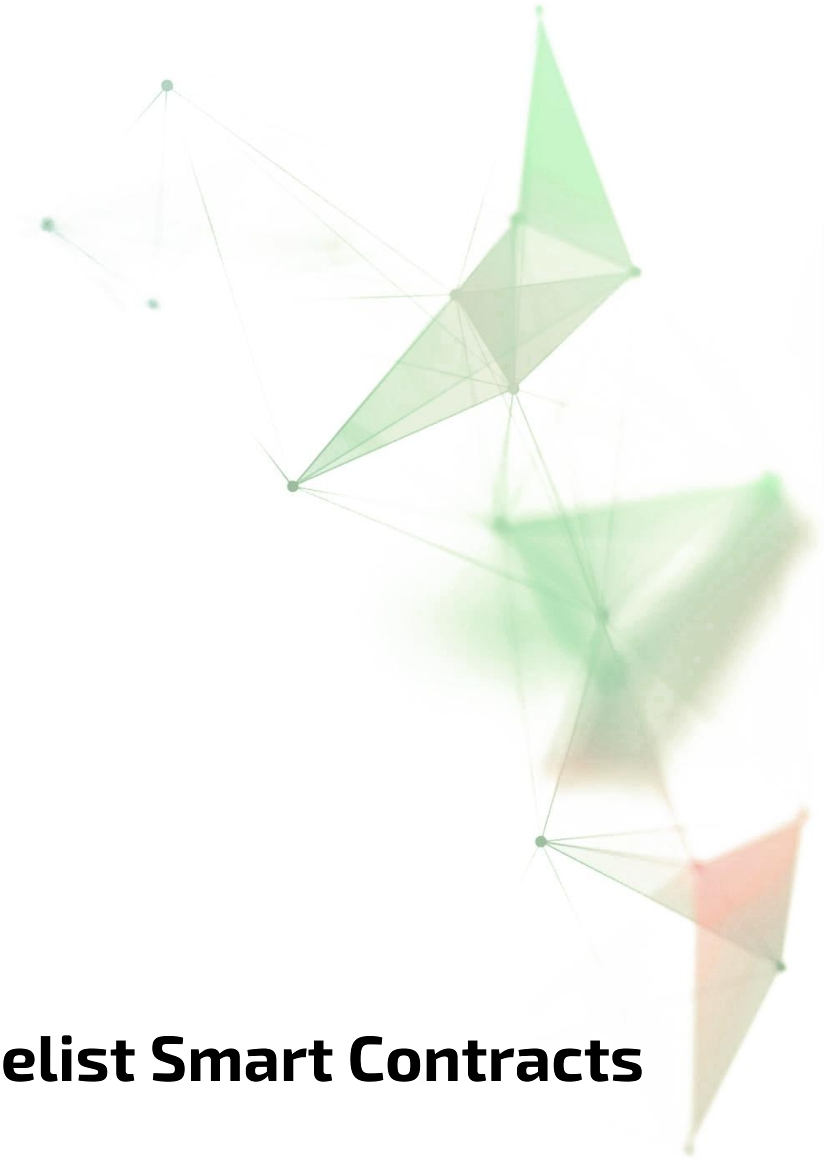
apriorit



Whitelist Smart Contracts

HIGHLY SECURE
AUDITED BY
APRIORIT

Security audit report

Prepared for Nomadic Labs

January 11, 2022

Table of contents

Project summary

Name	Whitelist Smart Contracts	
Source	**Repository**	**Revision**
	https://gitlab.com/tezos-paris-hub/whitelist-smart-contract-archetype	9b84b0a096d30c8bf19ddc077e48991e2633db97
Methods	Code review Behavioral analysis Unit test coverage analysis Manual penetration testing	

Coverage and scope of work

The audit focused on an in-depth analysis of the implementation of the smart contracts, including:

- contract/users_storage.arl
- contract/whitelist.arl

We conducted the audit in accordance with the following criteria:

- Behavioral analysis of smart contract source code
- Checks against our database of vulnerabilities and manual attacks against the contract
- Symbolic analysis of potentially vulnerable areas
- Manual code review and code quality evaluation
- Unit test coverage analysis

The audit was performed using manual code analysis. Once potential vulnerabilities were discovered, manual attacks were performed to check if they could be easily exploited.

