Smart Contract Security Audit Report for bStable

AnChain.AI Inc.

Auditors: AnChain.AI Audit Team

Contact: Audit@AnChain.ai

Date: 2020-11-4

Version: 3

Index

1. Introduction

1.1 Audit Info

This document includes the results of the security audit for the client's smart contract code bStable as found in the section 3. The security audit was conducted by the AnChain.AI team from Oct 12, 2020, to Oct 19, 2020.

The purpose of this audit is to review source code, functionality on test net, and provide feedback on the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

1.2 Methodology

AnChain.AI team adopts a suite of tools and best practice from cybersecurity to audit smart contract source code for vulnerabilities.

3 key aspects in the AnChain.AI security audit methodology:

- Vulnerability audit. We use various tools to scan for known vulnerabilities, including our proprietary CAS sandbox.
- Statistical audit. Our sandbox will predict the risk score for the audited code with hundreds of thousands of mainnet smart contracts to understand security stance.
- Business logic audit. Our experts will design specific test cases to cover the key custom functions, to identify potential risk exposures.

1.3 Environment

Compiled Version: Solidity Compiler v0.6.4+commit.1dca32f3.js

Compiled System: macOS

Compiled Tool: Remix IDE, Truffle

2. Executive Summary

2.1 Key Findings

Submission #1: We identified minor issues that can potentially compromise the integrity of the project under certain circumstances. This smart contract does not meet the AnChain.AI smart contract auditing standards. It is recommended for revision.

Submission #2: After revision, we **did not identify issues** that can compromise the integrity of the project. This smart contract **meets** the AnChain.AI smart contract auditing standards.

3. Smart Contract Scope

3.1 Audited Source Code File

File Name	Md5
StableSwapPool.sol	`17bc4e365c680028626f5981b55bc06f`

4. Vulnerability Audit

4.1 Vulnerability Checklist

Type	Status
Numerical Overflow / Underflow	PASS
Authorization Check	PASS
Transaction Ordering Dependency	PASS
Parity Multisig Bug	PASS
Random Number Practice	PASS
Re-entrancy	PASS
Function Visibility	PASS
Event Emitting	PASS
ERC20 Token Formatting	PASS

5. Business Logic Audit

This section is dedicated to the functionality test of the audited smart contract. The purpose is to cover the main business logic for completeness and correctness.

The smart contract has been deployed on Testnet, with the specified environment and parameters.

Compile Test

tool	Remix IDE
version	solc v0.6.4+commit.1dca32f3.js
result	PASS

Code modification for compiling

file name	n/a
before	n/a
after	n/a

There is no code modification for this audit.

Deployment Test:

Network	Local Testnet
Result	PASS

5.1 Ownable

Function	Status
transferOwnership	PASS
renounceOwnership	PASS

5.2 BEP20

Function	Status
approve	PASS
increaseAllowance	PASS
decreaseAllowance	PASS
transfer	PASS
transferFrom	PASS

5.3 StableSwapPool

Function	Status
kill_me	PASS
unkill_me	PASS
ramp_A	PASS
stop_ramp_A	PASS
revert_transfer_ownership	PASS
commit_new_fee	PASS
apply_new_fee	PASS
revert_new_parameters	PASS
add_liquidity	PASS
remove_liquidity	PASS
remove_liquidity_imbalance	PASS
remove_liquidity_one_coin	PASS
exchange	PASS
withdraw_admin_fees	PASS
donate_admin_fees	PASS

6. Gas Consumption Analysis

We tested gas consumptions of main functions, and all results are within normal gas consumption range.

Function Name	Gas Consumption
add_liquidity	~300,000
remove_liquidity	~120,000
remove_liquidity_imbalance	~260,000
remove_liquidity_one_coin	~110,000
exchange	~150,000

7. Recommendations

7.1 Integer Underflow

SmartSwapPool.sol, line 1697, might have integer underflow issue:

```
  dy = dy.sub(uint256(1)).div(PRECISION_MUL[i]);
  r1 = dy;
  r2 = dy_0 - dy;
}
```

Generally, it is safe to use native math operations (+, -, *, /) on constant variables. However, it is not the case for the above code where variables are mutable which could potentially results in integer underflow.

Update: This issue is resolved on submission #2.

7.2 Function Visibility

Line 93:

```
function transferOwnership(address newOwner) internal virtual  {
  require(newOwner != address(0), "Ownable: new owner is the zero address");
  emit OwnershipTransferred(_owner, newOwner);
  ...
```

Line 1812:

```
function commit_transfer_ownership(address _owner) external onlyOwner {
  require(transfer_ownership_deadline == 0, "  # dev: active transfer");

  ...
```

Line 1822:

```
function apply_transfer_ownership() external onlyOwner {
  require(
    block.timestamp >= transfer_ownership_deadline,
    ...
```

The design of ***commit_transfer_ownership()*** and ***apply_transfer_ownership()*** is to implement a grace period which allows reverting the operation. However, there exists ***transferOwnership()*** which can also execute the operation and take effect immediately. This defeats the purpose of commit and apply design. It is recommended to change the visibility of ***transferOwnership()*** to internal.

