CRYPTIC LABS

www.crypticlabs.org

Smart Contract Code Review

Audit Report

for

Swerve Finance

Prepared by Cryptic Lab Code Audit Team
contact@crypticlabs.org

September 14, 2020

Version Information:

Version	**Date**	**Description**	**Point of Contact**
0.1	September 14, 2020		

For more information about this document, please use the following contact information:

Name	Bril Wang
Phone	650-391-6281
Email	bril@crypticlabs.org

Table of Contents

1. **Introduction**

1. Purpose of This Document

This report is prepared by Cryptic Lab Code Review team after carrying out a code review on the smart contract source codes by Swerve Finance projects. The information about the Swerve Finance is described as follows.

Swerve Finance Homepage	https://swerve-finance.com
Code location	https://github.com/SwerveFinance/SwerveContracts 487b410
Type of Codes	Ethereum Smart Contracts
Twitter	https://twitter.com/SwerveFinance
Telegram	https://t.me/swervefi

This report reviews Swerve Contract codes and this report does not include reviews for the swerve-web and swerve-ui codes.

2. About Swerve

Swerve is a fork of Curve Finance with the following claim: "*there is no fake-out deployment, no questionable pre-mining, no founder controlling majority of the governance vote, no suspect team proposals, no 30% allocation to 'shareholders', no team allocation, no decades long distribution, none of it. It is a simple 33,000,000 supply owned entirely by you, the community of liquidity providers and users. If you provide liquidity to Swerve, you get ySWRV tokens which can be staked in the Swerve DAO to earn $SWRV. To kickstart the protocol and encourage users to try out Swerve, the first two weeks will have a larger distribution of $SWRV awarded.*" The entire code of Swerve has not been fully reviewed and audited as the developer mentioned that "*There is very minimal new code introduced (and is quite simple), I have had some fellow farmers take a look and have been reviewing for some time but ultimately will recommend participants to proceed with caution (as they always should) until it is more publicly vetted.*" As a summary, the Swerve project removed some un-used contracts from the Curve project and "*have written independent Solidity code that interacts with Curve's contracts essentially as an on-chain API via delegation.*"

There will be a total of 33,000,000 SWRV tokens with the following release schedule and They will be issued in proportion to providers' supplied liquidity in comparison to the total liquidity provided in Swerve:

First 2 weeks: 9,000,000 SWRV
Year 1 (after the 2 weeks): 9,000,000 SWRV
Year 2: 3,000,000 SWRV
Year 3: 3,000,000 SWRV
Year 4: 3,000,000 SWRV
Year 5: 3,000,000 SWRV
Year 6: 3,000,000 SWRV

Swerve will initially deploy with a Y Pool [DAI, USDC, USDT, TUSD] and leave the addition of future pools up to the decision of the DAO.

3. References

1. https://thedefiant.substack.com/p/a-chat-with-john-deere-the-anonymous
2. https://www.curve.fi/curve_audits/curve-dao-quantstamp.pdf
3. https://www.curve.fi/curve_audits/00-ToB.pdf
4. https://www.curve.fi/curve_audits/01-ToB.pdf
5. https://resources.curve.fi/

4. Programming languages

The smart contracts are written in Vyper and Solidity.

5. Defect Checklist

No defect has been identified.

6. About Cryptic Labs

Cryptic Labs is an innovative commercial lab, accelerator and advisory focused on solving fundamental problems in Blockchain, security, privacy and trust. Cryptic Labs are comprised of expert security advisors, cryptographers, researchers, engineers, scientists and outstanding blockchain practitioners. Cryptic Labs work with companies globally to solve security, privacy, and decentralized and distributed trust challenges. Cryptic Labs also functions as a commercial accelerator focusing on security, economics, privacy and trust to advance the viability of the Blockchain. To established companies, Cryptic Labs offer advice and collaboration on research and applied technology solutions. To startup companies in its accelerator, Cryptic Labs act as a bridge between venture capital, academia and industry in the research and development of practical industry-altering solutions, ideas and resources.

7. Disclaimer

The Cryptic Lab's code review team has tried its best to identify security vulnerabilities in the code. However, the team does not give any warranty on finding all security issues of the given smart contracts. This security audit report should not be used as an investment advice.

2. Executive Summary

The team reviewed/inspected the Swerve Contracts codes located at https://github.com/SwerveFinance/SwerveContracts (487b410). The Swerve Contracts directory contains ten (10) source code files:

1. APYoracle.sol
2. ERC20CRV.vy
3. ERC20LP.vy
4. GaugeController.vy
5. LiquidityGauge.vy
6. Minter.vy
7. PoolProxy.vy
8. VotingEscrow.vy
9. YPoolDelegator.sol
10. ZapDelegator.sol

Among these ten (10) files, seven (7) files are written using Vyper (with *.vy extensions) and these source codes are revised from the "Curve Finance" project located in https://github.com/curvefi/. Since Curve smart contracts have been audited by Trail of Bits (https://www.curve.fi/audits) and the Curve DAO smart contracts have been audited by Trail of Bits, MixedBytes and Quantstamp (see, e.g., https://www.curve.fi/curve_audits/curve-dao-quantstamp.pdf), the focus of this review report will be on the three new Solidity source codes that have not been audited by other teams: APYoracle.sol, YPoolDelegator.sol, and ZapDelegator.sol. These solidity codes are around 153 lines of Solidity codes.

