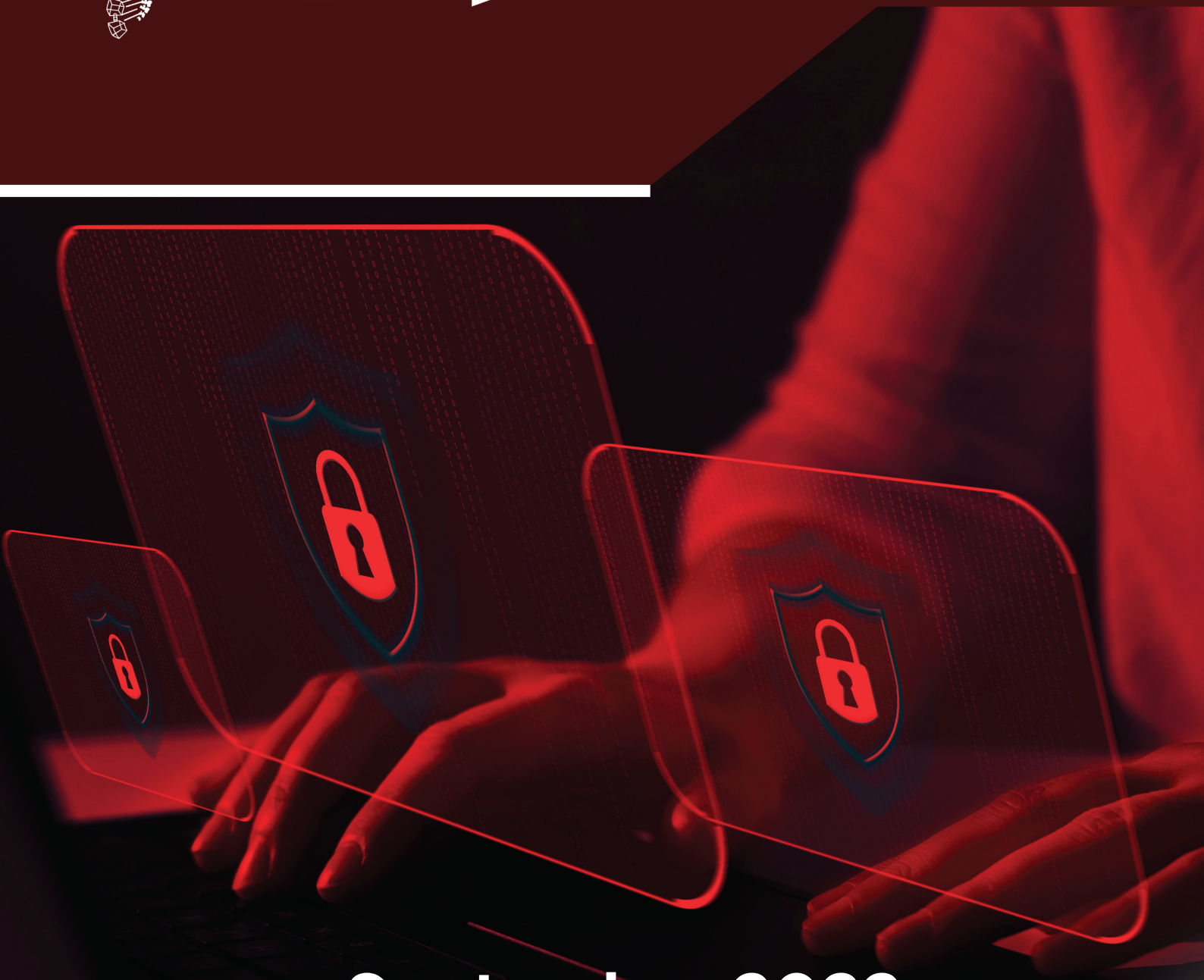
DFX

CONTRACT AUDIT BY

SEBONE



September 2022

Audited By: Elias Leers and Alexander Blair

Table of Contents

Summary

Overview of Contract

■ Commit Hash: ff0857cfd5a93883329dc4682839b6996c1f10f9

DFX Protocol is a decentralized foreign exchange protocol optimized for stablecoins. DFX v2 is an update from DFX Protocol v0.5 with some additional features including the protocol fee, which is set by the percentage of the platform fee (which incurs for each swap on all the pools across the platform), fixing issues of invariant check, and the support of flashloans. The major change from the previous version is, v2 is more generalized for users, meaning anybody can create their curves (pools) while v0.5 only allowed the DFX team to create pools. It has been a pleasure working with the DFX team and they resolved each of the issues promptly.

