#HashEx
BLOCKCHAIN SECURITY

Zunami Stable (UZD)

smart contracts
final audit report

February 2023

hashex.org

contact@hashex.org

Contents

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn’t say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the Zunami Protocol team to perform an audit of their smart contract. The audit was conducted between 22/02/2023 and 26/02/2023.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code is available at @ZunamiProtocol/ZunamiStable GitHub repo after the e1b5b9d commit.

Update. The Zunami Protocol team has responded to this report, the updated code is located in the same repository after the 4364dd8 commit.

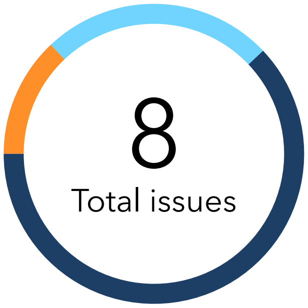
2.1 Summary

Project name	Zunami Stable (UZD)
URL	https://www.zunami.io/
Platform	Ethereum
Language	Solidity

2.2 Contracts

Name	Address
ElasticERC20	0xb40b6608B2743E691C9B54DdBDEe7bf03cd79f1c
ElasticERC20RigidExtension	0xb40b6608B2743E691C9B54DdBDEe7bf03cd79f1c
ElasticRigidVault	0xb40b6608B2743E691C9B54DdBDEe7bf03cd79f1c
ELT	
PricableAsset	0xb40b6608B2743E691C9B54DdBDEe7bf03cd79f1c
RigidAddressSet	0xb40b6608B2743E691C9B54DdBDEe7bf03cd79f1c
ZunamiElasticRigidVault	0xb40b6608B2743E691C9B54DdBDEe7bf03cd79f1c
UZD	0xb40b6608B2743E691C9B54DdBDEe7bf03cd79f1c
ZunamiRedistributor	
GitHub public repo	

3. Found issues



●	High	1 (13%)
●	Low	2 (25%)
●	Info	5 (62%)

C3. ElasticRigidVault

ID	Severity	Title	Status
C3-01	● Low	Gas optimization	✓ Partially fixed
C3-02	● Info	Spelling errors	✓ Resolved
C3-03	● Info	Standard noncompliance	✓ Partially fixed

C5. PricableAsset

ID	Severity	Title	Status
C5-01	● Info	Rebasing event can be called by anyone	✓ Acknowledged

C7. ZunamiElasticRigidVault

ID	Severity	Title	Status
C7-01	● Low	Gas optimization	✓ Resolved
C7-02	● Info	Address balance can be switched between elastic and rigid	✓ Acknowledged

C9. ZunamiRedistributor

ID	Severity	Title	Status
C9-01	● High	safeApprove is not spent	✓ Resolved

C10. GitHub public repo

ID	Severity	Title	Status
C10-01	● Info	Private keys in hardhat config file	✓ Acknowledged

4. Contracts

C1. ElasticERC20

Overview

Implementation of ERC-20 token standard with rebasing support for external getters `balanceOf()`, `totalSupply()`, `allowance()`. Balances are stored in a converted form, user-interacting functions receive values to be converted with the current asset price to be defined in the child contract. Inherits PricableAsset contract.

No issues were found.

C2. ElasticERC20RigidExtension

Overview

Extension for the ElasticERC20 to support a limited set of addresses, whose balances are not subjected to rebasing. Inherits ElasticERC20 contract.

No issues were found.

C3. ElasticRigidVault

Overview

Vault contract inheriting ElasticERC20RigidExtension, based on the ERC4626 implementation from OpenZeppelin library. Allows users to deposit and withdraw immutable underlying assets. Inherits ElasticERC20RigidExtension contract.

Issues

C3-01 Gas optimization ● Low ✓ Partially fixed

a. Double call of `_convertToNominalCached(value, Math.Rounding.Down)` in the `previewWithdraw()` function. Consider memoizing function result.

b. `convertToValue()` and `previewDeposit()` functions implement the same logic.

C3-02 Spelling errors ● Info ✓ Resolved

Typos in `reenterancy` and `transfered`.

C3-03 Standard noncompliance ● Info ✓ Partially fixed

The contract has NatSpec annotation stating it is an OpenZeppelin v4.7.0 ERC4626 fork. Nevertheless, the contract contains several ERC4626 noncompliances:

a. `withdraw()` and `previewWithdraw()` are in fact `redeem()` and `previewRedeem()` as `msg.sender` spends `owner` assets, but not shares;

b. The `deposit()` function must return the shares amount.

C4. ELT

Overview

ERC-20 standard token with a rebasing mechanism with Zunami token (0x2ffCC661011beC72e1A9524E12060983E74D14ce) as an underlying asset. Inherits ZunamiElasticRigidVault contract.

No issues were found.

C5. PricableAsset

Overview

Basing contract to handle price changes of the underlying asset, reported by an external oracle. A price update can be triggered externally at any time or automatically as frequently as once per specified period.

