Empty Set Dollar

Security Assessment

November 6, 2020

For :

[eqparenthesis] @ [Empty Set Squad]

By :

[Adrian Hetman] @ CertiK

adrian.hetman@certik.org

[Alex Papageorgiou] @ CertiK

alex.papageorgiou@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	Empty Set Dollar
Description	Empty Set Dollar is a stable coin, The ESD protocol is operated by a DAO that governs and regulates the supply of its stablecoin ESD. Its DAO utilizes a price oracle contract built on top of the ESD:USDC Uniswap v2 pool. This modular design allows for easy upgrades as Empty Set Dollar's ecosystem becomes more robust.
Platform	Ethereum; Solidity
Codebase	GitHub Repository
Commits	1. 9a92fe927f4f953240191f3dcb541aba7f97b830 2. 2e4d49a24aa70993dfdb19ca4371fb53ed76e445

Audit Summary

Delivery Date	Nov. 6, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Oct 6. - Oct 10. 2020 & Nov 3rd 2020 & Nov 5th 2020

Vulnerability Summary

Total Issues	15
Total Critical	0
Total Major	4
Total Minor	4
Total Informational	7

Executive Summary

The report represents the results of our engagement with Empty Set Dollar on their ESD stablecoin protocol. The initial review was conducted for 5 days: Oct. 6, 2020 - Oct. 10 2020 by Adrian Hetman and Alex Papageorgiou.

Many of the findings described here were alleviated by the client and discussed with the auditors' team. While we were conducting our audit, the client found two major issues:

1. Incorrect quorum calculation. Detailed write-up and fixed in this commit
2. Incorrect net supply computation with vesting feature. Fixed in commit

The team referenced the code relevant to the issues brought up by the ESD team to verify their existence and evaluate the implemented alleviations for validity and potential new flaws. Issue for quorum calculation was well described in the pull request and code changes for the fix were clean and didn't introduced new problems to the system. Same goes for the fix for incorrect net supply computation with vesting feature.

Findings

ID	Title	Type	Severity	Resolved
ESD-01	Dollar token object not set in contract state	Implementation	Major	✓
ESD-02	Pragma version not locked	Implementation	Informational	✓
ESD-03	Incorrect version of solidity	Implementation	Minor	✓
ESD-04	Lack of natspec comments	Implementation	Informational	!
ESD-05	Usage of custom ERC20 interface	Implementation	Informational	!
ESD-06	Variable shadowing	Implementation	Minor	✓
ESD-07	Checks-effects-pattern not used	Implementation	Minor	✓
ESD-08	Unused return value	Implementation	Minor	✓
ESD-09	Uses a dangerous strict equality on balance	Implementation	Major	✓

ID	Title	Type	Severity	Resolved
ESD-10	Inefficient greater-than comparison w/ zero	Optimization	Informational	✓
ESD-11	Suppression attack on advance() functionality	Implementation	Major	✓
ESD-12	Functions that could be declare external	Implementation	Informational	✓
ESD-13	Mark external calls safe / not safe	Implementation	Informational	✓
ESD-14	Dead code, not able to call certain functions.	Implementation	Major	✓
ESD-15	Specify the version of external contracts usage	Implementation	Informational	✓

ESD-01: Dollar token object not set in contract state

Type	Severity	Location	Status
Implementation	Major	Getters.sol L62-L64, Bonding.sol,Setter.sol	Closed

Description:

DAO depends on `dollar()` function from Getters.sol but variable this function returns i.e. `_state.provider.dollar;` is never initialised, so `dollar()` call will always return an empty token object resulting in not working system. Because of that, calls like this will fail for users.

```
function deposit(uint256 value) external onlyFrozenOrLocked(msg.sender) {
        dollar().transferFrom(msg.sender, address(this), value);
        incrementBalanceOfStaged(msg.sender, value);

        emit Deposit(msg.sender, value);
    }
```

Recommendation:

Initialize `_state.provider.dollar;` during migrations.

Alleviation:

Client comment:
"The protocol is already deployed and has been running for a couple of months now. When it was originally deployed it used the linked Deployer contract to initialize these state variables.
To clean up the codebase we removed the Deployer contract since then because it was no longer used in the protocol."

