# SMART CONTRACT SECURITY AUDIT

This report presents the results of our engagement with DefiXFinance to review all DefiXFinance and DefiXFinance Family smart contracts.

## Document Revision History

| Version | Modification | Date | Author |
|---|---|---|---|
| 0.1 | Document Creation | 12/04/2023 | Johanne Blake |
| 0.2 | Document Edits | 18/04/2023 | Johanne Blake |
| 1.0 | Final Edits | 20/04/2023 | David Bhanke |
| 1.1 | Remediation Plan | 23/04/2023 | David Bhanke |

## Contact


| Contact | Company | Email |
|---|---|---|
| Stephen Urgue | Xsec Finance | stephen.urgue@xsec.finance |
| Saufi Yuriv | Xsec Finance | saufi.y@xsec.finance |
| Khalid Wvyne | Xsec Finance | khalid.w@xsec.finance |
| Byori Vlad | Xsec Finance | byori.v@xsec.finance |


## 1 | EXECUTIVE OVERVIEW

### 1.1 | Introduction

DefiXFinance's staking implementation platform is called DefiStaking. Xsec Finance was hired by DefiXFinance to perform a security audit of their DefiStaking smart contracts. The audit began on February 14th 2023 and ended on April 23rd 2023. The security audit was performed on the Smart Contract EasyFi Staking Smart Contract.

Although the security audit was successful, it only covered the essential elements. This is due to resource and time constraints. It is important to highlight the best practices in secure smart-contract development.

### 1.2 | Audit Summary

DefiXFinance provided Xsec Finance with a week to complete the engagement. They also assigned a full-time security engineer to inspect the security of the smart contracts. The security engineers are experts in smart-contract security and advanced penetration testing. They also have deep knowledge about multiple blockchain protocols.

This audit is designed to accomplish the following:

- Make sure smart contract functions are used.
- Identify security concerns with smart contracts.

Xsec Finance concluded that there were no security threats. He recommends further testing to verify extended safety and correctness within the context of all contracts. External threats such as economic attacks and oracle attacks and inter-contract functions, calls, should be tested for expected logic, state, and safety.

# SMART CONTRACT SECURITY AUDIT

This report presents the results of our engagement with DefiXFinance to review all DefiXFinance and DefiXFinance Family smart contracts.

## 1.3 | Test Approach & Methodology

To balance efficiency, timeliness, practicality and accuracy, Halborn used a combination manual and automated security testing. These phases and the associated tools were used throughout this audit.

- Research into architecture and its purpose.
- Walkthrough and manual code reading for Smart Contracts
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual Assessment of safety and use for critical Solidity variables and functions. This is to determine any arithmetic-related vulnerability classes.
- Static Analysis for security for scoped contracts and import functions. (Slither)
- Testnet deployment (Truffle, Ganache)

### Risk Methodology

Xsec Finance ranks vulnerabilities and issues based on the risk assessment method. This methodology measures the LIKELIHOOD for a security incident and the IMPACT should it occur. This framework is used to communicate the nature and impact of technology vulnerabilities. The quantitative model allows for repeatable, accurate measurement and allows users to see the vulnerability characteristics that were used to generate the Risk Scores. A risk level for each vulnerability will be calculated using a scale from 5 to 1, with 5 representing the greatest likelihood or impact.

#### Likelihood - Risk Scale

5 - It is almost certain that an incident will happen.

4 - Very high chance of an incident.

3 - The long-term potential for a security incident

2 - Very low chance of an incident.

1 - A very unlikely problem will cause an incident.

#### Risk Scale - Impact

5 - May cause irreparable and devastating loss.

4 - Can cause significant loss or impact.

3 - This may cause a partial or complete loss for many.

2 - May cause temporary loss or impact.

1 - May have a minimal or non-noticeable effect.

