TENSET

Smart Contract Audit

Algoworld Swapper

Table of Content

Executive Summary

"Algoworld Swapper" introduces a cutting-edge solution within the blockchain landscape, powered by the Algorand blockchain. This innovative platform focuses on revolutionizing asset exchange and management through a decentralized, secure, and efficient ecosystem.

Built on Algorand's high-performance blockchain infrastructure, the "Algoworld Swapper" project offers users the ability to seamlessly swap digital assets with ease. Leveraging the Algorand blockchain's advanced features, such as fast transaction speeds and robust security, the platform enables quick and reliable asset swapping.

Key features of the "Algoworld Swapper" include user-friendly asset swaps, secure transactions, and a transparent environment. Users can trust the platform's integrity, backed by the Algorand network's renowned security protocols.

The "Algoworld Swapper" project aims to provide users with a streamlined and intuitive experience for exchanging digital assets, contributing to the broader adoption of decentralized finance (DeFi) solutions. By embracing the power of blockchain technology, it empowers users to take control of their assets while ensuring the highest standards of security.

In summary, the "Algoworld Swapper" project showcases a forward-thinking approach to digital asset management and exchange, leveraging Algorand's strengths to offer a secure and efficient platform. Its focus on user-friendliness, security, and transparency positions it as a significant player in the evolving landscape of blockchain-based asset swapping.

Project Summary

Language	Codebase	Commits
Python	https://github.com/AlgoWorldNFT/algoworld-contracts	f02bcc0fe2ff3f37153d9318e6ca0dd94b4de7c8

Audit Scope

ID	File	SHA256
AS	algoworld_contracts/swapper/asa_to_asa_swapper.py	2bfe0d954d1fdeedb2969ddf57a50a3af51b9e036c01522efdf2fad789c1d56f

AL	algoworld_contracts/swapper/asas_to_algo_swapper.py	46e798be6e886a3eb4e3d49b04bca8ffee6be164aaccb6bab3929faec7f18031
SP	algoworld_contracts/swapper/swap_proxy.py	2b04c1a053e02e8b8448dffd4a92a391fde462d542726893ff8f59413d3a2725

General security assumptions

In conducting our security audit of the Algoworld Swapper, we have made a number of important security assumptions. These assumptions are fundamental to our analysis and should be considered when evaluating the overall security and potential vulnerabilities of the system:

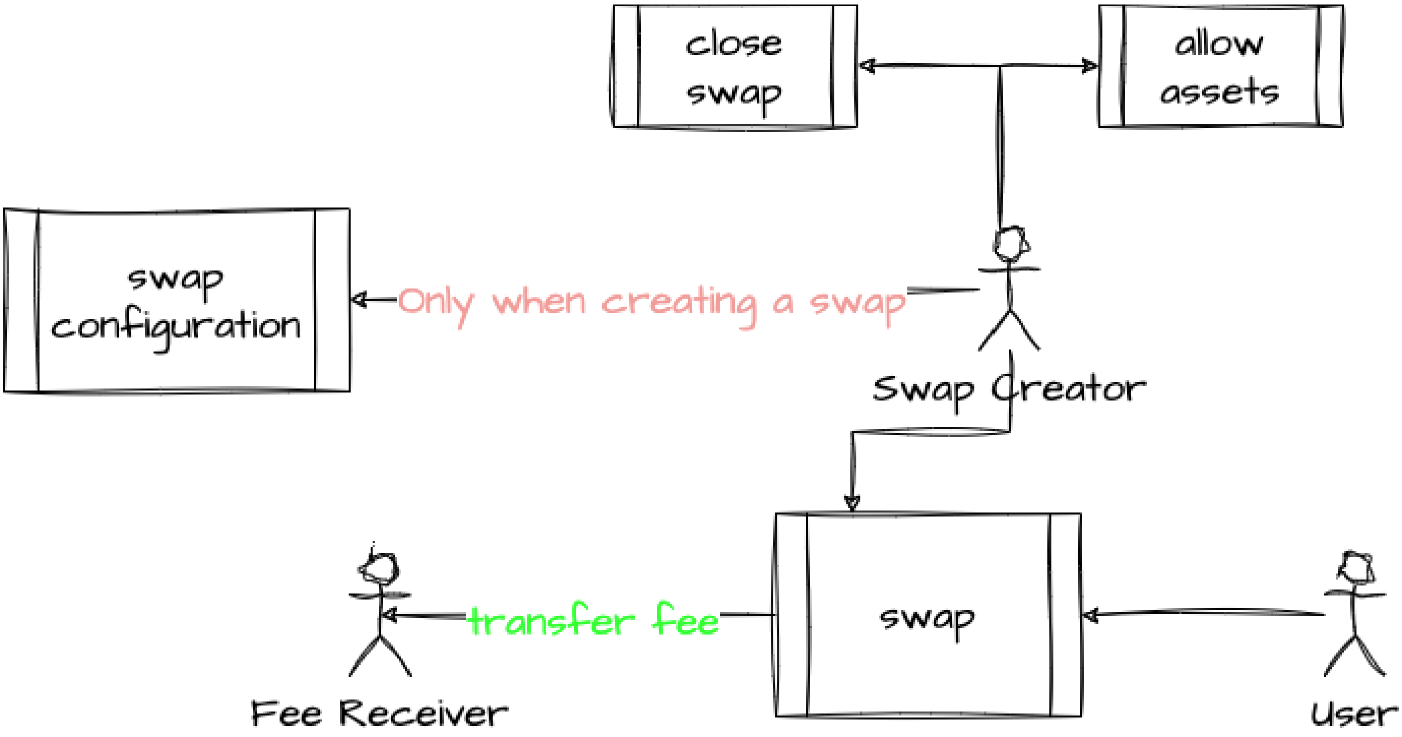
Security of the Underlying Blockchain
Our security analysis is predicated on the assumption that the underlying blockchain (for example, Algorand) is secure and functions as intended. Potential vulnerabilities or flaws in the blockchain protocol are outside the scope of this audit.

Trusted Environment
We assume that users of the smart contract will interact with it in a secure and trusted environment. This encompasses the use of secure, up-to- date software and hardware, which is free from malware or any potential interference from malicious third parties.

Private Key Security
Users of the smart contract are assumed to safeguard their private keys effectively. This involves not sharing private keys, securely storing key backups, and using hardware wallets or other secure means for key management.

These security assumptions form the basis of our audit. Deviations from these assumptions could potentially lead to risks or vulnerabilities not covered in this analysis. Therefore, the effective management of these factors is critical to ensure the ongoing security of the Algoworld Swapper.

Actors in the system



In our examination of the Algoworld swapper, we've identified several key actors, each with their distinct roles, responsibilities, and potential influence within the protocol. These actors are as follows:

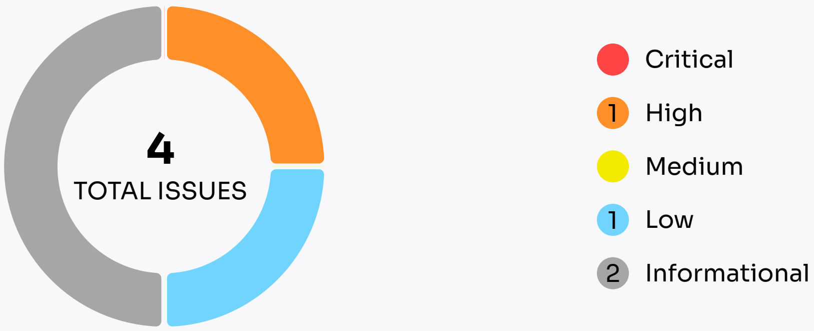
Users
These are individuals who interact with the smart contract to perform various actions, such as opting in to offered ASAs, initiating asset swaps, and closing swap transactions. Users initiate transactions and interact with the contract's functionalities.