Smart contract overview

Whitelist Smart Contracts were developed using Archetype language for the Tezos blockchain. The contract allows to manage NFT minting and transfer and to restrict interaction with Ubisoft contracts to whitelisted users only.

The Whitelist contract interacts with the Users Storage contract, where the actual user records are stored.

Executive overview

Apriorit conducted a security assessment of Whitelist Smart Contracts in January 2022 to evaluate its current state and risk posture, evaluate exposure to known security vulnerabilities, determine potential attack vectors, and check if any can be exploited maliciously.

Summary of strengths

Building upon the strengths of the available implementation can help better secure it by continuing these good practices. In this case, a number of positive security aspects were readily apparent during the assessment:

- The code and project files are well structured which makes them easy to read and maintain. The code is self-explanatory. The naming policy makes instructions understandable
- Main execution flow was covered with unit tests
- Verification errors have a custom explanation
- The contracts are developed using an up-to-date Archetype compiler
- Cross-contract interaction is performed securely
- The code implemented securely without any known vulnerability

Summary of discovered vulnerabilities

During the assessment, no vulnerabilities were discovered, indicating good attention to security in the smart contract implementation.

Apriorit described additional recommendations in Code review and Test coverage analysis sections for findings that do not create vulnerabilities.

Security rating

Apriorit reviewed Nomadic Labs security posture in regards to the Whitelist Smart Contracts, and Apriorit consultants identified security strengths as well as the absence of vulnerabilities. Taken together, the combination of asset criticality, threat likelihood, and vulnerability severity have been used to assign a grade for the overall security of the application. An explanation of the grading scale is included in the second table below.

In conclusion, Apriorit recommends that Nomadic Labs continue to follow existing good security practices.

	High	**Medium**	**Low**	**Security**	**Grade**
Whitelist Smart Contracts	0	0	0	Highly Secure	A

Security grading criteria

Grade	Security	Criteria description
A	Highly secure	Exceptional attention to security. No high- or medium-risk vulnerabilities and few minor low-risk vulnerabilities.
B	Moderately secure	Good attention to security. No high-risk vulnerabilities and only a few medium- or several low-risk vulnerabilities.
C	Marginally secure	Some attention to security, but security requires improvement. A few high-risk vulnerabilities that can be exploited.
D	Insecure	Significant security gaps exist. A large number of high-risk vulnerabilities.

Code review and recommendations

Based on years of software development experience, Apriorit had formed a list of best practices to write clear and understandable code. Following these best practices makes maintenance easier.

During the assessment, smart contracts code was compared against our list of best practices. As a result of the code review, we formed the following recommendations.

1. **Update readme file**

   The current description in the readme file does not correspond to the currently implemented logic but to the previous version. It is important to keep documentation up-to-date simultaneously with the changes in the code.

2. **Implement users storage replacement**

   When working with a solution that consists of multiple smart contracts, it is recommended to implement the logic of updating smart contract addresses. That way, in case of the need to replace one of the contracts, others could be left as it is, only setting the address to the new contract in their storage.

   It is recommended to add an entrypoint in the Whitelist contract for updating the Users Storage contract address.

Test coverage analysis

Unit tests are an essential part of smart contract development. They help to find problems in the code that are missed by the compiler before deploying the contract to the blockchain.

During the audit, the percentage of unit test coverage for each of the contracts was evaluated. The results are presented in the table below.

Contract	Coverage
users_storage	39%
whitelist	70%

The main execution flow was fairly covered, but it is recommended to also cover exceptional scenarios.