These two values are combined to calculate the risk level, with 10 being highest.

| CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL |
|---|---|---|---|---|

10 - **CRITICAL**

9 - 8 - **HIGH**

7 - 6 - **MEDIUM**

5 - 4 - **LOW**

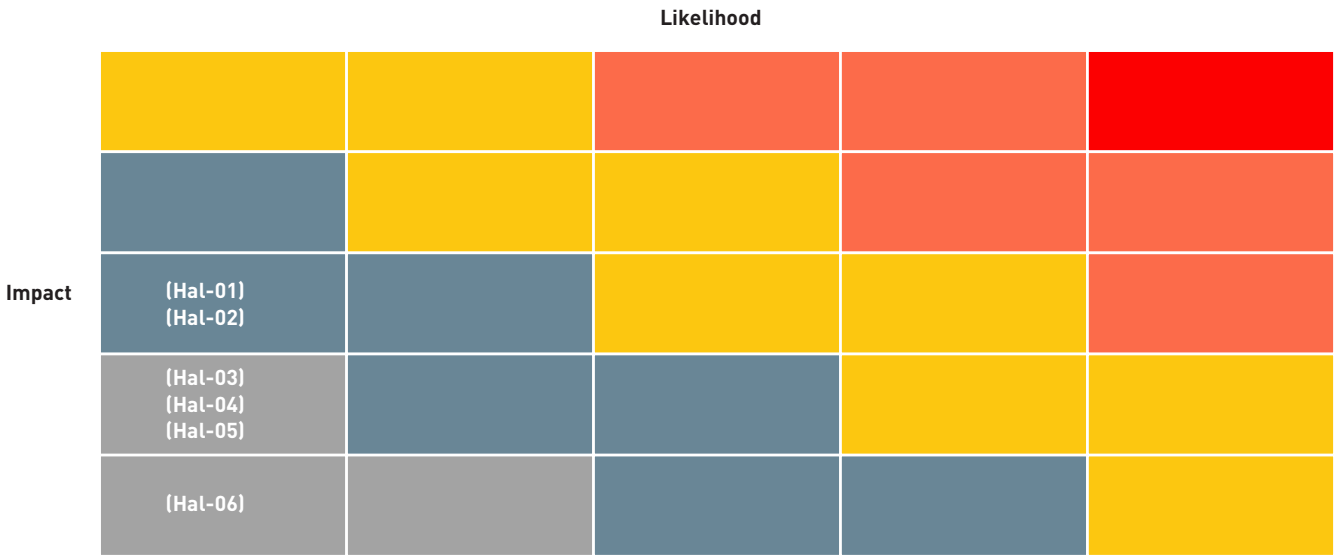
3 - 1 - **VERY LOW AND INFORMATIONAL**

# SMART CONTRACT SECURITY AUDIT

This report presents the results of our engagement with DefiXFinance to review all DefiXFinance and DefiXFinance Family smart contracts.

## 2 | ASSESSMENT SUMMARY & FINDINGS OVERVIEW

| CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL |
|---|---|---|---|---|
| 0 | 0 | 0 | 2 | 4 |




| Security Analysis | Risk Level | Date |
|---|---|---|
| Pragma Version Deprecated | Low Solved | Solved - 16/04/2023 |
| Floating Pragma | Low Solved | Solved - 16/104/2023 |
| Missing Bound Check | Informational | Solved - 18/04/2023 |
| Integer Overflow | Informational | Solved - 18/04/2023 |
| No Test Coverage | Informational | Solved - 20/04/2023 |
| Documentation | Informational | Solved - 20/04/2023 |

## 3 | FINDINGS & TECH DETAILS

### 3.1 (Hal-01) Pragma Version Deprecated LOW

Description

The current version of the contract is pragma ^0.5.16. This version is functional and can be used to mitigate security issues such as SafeMath.sol or ReentrancyGuard.sol. However, it increases the risk to the integrity and long-term sustainability of the solidity code.