Issues

C5-01 Rebasing event can be called by anyone ● Info ✓ Acknowledged

The `assetPriceCached()` return value can be updated in two ways: by calling `_cacheAssetPriceByBlock()` once per `assetPriceCacheDuration()`, which is possible only by calling `ZunamiElasticRigidVault.redistribute()`, or by manually calling the `cacheAssetPrice()` function. Being the main rebasing mechanism, the `cacheAssetPrice()` function allows anyone to sync cached prices with the oracle by minting the needed supply. An arbitrary user can arbitrage by sandwiched trade-rebase-trade operations. Any contracts wanting to support UZD tokens should take into account this possibility of potentially non-synced price.

C6. RigidAddressSet

Overview

Enumerable set of addresses with fixed (not rebasing) balances, i.e. trading pools or pairs. Built on EnumerableSet from OpenZeppelin library.

No issues were found.

C7. ZunamiElasticRigidVault

Overview

Vault contract implementing additional fees and limits: withdraw fee can be set up to 5%, daily deposit and withdraw limits aren't restricted. Allows anyone to redistribute excessive tokens that are rebased over fixed addresses and not accounted for them. Inherits ElasticRigidVault and RigidAddressSet contracts.

Issues

C7-01 Gas optimization ● Low ✓ Resolved

a. Unnecessary read from storage of `dailyDepositDuration` and `dailyWithdrawDuration` in the `changeDailyDepositParams()` and `changeDailyWithdrawParams()` functions;

b. Possible double-write state variables `dailyDepositTotal`, `dailyWithdrawTotal` in the `_beforeDeposit()` and `_beforeWithdraw()` functions;

c. Multiple reads from storage of `dailyDepositDuration`/`dailyWithdrawDuration` and `dailyDepositTotal`/`dailyWithdrawTotal` in the `_beforeDeposit()`/`_beforeWithdraw()` function;

d. Possible double reads from storage of `withdrawFee` in the `_calcFee()` function;

e. Double call of `lockedNominalRigid()` in the `redistribute()` function;

f. Double read from storage of `redistributor` address in the `redistribute()` function;

g. Unchecked math could be used in the `redistribute()` function in L203.

C7-02 Address balance can be switched between elastic and rigid ● Info ✓ Acknowledged

The owners have access to both `addRigidAddress()` and `removeRigidAddress()` functions, making it possible to convert desired addresses between rebasing and non-rebasing states. This could result in malicious actions, for example, if converting to rebasing a protocol that doesn't support rebasing tokens. Third-party protocol owners must consider this possibility before adding the UZD token. Users must accept risks before using semi or fully decentralized protocols with UZD tokens.

C8. UZD

Overview

ERC-20 standard token with a rebasing mechanism with Zunami token (0x2ffCC661011beC72e1A9524E12060983E74D14ce) as an underlying asset. Inherits ZunamiElasticRigidVault contract.

No issues were found.

C9. ZunamiRedistributor

Overview

Redistribution contract for ZunamiElasticRigidVault tokens to withdraw excessive Zunami tokens and proportionally redistribute acquired underlying assets to Zunami's strategies.

Issues

C9-01 safeApprove is not spent ● High ✓ Resolved

The `requestRedistribution()` function call the Redistributor contract safely approves the Zunami token to the Zunami contract. However, the approved funds are spent neither in `zunami.delegateWithdrawal()` nor in `zunami.completeWithdrawals()`. Since `safeApprove()` reverts when allowance is greater than zero, the UZD redistribution method can be called only once and all other calls will cause an error due to "SafeERC20: approve from non-zero to non-zero allowance".

```
function requestRedistribution(uint256 nominal) external nonReentrant() {
    SafeERC20.safeTransferFrom(IERC20(zunami), _msgSender(), address(this), nominal);
    SafeERC20.safeApprove(IERC20(zunami), address(zunami), nominal);
    zunami.delegateWithdrawal(nominal, [uint256(0), 0, 0]);
}
```

Recommendation

Remove the line with `safeApprove`.

C10. GitHub public repo

Overview

The section describes security concerns related to public access to the project's GitHub repository.

Issues

C10-01 Private keys in hardhat config file ● Info ✓ Acknowledged

hardhat.config.ts file in the repo contains private keys to 6 test addresses. However, the addresses have signed transactions in the Ethereum mainnet. Ensure these addresses are not used for production or funds-holding purposes.

5. Conclusion

1 high, 2 low severity issues were found during the audit. 1 high, 1 low issues were resolved in the update.

The audited token implements IERC20 and IERC4626 (partially) interfaces with slight non-compliance of the last one. The token realizes custom logic on maintaining a price pegged to an underlying asset, if the token costs more than its underlying asset it redistributes surplus among investment strategies to gain extra revenue for token holders. The research didn't reveal stable tokens with the same price-maintaining strategy, so predicting an unpeg possibility is uncertain consequently investors should realize inherent risks.

This audit includes recommendations on code improvement and the prevention of potential attacks.

Appendix A. Issues’ severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

Appendix B. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

✉ contact@hashex.org

@hashex_manager

blog.hashex.org

linkedin

github

twitter

HashEx

BLOCKCHAIN SECURITY