Users Storage contract uncovered test cases

Function	Description	Status
declare_ownership	as admin	OPEN
	as not admin	OPEN
claim_ownership	as candidate	OPEN
	as not candidate	OPEN
	without candidate	OPEN
add_whitelister	as not admin	OPEN
rm_whitelister	as admin	OPEN
	as not admin	OPEN

Whitelist contract uncovered test cases

Function	Description	Status
declareOwnership	as admin *(the current test is invalid)*	OPEN
claimOwnership	as candidate	OPEN
	as not candidate	OPEN
	without candidate	OPEN
pause	as admin	OPEN
	as not admin	OPEN
	already paused	OPEN
unpause	as admin	OPEN
	as not admin	OPEN
	already unpaused	OPEN
addWhitelister	as admin	OPEN
	as not admin	OPEN
	duplicate value	OPEN
removeWhitelister	as admin	OPEN
	as not admin	OPEN
	non existent value	OPEN
updateUser	as whitelister *(the user under whitelister variable is admin)*	OPEN
	paused contract	OPEN
updateUsers	as whitelister *(the user under whitelister variable is admin)*	OPEN
	paused contract	OPEN
updateTransferlist	paused contract	OPEN

Appendixes

Appendix A. Detailed findings

Risk rating

Our risk ratings are based on the same principles as the Common Vulnerability Scoring System. The rating takes into account two parameters: exploitability and impact. Each of these parameters can be rated as high, medium, or low.

Exploitability — What knowledge the attacker needs to exploit the system and what preconditions are necessary for the exploit to work:

- **High** — Tools for the exploit are readily available and the exploit requires no specialized system knowledge.
- **Medium** — Tools for the exploit are available but have to be modified. The exploit requires specialized knowledge about the system.
- **Low** — Custom tools must be created for the exploit. In-depth knowledge of the system is required to successfully perform the exploit.

Impact — What effect will the vulnerability have on the system if exploited:

- **High** — Administrator-level access and arbitrary code execution or disclosure of sensitive information (private keys, personal information)
- **Medium** — User-level access with no disclosure of sensitive information.
- **Low** — No disclosure of sensitive information. Failure to follow recommended best practices does not result in an immediately visible exploit.

Based on the combination of parameters, an overall risk rating is assigned to a vulnerability.

Appendix B. Description of methodologies

Smart contract security checks

Apriorit uses a comprehensive and methodical approach to assess the security of blockchain smart contracts. We take the following steps to find vulnerabilities, expose weaknesses, and identify deviations from accepted best practices in assessed applications. Notes and results from these testing steps are included in the corresponding section of the report.

Our security audit includes the following stages:

1. **Discovery**. The first step is to perform reconnaissance and information gathering to decide how resources can be used in the most secure way. It is important to obtain a thorough understanding of the smart economics, the logic of smart contracts, and the environment they operate within so tests can be targeted appropriately. Within this stage, the following tasks are done:

    a. Identifies technologies

    b. Analyzes the specification, whitepaper, and smart contract source base

    c. Creates a map of relations among smart contracts

    d. Researches the structure of smart contract storage

    e. Researches and analyzes standard implementations for functionality

2. **Configuration Management.** The configuration of the smart contracts is analyzed.

3. **User management and user permissions.** The majority of smart contracts have to manage individual users and their permissions. Most smart contracts split permissions between the contract owner, administrator, etc. Within this stage, the following are done:

   a. Determines whether all functions can be called only by the expected role

   b. Reviews user management functions and role assignment

   c. Reviews permissions for each role

4. **Data validation**. Inputs to a smart contract from users or other smart contracts are its operational life-blood but are also the source of most high-risk attacks. These steps ensure that data provided to the application is treated and checked. All cases of invalid or unexpected data should be handled appropriately.

5. **Efficiency check**. Each function uses some amount of GAS during the call. In the case of a GAS shortage or overlimit, the smart contract function call will fail. Chipper smart contracts will be more interesting for users because no one wants to waste money, so all functions should be optimized in terms of GAS use.

6. **High-quality software development standards**. The standard requires teams to follow the best practices of coding standards. This will help to avoid or mitigate the most common mistakes during development that lead to smart contract security vulnerabilities. It will also help with traceability and root cause analysis. This stage includes:

   a. Manual code review and evaluation of code quality

   b. Unit test coverage analysis

7. **Internal function protection.** Contract vulnerabilities are often introduced due to the semantic gap between the assumptions that contract developers make about the underlying execution semantics and the actual semantics of smart contracts. The internal function should not be callable from the outside because it can lead to inconsistency or unacceptable changes in the storage. Within this stage, the following known vulnerabilities are checked:

   a. The unauthorized contract function call by view callback

   b. The unauthorized intermediate function call.