CERTIK

CertiK Audit Report for Swipe-Network

Contents

Disclaimer

About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK, in partnership with grants from IBM and the Ethereum Foundation, CertiK's mission of every audit is to apply different approaches and detection methods, ranging from manual, static, and dynamic analysis, to ensure that projects are checked against known attacks and potential vulnerabilities. CertiK leverages a team of seasoned engineers and security auditors to apply testing methodologies and assessments to each project, in turn creating a more secure and robust software system.

CertiK has served more than 100 clients with high quality auditing and consulting services, ranging from stablecoins such as Binance's BGBP and Paxos Gold to decentralized oracles such as Band Protocol and Tellor. CertiK customizes its engineering tool kits, while applying cutting-edge research on smart contracts, for each client on its project to offer a high quality deliverable. For more information: https://certik.io.

Executive Summary

This report has been prepared for **Swipe** to discover issues and vulnerabilities in the source code of their **Swipe-Network** smart contracts. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structures and implementations against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Testing Summary

SECURITY LEVEL

Smart Contract Audit
This report has been prepared as a product of the Smart Contract Audit request by **Swipe**.
This audit was conducted to discover issues and vulnerabilities in the source code of **Swipe's Swipe-Network** Smart Contracts.

TYPE	**Smart Contract**
SOURCE CODE	https://github.com/SwipeWallet/Swipe-Network/tree/master/contracts
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	Aug 13, 2020
REVISION DATE	Aug 20, 2020
METHODS	A comprehensive examination has been performed using Dynamic Analysis, Static Analysis, and Manual Review.

Review Notes

Introduction

CertiK team was contracted by Swipe to audit the design and implementation of their Swipe-Network Smart Contracts for security vulnerabilities and compliance with Solidity language standards under different perspectives and with different tools such as static analysis and manual reviews by smart contract experts to find specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

The audited files and sha256 checksums are:

- **ERC20Token.sol**

  `9d259be2d5db40b8d39435244aa1377f3e6e96b21a99d301fe9feb2936eac3fe`
- **Event.sol**

  `bfdc9a12af187c7e48a24a810c8f63d3eb5681ed287a99a6a8da490b94f8f16b`
- **Ownable.sol**

  `5d54fca546081ed1dbe044ddf7bcb04f1661ff3dcb16e412741cdc59fad124c3`
- **Registry.sol**

  `4e909191dcb12964444655c2f8e093a0533a4ba0d81020694fc31ad99c77c20a`
- **SafeMath.sol**

  `469b57d4f3c4e1d39e117ea6839a987e2a6e5b2fde6cce7a72e609df8b7b1443`
- **Staking.sol**

  `6f6ac0c696aa886de78e72ec11ccadbb04ac37865dbdd8ee6507707eb0cab4b6`
- **Storage.sol**

  `c5d8a92eff6d0579914fbab0e10d2ae26c55b93b053ba4444c5c45b1b4277476`
- **Upgradeable.sol**

  `838b80a5a87b4ba08933920c2d86e279f479284b0edef9c63175ec868561dabe`

Summary

The team has reviewed the smart contracts of Swipe-Network for security vulnerabilities and compliance with Solidity and Yul language standards. The codebase was found to be very well-written with regards for performance and minimizing gas cost through the use of inline assembly (Yul).

The ERC20, Ownable and SafeMath smart contracts were all verified to adhere to the standard prototypes and specifications as defined in their respective Ethereum publications, or from previously-verified and secure smart contract implementations supplied by OpenZeppelin. The Registry contract was found to properly implement the Upgradeable contract via delegatecall-based proxy pattern.

There are some major concerns with the placement of external ERC20Token transferring in the implementation of the Staking contract that would allow for re-entrancy attacks due to lack of the Check Effects Interactions pattern, leading to an attacker having the ability to claim the reward pool for a particular token.

Findings

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Vulnerable to re-entrancy attacks from malicious users of ERC-20 tokens.	Control Flow	Major	Staking, L51-80

[MAJOR] Description:

This would allow for re-entrancy attacks due to lack of the Check Effects Interactions pattern, leading to an attacker having the ability to claim the reward pool of a particular ERC-20 token.