Code Location

Listing 1: DefixStaking.sol (Lines 1)

```
1 pragma solidity ^0.5.16;
2
3 /**
```

Risk Level

Likelihood - 1

Impact - 3

Recommendations

The current version of pragma is 0.8.6 at the time this audit was conducted. Use the latest and most tested pragma version whenever possible to benefit from new features such as checks and accounting as well as preventing insecure code use. (0.6.12)

Plan For Remediation

Pragma version upgraded to 0.7.6. Solved

### 3.2 | (Hal-02) Floating Pragma LOW

Description

Smart contract DefiStaking.sol uses floating pragma ^0.5.16. The lock helps ensure contracts are not accidentally deployed with another pragma. An outdated pragma might have bugs that negatively affect the contract system, or newer pragma versions could have security vulnerabilities.

Code Location

Listing 2: DefixStaking.sol (Lines 1)

```
1 pragma solidity ^0.5.16;
2
3 /**
```

Risk Level

Likelihood - 1

Impact - 3

### Recommendations

You might want to lock the pragma version. A floating pragma is not recommended for production. The sign (), which is used to lock the pragma version, must be removed. You can lock the pragma in two places: `truffle-config.js` in Truffle framework and `hardhat.config.js` in HardHat framework.

### Plan For Remediation

Pragma version locked at `0.7.6` **Solved**

## 3.3 | (Hal-03) Missing Bound Check INFORMATIONAL

### Description

`NotifyRewardAmount()` calculates `rewardRate` by dividing reward by `rewardDuration`. To calculate `rewardRate`, `reward.add (leftover)` must be divided by `rewardsDuration`. `RewardRate` is `0.01` if both the denominator and numerators are greater than each other.

### Code Location

Listing 3: DefixStaking.sol (Lines 1)

```
619     function notifyRewardAmount(uint256 reward) external
            onlyRewardsDistribution updateReward(address(0)) {
620
621         if (block.timestamp >= periodFinish) {
622
623             rewardRate = reward.div(rewardsDuration);
624
625         } else {
626
627             uint256 remaining = periodFinish.sub(block.timestamp);
628
629             uint256 leftover = remaining.mul(rewardRate);
630
631             rewardRate = reward.add(leftover).div(rewardsDuration)
                    ;
632
633         }
634
635
636
637         // Ensure the provided reward amount is not more than the
                balance in the contract.
638
639         // This keeps the reward rate in the right range,
                preventing overflows due to
640
641         // very high values of rewardRate in the earned and
                rewardsPerToken functions;
642
643         // Reward * leftover must be less than 2^256 / 10^18 to
                avoid overflow.
644
645         uint balance = rewardsToken.balanceOf(address(this));
646
647         require(rewardRate <= balance.div(rewardsDuration), "
```

```
            Provided reward too high");
648
649
650
651         lastUpdateTime = block.timestamp;
652
653         periodFinish = block.timestamp.add(rewardsDuration);
654
655         emit RewardAdded(reward);
```

Risk Level

Likelihood - 1

Impact - 2

Recommendation

Listing 4

```
1 require(reward >= rewardsDuration, "Reward is too small");
```

Listing 5

```
1 require(reward.add(leftover) >= rewardsDuration, "Reward is too
      small");
```

Remediation Plan

RISK ACCEPTED. RewardsDuration is based upon UNIX timestamp. The reward is in wei (token with decimals 6,8,18 included). Reward tokens will never be lower than 1000 1000,000,000 (for 1 year) will be the minimum case (Reward token having 6 decimals and a duration of one year)

### 3.4 | (Hal-04) Integer Overflow INFORMATIONAL

Description

`NotifyRewardAmount()` calculates `rewardRate` by dividing reward by `rewardDuration`. To calculate `rewardRate`, `reward.add (leftover)` must be divided by `rewardsDuration`. `RewardRate` is `0.01` if both the denominator and numerators are greater than each other.

# SMART CONTRACT SECURITY AUDIT

This report presents the results of our engagement with DefiXFinance to review all DefiXFinance and DefiXFinance Family smart contracts.

