SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

Swift

AUDIT

SECURITY ASSESSMENT

21. September, 2023

FOR

Swift

Introduction

SolidProof.io is a brand of the officially registered company MAKE Network GmbH, based in Germany. We're mainly focused on Blockchain Security such as Smart Contract Audits and KYC verification for project teams.
Solidproof.io assess potential security issues in the smart contracts implementations, review for potential inconsistencies between the code base and the whitepaper/documentation, and provide suggestions for improvement.

Disclaimer

SolidProof.io reports are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team. SolidProof.io do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'...)

SolidProof.io Audits do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technology proprietors. SolidProof Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

SolidProof.io Reports represent an extensive auditing 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. SolidProof's position is that each company and individual are responsible for their own due diligence and continuous security. SolidProof in no way claims any guarantee of the security or functionality of the technology we agree to analyze.

Project Overview

Summary

Project Name	Swift
Website	https://swiftbot.io/
About the project	Swift Trade crypto trading application that can be used on both Telegram and the downloadable application. Both services provide users with a user-friendly interface, which makes it easy for users to navigate through the applications.
Chain	Ethereum
Language	Solidity
Codebase Link	Provided as Files
Commit	N/A
Unit Tests	Not Provided

Social Medias

Telegram	https://t.me/Swifttokenportal
Twitter	https://twitter.com/SwiftEcosystem
Facebook	N/A
Instagram	https://www.instagram.com/swift.eco/
Github	N/A
Reddit	N/A
Medium	https://medium.com/@Swiftadmin/
Discord	https://discord.com/invite/swift-ecosystem
Youtube	https://www.youtube.com/@SwiftEcosystem
TikTok	https://www.tiktok.com/@swift.ecosystem
LinkedIn	N/A

Audit Summary

Version	Delivery Date	Changelog
v1.0	19. September 2023	• Layout Project • Automated- /Manual-Security Testing • Summary
v1.1	21. September 2023	• Reaudit

Note - The following audit report presents a comprehensive security analysis of the smart contract utilized in the project. This analysis did not include functional testing (or unit testing) of the contract/s logic. We cannot guarantee 100% logical correctness of the contract as we did not functionally test it.

File Overview

The Team provided us with the files that should be tested in the security assessment. This audit covered the following files listed below with an SHA-1 Hash.

File Name	SHA-1 Hash
contracts/SimulateTax.sol	18449c98182e093cc8df999cc1f5fd7754fc7a9f
contracts/FeeDist.sol	51988b7eb927aee5c716783e6d97f0481f01f618
contracts/AntiMEVSwap.sol	c079839488dca484d8940932f8d17ac751239058
contracts/multicall.sol	0d01bdbc72af6cf7ed7b7d4c7bc90f4ecff2e73e

Please note: Files with a different hash value than in this table have been modified after the security check, either intentionally or unintentionally. A different hash value may (but need not) be an indication of a changed state or potential vulnerability that was not the subject of this scan.

Imported packages

Used code from other Frameworks/Smart Contracts (direct imports).

Dependency / Import Path	Count
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	1
@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol	1
@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol	1
@openzeppelin/contracts/token/ERC20/IERC20.sol	1
@openzeppelin/contracts/utils/Strings.sol	1

Note for Investors: We only audited contracts mentioned in the scope above. All contracts related to the project apart from that are not a part of the audit, and we cannot comment on its security and are not responsible for it in any way

Audit Information

Vulnerability & Risk Level

Risk represents the probability that a certain source threat will exploit vulnerability and the impact of that event on the organization or system. The risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 - 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon aspossible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to check the repository for security-related issues, code quality, and compliance with specifications and best practices. To this end, our team of experienced pen-testers and smart contract developers reviewed the code line by line and documented any issues discovered.

We check every file manually. We use automated tools only so that they help us achieve faster and better results.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
    a. Reviewing the specifications, sources, and instructions provided to SolidProof to ensure we understand the size, scope, and functionality of the smart contract.
    b. Manual review of the code, i.e., reading the source code line by line to identify potential vulnerabilities.
    c. Comparison to the specification, i.e., verifying that the code does what is described in the specifications, sources, and instructions provided to SolidProof.

2. Testing and automated analysis that includes the following:
    a. Test coverage analysis determines whether test cases cover code and how much code is executed when those test cases are executed.
    b. Symbolic execution, which is analysing a program to determine what inputs cause each part of a program to execute.

3. Review best practices, i.e., review smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on best practices, recommendations, and research from industry and academia.

