SOLIDEX SECURITY AUDIT REPORT

March 21, 2022

MixBytes()

TABLE OF CONTENTS

1. Introduction 2

1.1. Disclaimer 2

1.2. Security Assessment Methodology 3

1.3. Project Overview 6

1.4. Project Dashboard 6

2. Findings Report 8

2.1. Critical 8

2.2. High 8

2.3. Medium 8

M-1 Spoofed Initial Deposits In `SexPartners.Sol` And `LpDepositor.Sol` 8

2.4. Low/Informational 9

L-1 Same Storage Variable Accessed Multiple Times 9

L-2 Usage Of Magic Numbers 10

3. About Mixbytes 11

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Customer. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals
- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the cients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Customer with an interim report.

5. Bug fixing & re-audit:

- The Customer either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Customer with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Customer with a re-audited report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss funds to be transferred to any party.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low/Informational	Other non-essential issues and recommendations reported to/acknowledged by the team.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

Solidex is a protocol to optimize yield, voting power, and liquidity provisioning on Solidly. Solidly is a decentralized exchange (AMM) where tokens may be swapped using liquidity provided by users.

Contract	Description
FeeDistributor.sol	Handles claiming fees and bribes earned by SEX lockers.
LpDepositor.sol	Stores the Solidex veNFT and handles core logic for farming Solidly.
LpDepositToken.sol	ERC-20 that is deployed for every gauge that is deposited into. Typical deposit receipt as a token wrapper.
SexPartners.sol	Allows launching NFT recipients to bootstrap Solidex by depositing their NFTs to become partners.
SolidexVoter.sol	Handles voting on SOLID emissions to Solidly gauges.
StakingRewards.sol	Stake SOLIDsex and claim rewards, a minimal mod of the SNX rewards contract.
Token.sol	SEX token. Standard ERC-20 contract.
TokenLocker.sol	Handles locking of SEX token in multiple locks of 1-16 weeks.
VeDepositor.sol	ERC-20 with extra functionality that mints SOLIDsex if it receives SOLID or a veNFT.
Whitelister.sol	Tokenizes the Solidly whitelisting process into an ERC-20.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Solidex
Project name	Solidex
Timeline	February 21, 2022 - March 21, 2022
Number of Auditors	5

Project Log

Date	Commit Hash	Note
21.02.2022	91c69f3ea3da607e432dda9f8264f7dd77f74b46	Private repository supplied for audit
23.02.2022	d35e07961a1df908bf733ff1dd432dbef777869b	Minor fixes supplied
18.03.2022	8b420ed8bed4b714695d51de2a0f82e38a72e1b2	Final commit at the public repository

Project Scope

The audit covered the following files:

File name	Link
FeeDistributor.sol	FeeDistributor.sol
LpDepositor.sol	LpDepositor.sol
LpDepositToken.sol	LpDepositToken.sol
SexPartners.sol	SexPartners.sol
SolidexVoter.sol	SolidexVoter.sol
StakingRewards.sol	StakingRewards.sol
Token.sol	Token.sol
TokenLocker.sol	TokenLocker.sol
VeDepositor.sol	VeDepositor.sol
Whitelister.sol	Whitelister.sol

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	0
Medium	1
Low/Informational	2

ID	Name	Severity	Status
M-1	Spoofed initial deposits in `SexPartners.sol` and `LpDepositor.sol`	Medium	Acknowledged
L-1	Same storage variable accessed multiple times	Low/Informational	Acknowledged
L-2	Usage of magic numbers	Low/Informational	Acknowledged

1.6 Conclusion

During the audit process, 1 MEDIUM and 2 LOW/INFORMATIONAL severity findings were spotted and acknowledged by the developers. These findings do not affect security of the audited project.

File name	Contract deployed on mainnet
FeeDistributor.sol	0xA5e76B97e12567bbA2e822aC68842097034C55e7
LpDepositor.sol	0x26E1A0d851CF28E697870e1b7F053B605C8b060F
LpDepositToken.sol	0xa4a4ca6a18f3f92615ac96f7e2ec2d7e746bc73e
SexPartners.sol	0x24c0e3b0eA69bd967d7ccA322801be7Cd53586a2
SolidexVoter.sol	0xca082181C4f4a811bed68Ab61De6aDCe11158948
StakingRewards.sol	0x7FcE87e203501C3a035CbBc5f0Ee72661976D6E1
Token.sol	0xD31Fcd1f7Ba190dBc75354046F6024A9b86014d7
TokenLocker.sol	0xDcC208496B8fcc8E99741df8c6b8856F1ba1C71F
VeDepositor.sol	0x41adAc6C1Ff52C5e27568f27998d747F7b69795B
Whitelister.sol	0xb7714B6402ff461f202dA8347d1D7c6Bb16F675d

2.FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

M-1	Spoofed initial deposits in `SexPartners.sol` and `LpDepositor.sol`
Files	SexPartners.sol#L95 LpDepositor.sol#L351
Severity	Medium
Status	Acknowledged

Description

The **initial** deposit of NFT is vulnerable to a spoofing attack.
The attacker can directly call `onERC721Received` and supply NFT id that was not actually transferred/not owned by the attacker. It may cause a contract malfunction, including contract unusability or incorrect calculation of the rewards. A contract malfunction cannot be fixed without a contract replacement.

Please note, this attack can only affect a contract at the initial stage, before the initial deposit was made. Spoofing attempts for any deposits except the initial deposit will be reverted or will not change the state of the contract. Thus a contract at the production stage (after the initial deposit) is not vulnerable anymore.

Location of the affected code:

- SexPartners.sol#L95
- LpDepositor.sol#L351

Recommendation

Although an attack can be easily detected and mitigated by redeploying of the contracts, we recommend to add explicit checks for the known attack vector.

Client's commentary

We will not add explicit checks for this already known issue for the reason that it is only applicable on the initial deposit. Since it is trivial to redeploy the contract to fix it, it would be inefficient and non-sensical to carry out explicit checks on every deposit in perpetuity to protect against this. By design, we prioritize gas savings for our end users here and we were ready to redeploy should this have occured on the first deposit of either contract.

2.4 Low/Informational

L-1	Same storage variable accessed multiple times
File	Token.sol#L43
Severity	Low/Informational
Status	Acknowledged

Description

`balanceOf[_from]` is accessed twice at Token.sol#L43

`allowance[_from][msg.sender]` is accessed 3 times at Token.sol#L76

Recommendation

In favour of gas savings, values can be cached in memory.

L-2	Usage of magic numbers
File	
Severity	Low/Informational
Status	Acknowledged

Description

Some numeric constants are used without comments. It decreases the readability of the code.

Recommendation

We recommend describing numeric constants in code and/or in comments.

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts


https://github.com/mixbytes/audits_public


https://mixbytes.io/


hello@mixbytes.io


https://twitter.com/mixbytes