In the next section, we will include a detailed review of three Solidity contracts and a review of the revisions in the seven (7) Vyper files revised from the Curve project. In a summary, Swerve project has added 153 lines of Solidity codes with a construction of two new contracts. The first contract YPoolDelegator stores a few state variables and delegates calls to the online "Curve.fi sUSD v2 Swap" contract. The second contract ZapDelegator stores a few state variables and delegates calls to the online "Curve.fi: sUSD v2 Deposit" contract. These codes are relatively simple contract constructions and the team have not found any vulnerabilities. Furthermore, Swerve made simple changes to three (3) of the Curve project Vyper files and the other four (4) Curve project Vyper files are identical. These changes are minimal and do not introduce new vulnerabilities. As we have mentioned in the preceding paragraph, we do not carry out an independent review on the Curve project codes. Our conclusion is that Swerve project is at least as secure as Curve project and we recommend that the user should be aware of the fact that Swerve smart contracts delegate their calls to the online API in “Curve.fi sUSD v2 Swap” contract and “Curve.fi: sUSD v2 Deposit” contract. Thus, the user will take the same risk that exist in the Curve project sUSD pool and also should take the same risks that may exist in the audited Curve project Vyper codes. Furthermore, the users should also be aware of the potential security vulnerabilities in the Swerve web portal and web-UI.

3. Detailed Inspection of Modules

1. APYoracle.sol

   This file defines an abstract contract Ypool which contains one function get_virtual_price(). Then it defines a contract APYOracle that contains three state variables

   a. pool (Ypool)
   b. poolDeployBlock
   c. blocksPerYear = 242584

   and a function getAPY() which will return the value:

   d. (pool.get_virtual_price()-1e18)*242584/(block.number-poolDeployBlock)

2. YPoolDelegator.sol

   This code contains the definition of the contract YPoolDelegator. The constructor for this contract is as follows:

```
constructor(address[4] memory _coinsIn, address[4] memory
_underlying_coinsIn,    address _pool_token, uint256 _A, uint256 _fee)
public {
    for (uint i = 0; i < 4; i++) {
            require(_coinsIn[i] != address(0));
            require(_underlying_coinsIn[i] != address(0));
            _balances.push(0);
            _coins.push(_coinsIn[i]);
            _underlying_coins.push(_underlying_coinsIn[i]);
    }
    A = _A;
    fee = _fee;
    admin_fee = 0;
    owner = msg.sender;
    kill_deadline = block.timestamp + 2 * 30 * 86400;
    is_killed = false;
    token = _pool_token;
  }
```

   Three “`public view returns`” functions are defined to query the state variable values for balances[], coins[], and underlying_coins[]. Furthermore, a fallback() function is defined to delegate call to “Curve.fi sUSD v2 Swap” contract address: `0xA5407eAE9Ba41422680e2e00537571bcC53efBfD`

3. ZapDelegator.sol

This code contains the definition of the contract ZapDelegator. The constructor for this contract is as follows

```
constructor(address[4] memory _coinsIn, address[4] memory
_underlying_coinsIn, address _curve, address _pool_token) public {
        for (uint i = 0; i < 4; i++) {
                require(_underlying_coinsIn[i] != address(0));
                require(_coinsIn[i] != address(0));
                _coins.push(_coinsIn[i]);
                _underlying_coins.push(_underlying_coinsIn[i]);
        }
        curve = _curve;
        token = _pool_token;
}
```

Two “`public view returns`” functions are defined to query the state variable values for _coins[] and _underlying_coins[]. Furthermore, a fallback() function is defined to delegate call to “Curve.fi: sUSD v2 Deposit” contract address: `0xFCBa3E75865d2d561BE8D220616520c171F12851`

4. ERC20CRV.vy

   This code is modified from “Curve Finance” codes: https://github.com/curvefi/curve-dao-contracts/blob/master/contracts/ERC20CRV.vy

   This modified code added the following constants (lines 50-52):

```
HOUR: constant(uint256) = 3600
DAY: constant(uint256) = 86400
WEEK: constant(uint256) = 86400 * 7
```