Swap Creator
This actor is responsible for creating the swap transactions. They set up the parameters for the swap, including the offered ASA, requested ASA, incentive fees, and other relevant details. The swap creator can also initiate the closure of a swap.

Fee Receiver
This actor is the designated recipient of the incentive fee associated with the swap. The incentive fee is paid as part of the swap transaction and is meant to reward participants in the swapping process.

Findings

Audit Overview



Issues

Severity	Found	Resolved	Partially Fixed	Acknowledged
Critical	0	0	0	0
High	0	1	0	0
Medium	0	0	0	0
Low	1	0	0	0
Informational	2	0	0	0
Total	**3**	**1**	**0**	**0**

AL-01
Takeover of The Escrow Account.

● High ✓ Resolved

Description

An inherent security vulnerability has been detected in the 'swap' transaction group of the smart contract. This vulnerability potentially enables an attacker to exploit the contract by acquiring both the contract logic and signed transactions, then resubmitting the transactions with altered configuration parameters. This could lead to unauthorized actions within the contract. The absence of validation for the 'rekeyTo' field in the 'incentive_fee' and 'requested_algo_amount' payment transactions, along with the lack of sender verification, creates an avenue for an attacker to manipulate these transactions, potentially leading to funds theft.

Recommendation

To address this vulnerability, a robust validation mechanism must be implemented within the 'swap' transaction group. It is imperative to validate the 'rekeyTo' and 'closeRemainderTo' fields in the 'incentive_fee' and 'requested_algo_amount' payment transactions. Moreover, stringent sender verification should be employed to ensure that the sender corresponds to the expected asset receiver.

By implementing comprehensive validation for the 'rekeyTo' and 'closeRemainderTo' fields and enforcing sender verification, the contract can significantly mitigate the risk of unauthorized rekeying and asset theft. These measures act as a safeguard, thwarting potential malicious activities and fortifying the overall security posture of the smart contract.

AS-02
Minimum Fee Hard-Coding.

● Low Found

Description

The smart contract has hard-coded the minimum transaction fee as 1,000 microAlgos in L41. However, it's important to note that the minimum fee is a consensus parameter that can be modified over time. Hard-coding the fee may lead to issues if the minimum fee is adjusted in the future. Smart contracts or code relying on a fixed fee value may fail or behave unexpectedly when the minimum fee changes.

Recommendation

To ensure the stability and resilience of the smart contract, it is advised not to hard-code specific values for the minimum transaction fee. Instead, utilize appropriate methods or functions provided by the blockchain platform to dynamically retrieve the current minimum fee. This practice will enable the smart contract to adapt seamlessly to changes in the consensus parameters, ensuring its functionality remains intact.

GLOBA1-01
Limited Usage of PyTeal.

● Informational Found

Description

The current implementation exclusively utilizes PyTeal for the smart contract. While PyTeal is a powerful tool for developing Algorand smart contracts, considering the potential benefits of incorporating Beaker could enhance the contract's capabilities and security.

Recommendation

Consider expanding the smart contract development approach to include Beaker, a versatile and advanced tool for Algorand smart contract development. Beaker offers additional features and functionalities that could provide more robust security mechanisms, optimize gas usage, and potentially simplify the contract logic. By exploring the use of Beaker in conjunction with PyTeal, the smart contract's overall quality and security could be improved.

AS-AL-01
Lack of Numeric Literal Separators.

● Informational Found

Description

The smart contract does not utilize underscores to separate digits of numeric literals, potentially impacting code readability and increasing the risk of errors when dealing with large numbers.

Recommendation

To enhance code readability and reduce the likelihood of errors when working with numeric literals, consider incorporating underscores to separate groups of digits in these literals. This practice enhances code clarity and makes it easier to parse and understand large numbers at a glance.

Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, Consultant cannot guarantee the explicit security of the audited smart contracts.