Code Location

Listing 6: DefixStaking.sol (Lines 1)

```
619     function notifyRewardAmount(uint256 reward) external
            onlyRewardsDistribution updateReward(address(0)) {
620
621         if (block.timestamp >= periodFinish) {
622
623             rewardRate = reward.div(rewardsDuration);
624
625         } else {
626
627             uint256 remaining = periodFinish.sub(block.timestamp);
628
629             uint256 leftover = remaining.mul(rewardRate);
630
631             rewardRate = reward.add(leftover).div(rewardsDuration)
                    ;
632
633         }
634
635
636
637         // Ensure the provided reward amount is not more than the
                balance in the contract.
638
639         // This keeps the reward rate in the right range,
                preventing overflows due to
640
641         // very high values of rewardRate in the earned and
                rewardsPerToken functions;
642
643         // Reward * leftover must be less than 2^256 / 10^18 to
                avoid overflow.
644
645         uint balance = rewardsToken.balanceOf(address(this));
646
647         require(rewardRate <= balance.div(rewardsDuration), "
                Provided reward too high");
648
649
650
651         lastUpdateTime = block.timestamp;
652
653         periodFinish = block.timestamp.add(rewardsDuration);
654
```

Risk Level

Likelihood - 1

Impact - 2

Recommendations

To cover all possible scenarios, we recommend that you run as many test cases and as many smart contracts as possible.

Listing 7

```
require(reward < uint(-1).div(1e18), "Reward overflow");
```

Plan For Remediation

RISK ACCEPTED. The contract will determine the reward amount at deployment.

### 3.5 | (Hal-05) No Test Coverage INFORMATIONAL

Description

Smart contracts are not like traditional software and cannot be modified unless they are deployed with a proxy contract. To ensure that the code is correct before deployment, it is recommended to run unit and functional tests. Mocha and Chai can be used to run unit tests in smart contract contracts. Mocha is a Javascript framework to create synchronous and/or asynchronous unit test cases. Chai is an assertion library that can be used for custom unit testing.

References:

https://github.com/mochajs/mocha

https://github.com/chaijs/chai

https://docs.openzeppelin.com/learn/writing-automated-tests

Risk Level

Likelihood - 1

Impact - 2

Recommendation

To ensure that the smart contract covers all possible scenarios, we recommend running as many test cases and as many as possible.

Plan For Remediation

Xsec Finance Team increased test coverage. Solved

### 3.6 | (Hal-05) No Test Coverage INFORMATIONAL

Description

The documentation provided by EasyFi is incomplete. The documentation in the GitHub repository, for example, should contain a walkthrough of how to deploy and test smart contracts.

Recommendations

For greater ease, consider updating documentation in Github

Contracts are tested and deployed. To ensure that the set-up steps are correct due to technical assumptions, a non-developer or QA resource should be involved.

Plan For Remediation

The DefiXFinance Team documented every stage of deployment. Solved

## 4 | AUTOMATED TESTING

### 4.1 | Static Analysis Report

Description

To increase coverage in certain areas of the scoped contract, Halborn used automated testing techniques. Slither, a Solidity static analytics framework, was one of the tools used. Halborn checked all contracts in the repository. He was then able to compile them into the correct abi and binary formats. This tool allows you to statically verify the mathematical relationships between Solidity variables in order to detect inconsistent or invalid usage of contracts' APIs throughout the entire code-base.

```
INFO:Detectors:
StakingRewards.notifyRewardAmount(uint256) (contracts/staking/stakingFactory.sol#619-638) performs a multiplication on the result of a division:
        -rewardRate = reward.div(rewardsDuration) (contracts/staking/stakingFactory.sol#621)
        -leftover = remaining.mul(rewardRate) (contracts/staking/stakingFactory.sol#624)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
```