   This modified code changed the following original allocation (lines 52-67):

```
# Allocation:
# =========
# * shareholders - 30%
# * emplyees - 3%
# * DAO-controlled reserve - 5%
# * Early users - 5%
# == 43% ==
# left for inflation: 57%

# Supply parameters
INITIAL_SUPPLY: constant(uint256) = 1_303_030_303
INITIAL_RATE: constant(uint256) = 274_815_283 * 10 ** 18 / YEAR  # leading to 43% premine
RATE_REDUCTION_TIME: constant(uint256) = YEAR
RATE_REDUCTION_COEFFICIENT: constant(uint256) = 1189207115002721024  # 2 ** (1/4) * 1e18
RATE_DENOMINATOR: constant(uint256) = 10 ** 18
INFLATION_DELAY: constant(uint256) = 86400
```

   to the new allocation (lines 55-73):

```
# Allocation:
# ===========
# WE GIVE IT BACK TO YOU ... THE PEOPLE: 100%

# Supply parameters
INITIAL_SUPPLY: constant(uint256) = 0
INFLATION_DELAY: constant(uint256) = 3 * HOUR # Three Hour delay before minting may begin
RATE_DENOMINATOR: constant(uint256) = 10 ** 18
RATE_TIME: constant(uint256) = 2 * WEEK # How often the rate goes to the next epoch
INITIAL_RATE: constant(uint256) = 9_000_000 * 10 ** 18 / (2 * WEEK) # 9 million for the first 2 weeks
EPOCH_INITIAL_RATE: constant(uint256) = 9_000_000 * 10 ** 18 / YEAR # 9 million for the first year
thereafter
```

LATE_RATE: constant(uint256) = 3_000_000 * 10 ** 18 / YEAR # 3 million per year after
INITIAL_RATE_EPOCH_CUTTOF: constant(uint256) = 27 # After 52 Weeks use the late rate
FINAL_INFLATION_EPOCH: constant(uint256) = 157 # No more inflation after 6 years (0 epoch is the 2 week period)

The modified code changed the following original code (lines 112-119) in @internal def _update_mining_parameters()

```
if _rate == 0:
        _rate = INITIAL_RATE
else:
        _start_epoch_supply += _rate * RATE_REDUCTION_TIME
        self.start_epoch_supply = _start_epoch_supply
        _rate = _rate * RATE_DENOMINATOR / RATE_REDUCTION_COEFFICIENT
self.rate = _rate
```

to the following new one (lines 113-126):

```
if self.mining_epoch == 0:
        _rate = INITIAL_RATE
else:
        _start_epoch_supply += _rate * RATE_TIME
        if self.mining_epoch < INITIAL_RATE_EPOCH_CUTTOF:
                _rate = EPOCH_INITIAL_RATE
        elif self.mining_epoch >= FINAL_INFLATION_EPOCH:
                _rate = 0
        else:
                _rate = LATE_RATE
self.start_epoch_supply = _start_epoch_supply
self.rate = _rate
```

The revised code deleted the following function definition (lines 182-222): def mintable_in_timeframe(start: uint256, end: uint256) -> uint256

5. ERC20LP.vy
   This code is identical to "Curve Finance" code with a few format re-organization: https://github.com/curvefi/curve-dao-contracts/blob/master/contracts/testing/ERC20LP.vy

6. GaugeController.vy
   This code is revised from "Curve Finance" code: https://github.com/curvefi/curve-dao-contracts/blob/master/contracts/GaugeController.vy with the following added two lines of code (line 356 to 358)

```
# short circuit if single gauge and just give full weight
if self.n_gauges == 1 and self.gauges[0] == addr:
        return MULTIPLIER
```

   This code revision does not have any semantic changes from the original code.

7. LiquidityGauge.vy
   This code is identical to the "Curve Finance" code: https://github.com/curvefi/curve-dao-contracts/blob/master/contracts/LiquidityGauge.vy

8. Minter.vy
   This code is identical to “Curve Finance” code: https://github.com/curvefi/curve-dao-contracts/blob/master/contracts/Minter.vy

9. PoolProxy.vy
   This code is identical to “Curve Finance” code: https://github.com/curvefi/curve-dao-contracts/blob/master/contracts/PoolProxy.vy

10. VotingEscrow.vy
    This code is revised from “Curve Finance”: https://github.com/curvefi/curve-dao-contracts/blob/master/contracts/VotingEscrow.vy with the following changes.

    The revised code deleted the following lines (lines 108-111) from the original code

    ```
    # Checker for whitelisted (smart contract) wallets which are allowed to deposit
    # The goal is to prevent tokenizing the escrow
    future_smart_wallet_checker: public(address)
    smart_wallet_checker: public(address)
    ```

    The revised code deleted the following three function definitions (lines 165 to 195):

    ```
    commit_smart_wallet_checker(addr: address)
    apply_smart_wallet_checker()
    assert_not_contract(addr: address)
    ```

    The revised code deleted the following line (line 418) from the function create_lock(_value: uint256, _unlock_time: uint256)

    ```
    self.assert_not_contract(msg.sender)
    ```

    The revised code deleted the following line (line 438) from the function increase_amount(_value: uint256)

    ```
    self.assert_not_contract(msg.sender)
    ```

    The revised code deleted the following line (line 455) from the function increase_unlock_time(_unlock_time: uint256)

    ```
    self.assert_not_contract(msg.sender)
    ```

4. **Defects**

No defect has been identified.