ESD-02: Pragma version not locked

Type	Severity	Location	Status
Implementation	Informational	General	Closed

Description:

Contract uses `pragma solidity ^0.5.17;` which is not recommended. Pragmas should be locked to specific compiler version and flags that they have been tested the most with. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler, which may have higher risks of undiscovered bugs.

Recommendation:

Avoid a floating pragma version (i.e. `pragma solidity ^0.5.17;` or `version>=0.5.0`) instead specify pragma version without using the caret symbol, i.e., `pragma solidity 0.5.17;`

Alleviation:

Issue resolved.

ESD-03: Incorrect version of solidity

Type	Severity	Location	Status
Implementation	Informational	Decimal.sol, LibEIP712.sol, Require.sol	Closed

Description:

The linked contracts necessitate a version too old to be trusted. We do not recommend using any latest version for deployment or older than 0.5.11, mostly if any changes were made in the optimizer or the language semantic. Versions 0.5.7 and 0.5.9 made changes to the optimizer; that's why we do not recommend using this version.

Recommendation:

Consider deploying with Solidity version 0.5.17 to be consistent with the rest of the contracts.

Alleviation:

Issue resolved.

ESD-04: Lack of natspec comments

Type	Severity	Location	Status
Implementation	Informational	All internal contracts	Open

Description:

Contract code is missing natspec comments, which helps understand the code and all the function's parameters.

Recommendation:

Please follow these style guides for adding natspec comments.

Alleviation:

The team will be fixing the issues in their own timeframe.

ESD-05: Usage of custom ERC20 interface

Type	Severity	Location	Status
Implementation	Informational	Getters.sol L30-L56, Setters.sol L29-L43	Acknowledged

Description:

The current approach whereby half of the ERC20 interface is implemented in Getters and half is implemented in Setters neither aids in maintainability or legibility of the codebase. The transfer, transferFrom, and approve functions all return false, which is not advised as it does not guarantee that that transaction will revert.

Recommendation:

We advise to inherit the ERC20 interface or even implementation and override any methods needed. For transfer, transferFrom and approve method returning false, please refer to the SafeERC20 `_callOptionalReturn` implementation.

Alleviation:

Client comment:
"Our goal with this was to surface an ERC20 interface for the ESDS accounting token, but make that token explicitly non-transferrable. The implementation has the normal transfer and approve methods, but reverts if they're called.
We're hesitant to refactor these into a single ERC20 contract. Right now our upgradeable state model restricts read/write access to Getters/Setters respectively with no other contracts allowed to interface directly with the state. Adding another ERC20 contract would extend this, opening up the surface area that has access to the state."

ESD-06: Variable shadowing

Type	Severity	Location	Status
Implementation	minor	Bonding.sol L29, Govern.sol L32, Market.sol L27	Closed

Description:

State variable `FILE` from `Permission.sol` is shadowed.

Recommendation:

Remove the state variable shadowing.

Alleviation:

The visibility of state variables affected was changed to private. Issue resolved.

ESD-07: Checks-effects-pattern not used

Type	Severity	Location	Status
Implementation	minor	Bonding.sol L54-L59, Comptroller.sol L36-L56, Comptroller.sol L71-L103, Pool.sol L50-L106	Closed

Description:

During `withdraw, burnFromAccount, redeemToAccount, burnRedeemable, increaseSupply` function calls state variables for balance are changed after transfers are done. Although functions from `Comptroller.sol` contract are internal, it's always good to follow the said pattern.

Recommendation:

It is recommended to follow checks-effects-interactions pattern for cases like this. It shields public functions from re-entrancy attacks. It's always a good practice to follow this pattern. `checks-effects-interactions` pattern also applies to ERC20 tokens as they can inform the recipient of a transfer in certain implementations.

Alleviation:

Issue resolved.

ESD-08: Unused return value

Type	Severity	Location	Status
Implementation	minor	Bonding.sol L48, Bonding.sol L55, Comptroller.sol L39, Comptroller.sol L47, Pool.sol L41-L66, Liquidity.sol L29-L41	Closed

Description:

Return variables for `transfer` and `transferFrom` are not utilized during function execution, dependent on the success of ERC20 transfers.

Recommendation:

It is advisable to use `SafeERC20` wrapper from OpenZeppelin not to be dependent on checking the return variable. The `SafeERC20` wrapper mitigates this by rendering the return variable optional.

Alleviation:

Issue resolved.