4. Concrete, itemized and actionable recommendations to help you secure your smart contracts.

Overall Security

Upgradeability

Contract is an upgradeable	❌ Deployer can update the contract with new functionalities
Description	The deployer can replace the old contract with a new one with new features. Be aware of this, because the owner can add new features that may have a negative impact on your investments.
Example	We assume that you have funds in the contract and it has been audited by any security audit firm. Now the audit has passed. After that, the deployer can upgrade the contract to allow him to transfer the funds you purchased without any approval from you. This has the consequence that your funds can be taken by the creator.
Comment	N/A

Ownership

The ownership is not renounced	❌ **The owner is not renounce**
Description	The owner has not renounced the ownership that means that the owner retains control over the contract's operations, including the ability to execute functions that may impact the contract's users or stakeholders. This can lead to several potential issues, including: • Centralizations • The owner has significant control over contract's operations
Comment	N/A

Note - If the contract is not deployed then we would consider the ownership to be not renounced. Moreover, if there are no ownership functionalities then the ownership is automatically considered renounced.

Ownership Privileges

These functions can be dangerous. Please note that abuse can lead to financial loss. We have a guide where you can learn more about these Functions.

Minting tokens

Minting tokens refer to the process of creating new tokens in a cryptocurrency or blockchain network. This process is typically performed by the project's owner or designated authority, who has the ability to add new tokens to the network's total supply.

Contract owner cannot mint new tokens	✅ **The owner cannot mint new tokens**
Description	The owner is not able to mint new tokens once the contract is deployed.
Comment	N/A

Burning tokens

Burning tokens is the process of permanently destroying a certain number of tokens, reducing the total supply of a cryptocurrency or token. This is usually done to increase the value of the remaining tokens, as the reduced supply can create scarcity and potentially drive up demand.

Contract owner cannot burn tokens	✅ **The owner cannot burn tokens**
Description	The owner is not able burn tokens without any allowances.
Comment	N/A

Blacklist addresses

Blacklisting addresses in smart contracts is the process of adding a certain address to a blacklist, effectively preventing them from accessing or participating in certain functionalities or transactions within the contract. This can be useful in preventing fraudulent or malicious activities, such as hacking attempts or money laundering.

Contract owner cannot blacklist addresses	✅ **The owner cannot blacklist addresses**
Description	The owner is not able blacklist addresses to lock funds.
Comment	N/A

Fees and Tax

In some smart contracts, the owner or creator of the contract can set fees for certain actions or operations within the contract. These fees can be used to cover the cost of running the contract, such as paying for gas fees or compensating the contract's owner for their time and effort in developing and maintaining the contract.

Contract owner cannot set fees more than 25%	✅ **The owner cannot levy unfair taxes**
Description	The owner is not able to set the fees above 25%
Comment	N/A

Lock User Funds

In a smart contract, locking refers to the process of restricting access to certain tokens or assets for a specified period of time. When tokens or assets are locked in a smart contract, they cannot be transferred or used until the lock-up period has expired or certain conditions have been met.

Owner cannot lock the contract	✅ **The owner cannot lock the contract**
Description	The owner is not able to lock the contract by any functions or updating any variables.
Comment	N/A

External/Public functions

External/public functions are functions that can be called from outside of a contract, i.e., they can be accessed by other contracts or external accounts on the blockchain. These functions are specified using the function declaration's external or public visibility modifier.

State variables

State variables are variables that are stored on the blockchain as part of the contract's state. They are declared at the contract level and can be accessed and modified by any function within the contract. State variables can be defined with a visibility modifier, such as public, private, or internal, which determines the access level of the variable.

Components

📝 Contracts	📚 Libraries	🔍 Interfaces	🎨 Abstract
4	0	3	0

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

🌐 Public	💰 Payable
73	25

External	Internal	Private	Pure	View
56	37	0	6	23

StateVariables

Total	🌐 Public
12	8

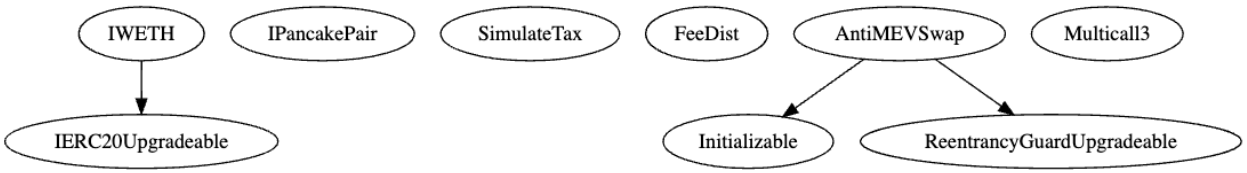
Capabilities

Solidity Versions observed	📤 Transfers ETH	💰 Can Receive Funds	👥 Delegate Call	💣 Has Destroyable Contracts