All issues have been resolved.

High	Medium	Low	Info
0	5	5	5

FINDINGS

Findings Overview

#	Contract	Issue	Risk Level
1.1	N/A	Oracle Front Run	Medium
2.1	Curve	Missing Proxy Protection	Medium
2.2	Curve	Deposit With Whitelist Amount Check	Low
2.3	Curve	Unclear Error	Info
2.4	Curve	Emergency State Clarification	Info
2.5	Curve	Transactable Check	Info
3.1	AssimilatorFactory	Protected Function	Medium
3.2	AssimilatorFactory	Verification	Low
3.3	AssimilatorFactory	newAssimilator Definition	Low
4.1	Storage	Overwriting Curve	Medium
5.1	Orchestrator	Potential Time Lag	Medium
5.2	Orchestrator	Unnecessary Function	Info
6.1	Router	Library Update	Low
7.1	Zap	Currency Check	Low
8.1	ProportionalLiquidity	Equivalent Function	Low

Findings Explanation

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or still need addressing. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior.

High

The issue affects the contract in such a way that funds may be lost, misallocated, or otherwise result in a significant loss.

Medium

The issue affects the ability of the contract to compile or operate in a significant way.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate and is meant only as additional.

Process

The Sebone team has followed best practices and industry-standard techniques to verify the implementation of DFX's contracts. To do so, the code is reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as they are discovered. Part of this work includes writing unit tests. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

- Due diligence in assessing the overall code quality of the codebase.
- Cross-comparison with other, similar smart contracts by industry leaders.
- Testing contract logic against common and uncommon attack vectors.
- Thorough, manual review of the codebase, line-by-line.
- Considering the deployment path of each component and how that will interact with a production environment.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of thecontract, but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Sebone recommend that the DFX team put in place a bug bounty program to encourage further and active analysis of the smart contract.

1.1 Oracle Front Run

Risk Level:

Medium

Description

Price updates to the oracle can be front-run to profit from volatility. This has a higher risk on listed tokens that have a higher change threshold between oracle updates such as NZDS. This requires a significant up-front investment of capital by the attacker.

Recommendation

Use oracles with a lower change threshold for updates. Monitor transactions to the contract for malicious activity.

Resolution

Team is going to setup their AMM infrastructure in advance per internal discussions.

2.1 Missing Proxy Protection

Risk Level:

Medium

◆ Contract

Curve

◆ Description

The functions that take in a curve from the memory space perform no checks against other tokens or transfers, and only verify the balances in the curve itself. The lack of checks allows a malicious actor to generate a proxy contract that could have adverse effects. This structure can be potentially misleading, & while not a critical issue, patching will be good for overall safety

◆ Example

Provide a proxy to inject a curve that contains incorrect balances.

◆ Recommendation

Implement `noDelegateCall` protection, as found in the `flash` loan system.

```
    function flash(
        address recipient,
        uint256 amount0,
        uint256 amount1,
        bytes calldata data
    ) external transactable noDelegateCall {
```

◆ Resolution

Team added `noDelegateCall` where needed.

2.2 DepositWithWhitelist Validation

Risk Level:

Low

◆ Contract

Curve

◆ Description

Per in-contract documentation, **Amount** should always be 1, but this is never enforced, nor is the index required to be within an acceptable range as defined.

DepositWithWhitelist relies on the Merkle tree to interact with the contract. Since the Merkle root is fixed, as defined in **MerkleProver,** the user is provided with 2 values that allow the potential long-term brute forcing of a valid Merkle path **(Index/Amount)** which are never used beyond determining thekeccak256 of the node value.