ESD-09: Uses a dangerous strict equality on balance

Type	Severity	Location	Status
Implementation	major	Comptroller.sol L106-L118, Pool.sol L156-L160	Closed

Description:

`balanceCheck()` relies on strict equality of ESD balances of Comptroller contract with what users have deposited and bonded. A malicious actor can send a ESD token to the contract, and `balanceCheck()` function will always fail as the first requirement will no longer be true.

Recommendation:

Remove strict equality. It's not recommended to rely on the balance of the contract as ether and tokens can always be forcibly sent to the contract. Usage of `balanceOf` in Getters.sol should help in this case.

Alleviation:

Issue resolved.

ESD-10: Inefficient greater-than comparison w/ zero

Type	Severity	Location	Status
Implementation	informational	Comptroller.sol L88, Comptroller.sol L99, Comptroller.sol L122, Comptroller.sol L130, Comptroller.sol L134, Market.sol L70, Govern.sol L40, Pool.sol L90, Pool.sol L110, Pool.sol L116	Closed

Description:

Within Solidity, unsigned integers are restricted to the non-negative range. As such, greater-than comparisons with the literal `0` are inefficient gas-wise.

Recommendation:

Consider converting the linked comparisons to inequality ones to optimize their gas cost.

Alleviation:

Issue resolved.

ESD-11: Suppression attack on advance() functionality.

Type	Severity	Location	Status
Implementation	Major	Implementation.sol L38-L56	Acknowledged

Description:

Epochs are advanced manually by sending an `advance()` transaction to the DAO, incentivizing users by minting reward ESD tokens to the sender upon successful advancement. As who is first is incentivized, it can lead to `Block stuffing` by an attacker, which can brute-force himself to win ESD reward. This is mostly exploitable where epochs are quicker (8h).

Recommendation:

https://consensys.github.io/smart-contract-best-practices/known_attacks/#mitigations

Alleviation:

Client comment:
"The goal here is to get anyone to call the function as soon as an epoch is available not necessarily to guarantee the average user has a fair shot at the advance reward.

In practice, we generally have been seeing a bot scoop this up with a decent margin in the first block after the timestamp which is exactly what we intended.
"

ESD-12: Functions that could be declare external

Type	Severity	Location	Status
Implementation	Informational	Oracle.sol L52, Oracle.sol L72	Acknowledged

Description:

Public functions that are never called by the contract should be declared external to save gas.

Recommendation:

Use the external attribute for functions never called from the contract.

Alleviation:

Client comment:
"These methods are explicitly public so that they can be overridden/mocked for testing"

ESD-13: Mark external calls safe / no safe

Type	Severity	Location	Status
Implementation	Informational	Oracle.sol L52 Oracle.sol L116	Open

Description:

External calls should be marked if they are safe / not safe.

Recommendation:

We advise that a comment is inserted in the preceding external call line that explains the call is safe or not safe.

Alleviation:

The team will be fixing the issues in their own timeframe.

ESD-14: Dead code, not able to call certain functions.

Type	Severity	Location	Status
Implementation	Informational	Pool.sol L141-L143	Closed

Description:

`emergencyWithdraw()` and `emergencyPause()` functions are only callable by DAO contract, but there is no function in DAO contracts that enable this call. We have observed this to be a proxy implementation; however, we should note that support for `emergencyWithdraw()` and `emergencyPause()` functions does not exist yet.

Recommendation:

Add functionality in DAO contracts to call those functions or implement `pass-through` fallback functionality in DAO for this to be called.

Alleviation:

Client comment:
"These functions are emergency hooks that can be used by the DAO in case the Pool contract gets bricked.

The usage flow would be to vote in new code in the DAO through the standard governance process that will use these methods to rescue stuck funds then presumably redistribute them by some means."

ESD-15: Specify the version of external contracts usage

Type	Severity	Location	Status
Implementation	Informational	External	Open

Description:

External contract usage is not specified with the version or Github commit.

Recommendation:

Add version or GitHub commit for all external contracts. It is advised to mention specific commit to check dependency with a specific version.

Alleviation:

The team will be fixing the issues in their own timeframe.

Icons explanation

 : Issue resolved

 : Issue not resolved / Acknowledged. The team will be fixing the issues in the own timeframe.

 : Issue partially resolved. Not all instances of an issue was resolved.