### 4.2 | Automated Security Scan

Description

To assist in detecting well-known security problems and identify low-hanging fruits on targets for this engagement, Xsec Finance used automated security scans. MythX was used as a security analysis tool for Ethereum smart contracts. MythX ran a scan of the testers' machines and sent the results to the analyzers for any vulnerabilities. Below are only security-related findings.

```
Report for contracts/staking/stakingFactory.sol
https://dashboard.mythx.io/#/console/analyses/01c2b4e8-641f-4f30-b5b5-b0fe400ba978
```

| Line | SWC Title | Severity | Short Description |
|---|---|---|---|
| 1 | (SWC-103) Floating Pragma | Low | A floating pragma is set. |
| 103 | (SWC-000) Unknown | Medium | Function could be marked as external. |
| 129 | (SWC-000) Unknown | Medium | Function could be marked as external. |
| 130 | (SWC-000) Unknown | Medium | Function could be marked as external. |
| 308 | (SWC-000) Unknown | Medium | Function could be marked as external. |
| 316 | (SWC-000) Unknown | Medium | Function could be marked as external. |
| 332 | (SWC-000) Unknown | Medium | Function could be marked as external. |
| 695 | (SWC-000) Unknown | Medium | Function could be marked as external. |
| 707 | (SWC-000) Unknown | Medium | Function could be marked as external. |
| 709 | (SWC-128) DoS With Block Gas Limit | Medium | Loop over unbounded data structure. |

# 5 | OUR METHODOLOGY

We prefer to work in a transparent manner and make our reviews a team effort. Our security audits aim to improve the quality and provide sufficient remediation to protect users. This is how we conduct our security audits.

## 1.1 | Manual Code Review

We manually review all code to identify any possible issues in code logic, error handling protocol and header parsing, cryptographic mistakes, random number generators, and code logic. We also look for areas where more defensive programming might reduce the risk of future errors and speed up future audits. While our main focus is on in-scope code, dependency code and behavior are also examined when they are relevant to a specific line of investigation.

## 1.2 | Vulnerability Analysis

Our audit methods included manual code analysis, user interface interaction and whitebox penetration testing. To get a good understanding of the functionality of the software under review, we look at the web site. Then, we meet with the developers to get a better understanding of their vision for the software. We then install the software and explore the roles and interactions of the users. We also brainstorm attack surface and threat models while we do this. We review design documentation, look at audit results, search to find similar projects, inspect source code dependencies and skim open issues tickets.

## 1.3 | Documenting the Results

We use a conservative and transparent approach to analyzing security vulnerabilities and ensuring that they are addressed. We immediately create an Issue entry in this document for any potential issue that is identified. However, we are not yet able to verify the impact or feasibility of the issue. We document our suspicions early, even if it later turns out to be not exploitable vulnerabilities. This is conservative. The process involves first documenting suspicions with unresolved queries, and then verifying the issue via code analysis, live experimentation or automated tests. We aim to provide log captures, screenshots and test code as proof of our findings, which is why code analysis is the most uncertain. The feasibility of an attack on a live system is then assessed.

## 1.4 | Suggested Solutions

We look for immediate mitigations that live deploys can take and then we recommend the requirements for remediation engineering to be used in future releases. Developers and deployment engineers should review the mitigation and remediation recommendations. Successful mitigation and remediation are ongoing collaborative processes after we have delivered our report.

# 6 | DISCLAIMERS

## 2.1 | Disclaimer for Xsec Finance Auditors

Audit of smart contracts has been done in accordance to industry best practices. This includes cybersecurity vulnerabilities and issues in smart-contract source code. Details of these are disclosed in the Source Code.

The audit does not make any statements or warranties about the security of the code because the number of test cases is unlimited. The audit cannot also be considered a complete assessment of the utility and safety, bugfree status, or any other statements about the contract. We have tried our best to produce this report and conduct the analysis. However, you shouldn't rely solely on it. We recommend that you do several independent audits as well as a public bug bounty program in order to ensure smart contract security.

## 2.1 | Technical Disclaimer

Blockchain platforms are used to deploy and execute smart contracts. Hackers can target the platform, its programming language and any software that is related to smart contracts. The audit cannot guarantee the security of audited smart contract.