Update: This issue is resolved on submission #2. However, *revert_transfer_ownership()* can also be deleted.

7.3 Coding Style

Some functions are named using underscore while others are using camel case, which is not unified.

Line 1708:

```
function remove_liquidity_one_coin(
```

Line 128:

```
function balanceOf(address account) external view returns (uint256);
```

It does not affect the security of the smart contract. However, it is a good practice to keep code in one style.

8. Conclusion

This project has the basic logic of Curve.fi smart contract, with additional editing on liquidity token initialization and SafeMath module. We have verified this revision does not affect normal usage of the smart contract.

Submission #1: However, there exists potential flaws:

I. Math operations are not fully covered by SafeMath

II. Function design contradicts each other

Hence we recommend for revision to address the above issues.

Submission #2: After revision, we have confirmed that this project is secured.

9. Appendix

9.1 Vulnerability Technical Explanation

These vulnerability scannings are powered by the patented AnChain.AI CAS (Smart Contract Auditing Sandbox), including static analysis, dynamic analysis, and statistical analysis.

- **Numerical Overflow/Underflow**

Severity Level: high

In ETH, failing to run a numerical check on arithmetic operations may cause the values to overflow or underflow, which may cause crypto asset loss. It is a good practice to use "SafeMath" library for all the numeric operations to take care of overflow and underflow issue of all the numeric operations.

The code provided pass the 'numerical overflow' vulnerability check.

- **Authorization**

Severity Level: high

Parameters passed into a function should be consistent with the actual caller.

The best practice is to use 'require()' to check if the user executing the action matches the provided parameter.

The code provided pass the 'Authorization' vulnerability check.

- **Transaction Ordering Dependency:**

Severity Level: high

In ETH, timestamp of a transaction getting approved can be manipulated by vicious minors. Transaction Ordering Dependency refer to critical logic fault if minors approve the later submitted transaction PRIOR than the earlier ones.

Transaction ordering dependency does not have critical impact on this contract.

- **Parity Multisig Bug/ Delegate Call:**

Severity Level: High

The code does not call any external contracts and thus there is no issue about parity multisig bug.

- **Random Number**

Severity Level: High

Random number generator algorithm should not use predictable seeds.

The code does not generate or use any random number, and thus there is no issue about the random number issue.

The code provided pass the "Random Number" vulnerability check.

- **Re-entrancy Attack:**

Severity Level: High

The code does not contain any function that is vulnerable to re-entrancy attack.

- **Function Visibility:**

Severity Level: Low

The code does not contain any function that is improperly set with visibility.

- **Event Emitting:**

Severity Level: Low

The ERC code has emit transfer & approval event properly, which meets a ERC token standard.

- **ERC Token Standard:**

Severity Level: Low

The token contract has the following variables: name, symbol, decimal, totalSupply, which meets the ERC token standard.

9.2 Severity Level

Level	Description
High	The issue poses an existential risk to the project, and the issue identified could lead to massive financial or reputational repercussions.
Medium	The potential risk is large, but there is some ambiguity surrounding whether or not the issue would practically manifest.
Low	The risk is small, unlikely, or not relevant to the project in a meaningful way.
Code Quality	The issue identified does not pose any obvious risk, but fixing it would improve overall code quality, conform to recommended best practices, and perhaps lead to fewer development issues in the future.

9.3 Reference

[1]. Curve.fi StableSwap3Pool Source Code. Retrieved from https://github.com/curvefi/curve-contract/blob/master/contracts/pools/3pool/StableSwap3Pool.vy

[2]. Curve.fi Official Documentation. Retrieved from https://resources.curve.fi/

[3]. Curve.fi Pool README. Retrieved from https://github.com/curvefi/curve-vue/blob/master/src/docs/README.md

10. Disclaimer

Disclaimer - Smart Contract Auditing - ETH

AnChain.ai makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and AnChain.ai specifically disclaims all implied warranties or merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

AnChain.ai will not be liable for any lost profits, business, contracts, revenues, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand against the company by any other party. If no event will AnChain.ai be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if AnChain.ai has been advised of the possibility of such damages.

The scope of this report and review is limited to a review of only the code presented by the client's team and only the source code AnChain.ai notes as being within the scope of AnChain.ai's review within this report. This report does not include an audit of the deployment scripts used to deploy the contracts in the repository corresponding to this audit. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than AnChain.ai. Such hyperlinks are provided for your reference and convenience only and are the exclusive responsibility of such websites' owners. You agree that AnChain.ai is not responsible for the content or operation of such websites and that AnChain.ai shall have no liability to your or any other person or entity for the use of third party websites. AnChain.ai assumes no responsibility for the use of third-party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.