Cyberscope

Audit Report

Wateract®

March 2024

SHA256 565e5a5e1d3781d583654d446f539960be223fa51ac9aa046562313baabc0908

Audited by © cyberscope

Table of Contents

Review

Network	Algorand

Audit Updates

Initial Audit	10 Mar 2024

Source Files

Filename	SHA256
contratto.py	565e5a5e1d3781d583654d446f539960be223fa51ac9aa046562313baabc0908

Overview

Wateract's smart contract is designed for the Algorand blockchain. The primary objective of the audited smart contract is to manage and regulate the distribution of a specific Algorand Standard Asset (ASA), identified by the asset ID 613277011, under a predefined set of rules. The smart contract introduces mechanisms to control how the ASA is requested and distributed to the creator's account, ensuring a structured and regulated asset flow based on the contract's logic.

The smart contract encapsulates two main functionalities: `withdraw` and `optin`. Each functionality serves a distinct purpose in the asset management process, as detailed below:

Withdraw Functionality

The withdraw function allows the contract's creator to request a specified amount of the ASA from the contract's balance. This functionality is subject to two critical regulations:

- A limitation on the amount of ASA that can be withdrawn per request, capped at 10 million tokens.
- A temporal restriction that mandates a minimum interval of 30 days between consecutive withdrawal requests.

Optin Functionality

The optin function facilitates a one-time setup process by which the smart contract's associated wallet "opts in" to receive the ASA.

Both functionalities are exclusively accessible by the contract's creator, reinforcing a strict access control model that centralizes the ability to initiate these operations. This access control mechanism is critical for maintaining the integrity and security of the contract's operations, ensuring that only authorized actions are executed.

Testing

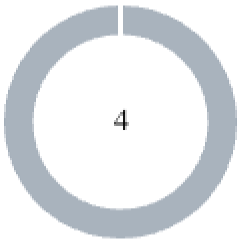
The contract was successfully compiled and deployed using Dappflow's integrated toolset, ensuring transparency and reliability in the deployment process. Throughout this audit, the contract's code structure, functionality, security considerations, and adherence to best practices were thoroughly evaluated. The primary objective was to identify potential vulnerabilities, optimize code efficiency, and ensure compliance with industry standards.

The audit process involved a detailed review of the contract's source code, including its initialization, methods, access controls, error handling mechanisms, and any other relevant components. Additionally, special attention was given to aspects such as gas optimization, modularity, and parameterization to enhance the contract's flexibility and usability.

By conducting this audit, our aim is to provide actionable insights and recommendations to enhance the overall robustness and reliability of the smart contract, thereby mitigating potential risks and promoting secure blockchain interactions.

The following sections of this report will delve into specific findings, recommendations, and observations derived from the audit process, ultimately contributing to the ongoing refinement and optimization of the Wateract smart contract.

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	4

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	4	0	0	0

Diagnostics

🔴 Critical 🟠 Medium ● Minor / Informative

Severity	Code	Description	Status
●	CR	Code Repetition	Unresolved
●	EEH	Enhanced Error Handling	Unresolved
●	LFE	Legacy Function Exit	Unresolved
●	VIO	Variable Initialization Optimization	Unresolved

CR - Code Repetition

Criticality	Minor / Informative
Location	contratto.py#L24,37
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using such code segments. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
Assert(Txn.sender() ==
Addr("XXXIN7CSV7SKPYNLB3QILIFWUOZXOJ7G4T7LDP7VGUNE65N2D7XTQX7ZFU"))
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in a function in order to avoid repeating the same code in multiple places.

EEH - Enhanced Error Handling

Criticality	Minor / Informative
Location	contratto.py
Status	Unresolved

Description

The contract currently lacks descriptive error messages, which can make it challenging for users to understand why a transaction failed. By incorporating the Reject method with informative error messages, users can receive clearer feedback on issues such as unauthorized access or violation of withdrawal limits.

Recommendation

To improve user experience and facilitate troubleshooting, the team could implement more descriptive error messages using the Reject method in the contract. For instance, provide specific messages when the sender is not the creator of the contract or when withdrawal frequency or amount limits are not adhered to. This enhancement will enhance the contract's usability and reduce user confusion in case of transaction failures. Additionally, ensure that error messages are informative yet concise to convey the relevant information effectively.

LFE - Legacy Function Exit

Criticality	Minor / Informative
Location	contratto.py#L33,50,55,56
Status	Unresolved

Description

The Algorand smart contract specifies version 6, which supports the usage of Approve and Reject expressions for transaction approval or rejection. However, the contract still utilizes the older convention of using Return(Int(1)) instead of Approve and Return(Int(0)) instead of Reject.

```
Return(Int(1))
```

Recommendation

To align with best practices and ensure clarity in code interpretation, it's recommended to update the contract to utilize the Approve and Reject expressions explicitly instead of relying on the Return expressions. This enhances readability and maintains consistency with the specified program version. Additionally, thorough testing should be conducted after making these changes to ensure the contract functions as intended.

VIO - Variable Initialization Optimization

Criticality	Minor / Informative
Location	contratto.py#L24,37,47
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract initializes the variables multiple times within its code. This redundancy could lead to increased gas costs and code maintenance overhead.

```
Addr("XXXIN7CSV7SKPYNLB3QILIFWUOZXOJ7G4T7LDP7VGUNE65N2D7XTQX7ZFU")
Int(613277011)
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it. The team could initialize the variables only once and reuse them throughout the contract. This can be achieved by assigning the value to a variable outside of the functions and referencing that variable wherever it is needed. Additionally, using descriptive variable names can enhance code readability and reduce the risk of errors.

Summary

Wateract's smart contract is crafted to serve as a controlled mechanism for the distribution of a specific ASA on the Algorand blockchain, with a focus on ensuring regulated access and distribution according to predefined rules. The smart contract audit report found no critical findings.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io