`>=0.8.0`
`^0.8.0`
`>=0.7.0`
`0.8.12`
	Yes	Yes	Yes	

Inheritance Graph

An inheritance graph is a graphical representation of the inheritance hierarchy among contracts. In object-oriented programming, inheritance is a mechanism that allows one class (or contract, in the case of Solidity) to inherit properties and methods from another class. It shows the relationships between different contracts and how they are related to each other through inheritance.



Centralization Privileges

Centralization can arise when one or more parties have privileged access or control over the contract's functionality, data, or decision-making. This can occur, for example, if the contract is controlled by a single entity or if certain participants have special permissions or abilities that others do not.

In the project, there are authorities that have access to the following functions:

File	Privileges

1. FeeDist.sol	• onlyOwner
	  • Transfer/Renounce ownership
	  • Set Fee Items

2. AntiMEVSwap.sol	• onlySupervisor
	  • Set Supervisor address
	  • Set Fee and Bribe Information (receiver address and amount)
	  •

Recommendations

To avoid potential hacking risks, it is advisable for the client to manage the private key of the privileged account with care. Additionally, we recommend enhancing the security practices of centralized privileges or roles in the protocol through a decentralized mechanism or smart-contract-based accounts, such as multi-signature wallets.

Here are some suggestions of what the client can do:

- Consider using multi-signature wallets: Multi-signature wallets require multiple parties to sign off on a transaction before it can be executed, providing an extra layer of security e.g. Gnosis Safe
- Use of a timelock at least with a latency of e.g. 48-72 hours for awareness of privileged operations
- Introduce a DAO/Governance/Voting module to increase transparency and user involvement
- Consider Renouncing the ownership so that the owner cannot modify any state variables of the contract anymore. Make sure to set up everything before renouncing.

Audit Results

Critical issues

No critical issues

High issues

No high issues

Medium issues

#1 | Missing Access Control

File	Severity	Location	Status
AntiMEVSwap	Medium	L83—93	ACK

Description - The functions lack an access control modifier, which will result in the public use of these functions. Moreover, any arbitrary user can claim/withdraw the complete balance of the contract regardless of the approval. The users can also withdraw the funds that are deposited by some other user because, in the contract, there is no mapping of the amount which a certain user will be depositing

Remediation - We recommend putting access control modifiers in the functions to prevent such attacks. For example, the users should only be able to withdraw the funds they have deposited, not the complete balance.

Alleviation - The contract also works as a router and doesn't store any funds, and modifiers would hinder the functionality of the contract. The chance of occurrence is highly unlikely as the contract doesn't hold any funds and never will.

Low issues

#1 | Missing Dead Address Validation

File	Severity	Location	Status
AntiMEVSwap	Low	L48, 55	Open

Description - Make sure to validate that the address passed in the function parameters is not the dead address; otherwise, the fees will be credited to the dead address.

#2 | Missing Events

File	Severity	Location	Status
AntiMEVSwap	Low	All	Open

Description - Make sure to emit events for all the critical parameter changes in the contract to ensure the transparency and trackability of all the state variable changes.

Informational issues

#1 | NatSpec documentation missing

File	Severity	Location	Status
AntiMEVSwap	Informational	N/A	Open

Description - If you started to comment on your code, comment on all other functions, variables, etc.

#2 | Disable initializing

File	Severity	Location	Status
AntiMEVSwap	Informational	L29	Open

Description - If the owner updates the contract, a disableInitializer call in the constructor must be implemented. This prevents calling the initialize function again to set the state variables in the contract. This should be implemented only if the contract was deployed before. Otherwise, the owner cannot call the initialize function to set the variables.

Recommendation
If the contract hasn't been deployed, remove the disableInitializer in the constructor. Otherwise, you are not able to initialize the contract. When the contract has a deployed version already, leave it as it is.

#3 | Floating Pragma

File	Severity	Location	Status
AntiMEVSwap FeeDist SimulateTax	Informational	N/A	Open

Description - The contracts should be deployed with the same compiler version and flag that they have been tested thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using other versions.

Legend for the Issue Status

Attribute or Symbol	Meaning
Open	The issue is not fixed by the project team.
Fixed	The issue is fixed by the project team.
Acknowledged(ACK)	The issue has been acknowledged or declared as part of business logic.

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY