SOLIDIFIED

Audit Report for XIO Network - December 04, 2020

Summary

Audit Report prepared by Solidified covering the protocol smart contracts (and their associated components).

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The debrief took place on November 30th and December 1st, 2020, and the results are presented here.

Audited Files

The following contracts were covered during the audit:

```
contracts
├── FlashProtocol.sol
├── interfaces
│   ├── IFlashProtocol.sol
│   ├── IFlashReceiver.sol
│   └── IFlashToken.sol
└── libraries
    ├── Address.sol
    └── SafeMath.sol
```

```
contracts
├── FlashApp.sol
├── interfaces
│   ├── IERC20.sol
│   ├── IFlashProtocol.sol
│   └── IFlashReceiver.sol
├── libraries
│   ├── Address.sol
│   ├── Create2.sol
│   └── SafeMath.sol
└── pool
    ├── contracts
    │   ├── Pool.sol
    │   └── PoolERC20.sol
    └── interfaces
        └── IPool.sol
```

SOLIDIFIED

Audit Report for XIO Network - December 04, 2020

```
contracts
├── FlashToken.sol
├── interfaces
│   ├── IERC20.sol
│   ├── IFlashMinter.sol
│   └── tests
│       └── IFlashMint.sol
├── libraries
│   └── SafeMath.sol
└── tests
    └── FlashMinter.sol
```

Supplied in the following source code repositories:

https://github.com/XIO-Network/xio-flash-token

commit number `8c2ce92887166bbd17a908e0609a755d3b047a94`

Update commit `0a976e22d29d9051f6c6797848fbe0c7ed728f50`

https://github.com/XIO-Network/xio-flash-protocol

commit number `8bfdc8f500bc25065bd07f766120f3efcf425573`

Update commit `d4b85921d4eeebd97fdcf1734eb9a78abbe01c53`

https://github.com/XIO-Network/xio-flashapp-contracts

commit number `ffd26b70a0a7c43bb04c4f292e7cdb43486bd1e9`

Update commit `f9bcbba6671e80484478b533f788b56885f085a3`

Intended Behavior

The smart contract implements the following functionality:

- A flashmintable ERC-20 token
- A corresponding staking protocol
- A staking pool

SOLIDIFIED

Audit Report for XIO Network - December 04, 2020

Executive Summary

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium	-
Level of Documentation	Medium	-
Test Coverage	N/A	It has not been possible to assess the test coverage adequately, since automated unit tests have not been submitted to the audit. However, the team has held a pre-audit bug bounty program.

Issues Found

Solidified found that the Flash Protocol contracts contain no critical issue, 1 major issue, and 4 minor issues, in addition to 2 informational notes.

We recommend all issues are amended, while the notes are up to the team's discretion, as it refers to best practices.

Issue #	Description	Severity	Status
1	Reentrancy issues in pool contract allow malicious tokens or ERC777 tokens to manipulate balances	Major	Resolved
2	ERC-20 Return Values Ignored	Minor	Resolved
3	FlashToken.sol: Incorrect check of balance in flashMint	Minor	Resolved
4	Potential problems with some ERC tokens	Minor	Acknowledged
5	Token susceptible to approve attack	Minor	Acknowledged
6	FlashToken.sol: minter role unnecessary	Note	Resolved
7	Tautology in precondition check	Note	Resolved

Critical Issues

No critical issues have been found.

Major Issues

1. Reentrancy issues in pool contract allow malicious tokens or ERC-777 tokens to manipulate balances

The pool code, based on Uniswap v2, does not implement a locking mechanism that prevents tokens to call back into the pool functions on `transfer()` or `transferFrom()` calls.
This means that the pool is vulnerable to malicious token implementations or ERC-777 tokens that provide a hook allowing user code to be executed.
One prominent example of this is the `burn()` function in `Pool.sol`.

Recommendation
Implement a locking mechanism that prevents re-entrancy. For example, the `lock` modifier used in the original Uniswap code:

```
modifier lock() {
        require(unlocked == 1, 'UniswapV2: LOCKED');
        unlocked = 0;
        _;
        unlocked = 1;
}
```

Additionally, it is recommended that all state changes are performed at the end of functions.

Update
Fixed.

Minor Issues

2. ERC-20 Return Values Ignored

Throughout the codebase return values of ERC-20 calls are not checked. While many ERC-20 tokens revert on error, this is not required by the standard and some tokens may return false instead of reverting. This is not an issue when interacting with the FashToken but could lead to problems with the external tokens managed in liquidity pools.

Recommendation
Wrap all ERC-20 calls with `require` statements.

Update
Fixed.

3. FlashToken.sol: Incorrect check of balance in flashMint

In the function `flashMint()` there's a requirement to check whether the flash minted amount plus the current balance is larger than the max `uint112`. However, this operation is adding the ETH balance of the contract with the `flashAmount` in tokens.

Recommendation
Adjust the requirement to account for the flashToken balance.

Update
Fixed.

4. Potential problems with some ERC tokens

The protocol is designed to be used with any compliant ERC20 tokens, but tokens come in multiple implementations and some of them could cause unintended behaviours. For example, some tokens charge fees on transfers, are deflationary in another way or are simply malicious, which adhere to the ERC20 interface but do not implement it as expected.

Recommendation

This is a difficult problem to solve and it's a commonly exploited weakness of some protocols, like Uniswap. There's no clear recommendation on how to solve this issue at the smart contract level. We have chosen to report it in this audit, so the XIO team can be aware of it and make a decision considering its trade-offs. Users should be made aware of the risk of malicious or incompatible tokens and UI-level checks could be added.

Update
The team acknowledges the issue and chooses, like most projects, to not deal with this as the smart contract layer.

5. Token susceptible to approve attack

Changing an allowance through the `approve()` method brings the risk that someone may use both the old and the new allowance by unfortunate transaction ordering.
A detailed description of this vulnerability can be found here:
https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM

Recommendation
One possible solution to mitigate this race condition is to implement `increaseAllowance` and `decreaseAllowance` functions.

Update
The team acknowledges the issue and chooses, like most projects, to not deal with this as the smart contract layer.

Notes

6. FlashToken.sol: minter role unnecessary

In FlashToken.sol a MinterRole is maintained. However, this cannot be used, since there is no provision for adding or removing minters, which are set in the constructor.

Recommendation
Consider simplifying the code for readability.

7. Tautology in precondition check

In `FlashProtocol.sol` (line 95) the following precondition check is performed:

```
require(_newMatchRatio >= 0 && _newMatchRatio <= 2000, "FlashProtocol::
INVALID_MATCH_RATIO");
```

However, the parameter `_newMatchRatio` is of type `uint256`, which is `>= 0` by definition.

Recommendation
Remove unnecessary check for clarity.

Update
Fixed.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of XIO Network or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Solidified Technologies Inc.