Due to the lack of a check on **Amount,** there are 2 unique **uint256's** that can be used to aim for a collision, as there are 947 previously provided hashes that could be targeted at various points in an attacker's tree. This could be partially prevented by removing the JSON file, but they could still be derived from calls to the **depositWithWhitelistFunction.**

Most importantly, this exploit can be done entirely offline and requires no online/chain-based resources to perform. Given the high speed of calculations for **keccak256** in optimized GPU environments, it could be a consideration in the future.

Likely a non-issue due to time constraints, but as there is no way to un-whitelist someone once they've solved a way to get into the Merkle tree, this may be a consideration if the platform continues to grow to a large enough size where this could be valuable.

◆ Recommendation

This could be be a a hard-coded list but in lieu of that at a a minimum, both **Amount** and **Index** value should be verified.

◆ Resolution

Checks have been added to cover both **Amount** and **Index**.

2.3 Unclear Error

Risk Level:

Info

◆ Contract

Curve

◆ Description

The naming of the functions and the associated errors would indicate that they are inverted:

→ **inWhitelistingStage** should return an error that whitelisting stage has passed.

→ **notInWhitelistingStage** should return an error that the whitelisting stage has not ended.

◆ Recommendation

Review the errors and ensure that they are operating as intended.

◆ Resolution

Modifier errors corrected.

2.4 Emergency State Clarification

Risk Level:

Info

◆ Contract

Curve

◆ Description

Code: `emergency-only-allowing-emergency-proportional-withdraw`
During the emergency state, normal ERC-20 functions, and deposit functions are not disabled. Flashes are also not being prevented.

Additionally, `emergencyProportionalWithdraw` and `proportionalWithdraw` are the same exact code, so there is a question of the necessity of this state and function (see Equivalent Function for more details).

◆ Recommendation

If this is as intended, documentation should be updated accordingly. Otherwise, the function should be updated to perform as required.

◆ Resolution

Checks have been added to ERC20 functions as well as the swaps and flash functions.

2.5 Transactable Check

Risk Level:

Info

◆ Contract

Curve

◆ Description

The **transactable** check still permits ERC-20 transfers, which is separate from the internal definition of **transactable,** which is deposit and withdraw, but this may be intentional.

◆ Resolution

This has been indicated as working as intended and no updates are required.

3.1 Protected Function

Risk Level:

Medium

◆ Contract

AssimilatorFactory

◆ Description

This contract could be front-run to implement an invalid oracle, through someone generating 2 seperate /bad/ curves in **curve** factory, one that contains a bad **baseCurrency** pair, and one that contains a bad **quoteCurrency** pair as the hash for a **curveID** is k256(base, quote), it is viable (If expensive) to generate 2 curves at higher fees that would inject valid assimilators with bad oracles that are then picked up by a real contract. This may be partially alleviated by manual review after to ensure that the correct assimilator has the expected oracle, as well, this could be done generally during **newCurve,** which would also fix the issue and point out bad oracles in the system so they could be cleaned up by an admin **(revokeAssimilator)**.

Alternatively, it may be preferable to not require anyone and/or everyone to be able to generate an **Assimilator** (Which is possible via **newCurve),** though this is dependent on the aim of the contract. Due to this, there's no real reason to protect this function as its caller is unprotected.

We found no simple way to validate oracle correctness. In order to verify if the oracle is good/bad, a manual review of the constructor or a contract execution flow would be required. This is due to the token and the oracle both being tagged private with no view function.

◆ Resolution

The oracles that are being deployed are now tracked, rendering the assimilator creation path much safer.

3.2 Verification

Risk Level:

Low

◆ Contract

AssimilatorFactory

◆ Description

newAssimilator and the oracles are not verified properly.

The oracle is never verified to be a valid oracle or for code availability.

Since this is used for pricing, someone could register a token with an invalid oracle, which could then be used in another pair.

newAssimilator never verifies that the token address is actually a contract that has code available.

◆ Recommendation

Add verification to check that the addresses are functioning as expected.

◆ Resolution

Oracles are now being verified to be oracles.

3.3 newAssimilator Definition

Risk Level:

Low

◆ Contract

AssimilatorFactory

◆ Description

newAssimilator error is also potentially incorrectly written, as an Assimilator is not actually generating a pair in the way that the rest of the system where it is a pair of coins, typically coin/UCDC. In the case of this function, it is an oracle/erc-20 pair.

◆ Recommendation

Update the language to reflect the type of pair that is being generated or use different verbiage.

◆ Resolution

The verbiage has been updated to be more clear.

4.1 Overwriting Curve

Risk Level:

Medium

◆ Contract

Storage

◆ Description

Curve is considered to be a soft “internal” variable so, while it’s not passed directly into the functions, it can be overwritten by **callcode/delegatecall.**

Due to this, of the contracts that have the potential for abuse on **delegatecall** and/or **callcode**, this contract is the biggest concern because the entire curve is in Storage which makes it very easy to copy the memory space.

Callcode is still valid in current EVM op, though solidity doesn't particularly like to compile it.

Note: While we were able to inject into the memory space and overwrite what is stored as the curve, we were unable to find an attack vector past that point to complete the exploit.

◆ Recommendation

- → Add **delegatecall/callcode** protections
- → Make a plan to track EVM updates that could open up exploits
- → Add to review checklist for any future contracts to be developed

◆ Resolution

noDelegateCall has been added to functions that use curve as an internal variable.

5.1 Potential Time Lag

Risk Level:

Info

◆ Contract

Orchestrator

◆ Description

There is a potential time lag between the pool init **(Construction)** and the **setParams** call

As **newCurve** does not initially receive the data required to set the parameters, they must either be generated by an external contract for a single block call or done with pre-calculation of the next contract id.

Any improper usage of this could cause a spike in block fees that may leave the contract in an unexpected state.

◆ Recommendation

While this technically should be safe due to the whitelisting stage, it may be worth considering converting **newCurve** to taking arguments that set a baseline curve param set for safety.

◆ Resolution

Default parameters have been added while allowing them to be altered later while maintaining whitelist protections.

5.2 Unnecessary Function

Risk Level:

Info

◆ Contract

Orchestrator

◆ Description

Private function **includeAssimilator** is not used and can likely be stripped.

◆ Recommendation

Remove **includeAssimilator**.

◆ Resolution

Function has been removed.

6.1 Library Update

Risk Level:

Info

◆ Contract

Router

◆ Description

Implements IERC20 for **safe<transfer/etc>**, which is used as well for flashes. However, it is only included, not used, in the assimilator code.

◆ Recommendation

This library should be updated for similar purposes, as the safe library protects against bad return values, which could be needed for a poorly written external contract.

◆ Resolution

Functions have been moved to Safe versions from the IERC20 wrapper.

7.1 Currency Check

Risk Level:

Low

◆ Contract

Zap

◆ Description

There is no check on the type of currency being deposited/used against the numeraire for the current curve.

◆ Recommendation

Need to check error handling around if the currency being deposited and/or the currency being used isn't in the numeraire for the curve.

◆ Resolution

A check has been added to ensure that the base is in the numeraires.

8.1 Equivalent Function

Risk Level:

Low

◆ Contract

ProportionalLiquidity

◆ Description

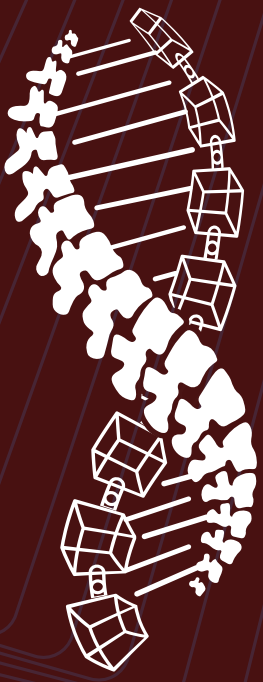
`_oGliq` was removed from the **proportionalWithdraw** function, making it identical to **emergencyProportionalWithdraw.**

◆ Recommendation

Remove or rename the emergency function to avoid redundancy and unnecessary code.

◆ Resolution

Function removed and references updated.



SEBONE

www.sebone.dev

September 2022