Recommendation:

Reference the Check Effects Interactions pattern:

https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html

Apply changes to state variables before transferring ERC-20 tokens. If the transfer of the ERC-20 tokens fails, the state variables will be reverted. This will undo the deletion of the approved claim amount from the mapping and restore the previous reward pool amount, safely preventing the possibility of re-entrancy attacks:

```
delete _approvedClaimMap[msg.sender][nonce];
_rewardPoolAmount = _rewardPoolAmount.sub(amount);

emit Claim(msg.sender, amount, nonce);

require(
    ERC20Token(_tokenAddress).transfer(msg.sender, amount),
    "Claim failed"
);
```

Alleviation: A change was imposed inside the “claim” function that prevents this issue.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Vulnerable to re-entrancy attacks from malicious users of ERC-20 tokens.	Control Flow	Minor	Staking, L22-44

[MINOR] Description:

While this issue will not lead to compromising the reward pool, it will lead to gas exhaustion and should generally be avoided.

Recommendation:

Reference the Check Effects Interactions pattern:

https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html

Apply changes to state variables before transfering from ERC-20 tokens. If the transfer from the ERC-20 token fails, the state variables will be reverted. This will undo the action of increasing the staked amount from the sender and restore the total staked amount:

```
_stakedMap[msg.sender] = _stakedMap[msg.sender].add(amount);
_totalStaked = _totalStaked.add(amount);

emit Stake(msg.sender, amount);

require(
    ERC20Token(_tokenAddress).transferFrom(
        msg.sender,
        address(this),
        amount
    ),
    "Stake failed"
);
```

Alleviation:

A change was imposed inside the “stake” function that prevents the aforementioned issue.

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Vulnerable to re-entrancy attacks from malicious users of ERC-20 tokens.	Control Flow	Minor	Staking, L87-108

[MINOR] Description:

While this issue will not lead to compromising the reward pool, it will lead to gas exhaustion and should generally be avoided.

Recommendation:

Reference the Check Effects Interactions pattern:

https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html

Apply changes to state variables before transferring ERC-20 tokens. If the transfer of the ERC-20 tokens fails, the state variables will be reverted. This will restore the previous total staked amount and restore the staked about for the message sender, safely preventing the possibility of re-entrancy attacks:

```
_totalStaked = _totalStaked.sub(amount);
_stakedMap[msg.sender] = _stakedMap[msg.sender].sub(amount);

emit Withdraw(msg.sender, amount);

require(
    ERC20Token(_tokenAddress).transfer(msg.sender, amount),
    "Withdraw failed"
);
```

Alleviation:

A change was imposed inside the “withdraw” function that prevents the aforementioned issue.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Use of block.timestamp	Exposure	Informational	Staking, L136

[INFORMATIONAL] Description:

block.timestamp can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Use of block.timestamp	Exposure	Informational	Staking, L240

[INFORMATIONAL] Description:

block.timestamp can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Vulnerable to re-entrancy attacks from malicious users of ERC-20 tokens.	Control Flow	Minor	Staking, L256-277

[MINOR] Description:

While this issue will not lead to compromising the reward pool, it will lead to gas exhaustion and should generally be avoided.

Recommendation:

Reference the Check Effects Interactions pattern:

https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html

Apply changes to state variables before transferring ERC-20 tokens. If the transfer of the ERC-20 tokens fails, the state variables will be reverted. This will restore the previous reward pool amount, safely preventing the possibility of re-entrancy attacks:

```
_rewardPoolAmount = _rewardPoolAmount.add(amount);

emit DepositRewardPool(msg.sender, amount);

require(
    ERC20Token(_tokenAddress).transferFrom(
        msg.sender,
        address(this),
        amount
    ),
    "Deposit reward pool failed"
);
```

Alleviation:

A change was imposed inside the “depositRewardPool” function that prevents the aforementioned issue.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Vulnerable to re-entrancy attacks from malicious users of ERC-20 tokens.	Control Flow	Minor	Staking, L284-309

[MINOR] Description:

While this issue will not lead to compromising the reward pool, it will lead to gas exhaustion and should generally be avoided.

Recommendation:

Reference the Check Effects Interactions pattern:

https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html

Apply changes to state variables before transferring ERC-20 tokens. If the transfer of the ERC-20 tokens fails, the state variables will be reverted. This will restore the previous reward pool amount, safely preventing the possibility of re-entrancy attacks:

```
_rewardPoolAmount = _rewardPoolAmount.sub(amount);

emit WithdrawRewardPool(msg.sender, amount);

require(
    ERC20Token(_tokenAddress).transfer(msg.sender, amount),
    "Withdraw failed"
);
```

Alleviation:

A change was imposed inside the “withdrawRewardPool” function that prevents the aforementioned issue.