

# Zunami

UZD

by Ackee Blockchain

*26.9.2022*



# Contents

1. Document Revisions .....	3
2. Overview .....	4
2.1. Ackee Blockchain .....	4
2.2. Audit Methodology .....	4
2.3. Finding classification .....	5
2.4. Review team .....	7
2.5. Disclaimer .....	7
3. Executive Summary .....	8
Revision 1 .....	8
Revision 1.1 .....	9
4. Summary of Findings .....	10
5. Report revision 1.0 .....	12
System Overview .....	12
Trust model .....	13
H1: Anybody can cause DoS of the protocol if the limits are set .....	14
H2: Daily deposit/withdrawal limits can be violated .....	16
H3: The <code>previewWithdraw</code> function does not include fee calculation .....	20
M1: Fees can be set to 100% anytime .....	22
M2: Two-phase transfer of ownership .....	24
M3: Renounce ownership .....	26
W1: Support for the meta-transactions .....	27
W2: Variable shadowing of the <code>owner</code> variable .....	29
W3: Floating pragma .....	30
W4: Usage of <code>solc</code> optimizer .....	31
W5: Missing package-lock.json .....	32
I1: Unnecessary call for <code>currentAssetPrice</code> .....	33

I2: Functions that could be external .....	34
I3: Typos .....	35
6. Report revision 1.1 .....	36
System Overview .....	36
Trust model .....	36
Appendix A: How to cite .....	37
Appendix B: Glossary of terms .....	38
Appendix C: Woke outputs .....	39
Detectors .....	39
Control flow graphs .....	39

# 1. Document Revisions

0.1	Draft report	19.9.2022
<a href="#">1.0</a>	Final report	23.9.2022
<a href="#">1.1</a>	Fix review	26.9.2022

## 2. Overview

This document presents our findings in reviewed contracts.

### 2.1. Ackee Blockchain

[Ackee Blockchain](#) is an auditing company based in Prague, Czech Republic, specializing in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run free certification courses [School of Solana](#), [Summer School of Solidity](#) and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, [Rockaway Blockchain Fund](#).

### 2.2. Audit Methodology

1. **Technical specification/documentation** - a brief overview of the system is requested from the client and the scope of the audit is defined.
2. **Tool-based analysis** - deep check with automated Solidity analysis tools and Woke is performed.
3. **Manual code review** - the code is checked line by line for common vulnerabilities, code duplication, best practices and the code architecture is reviewed.
4. **Local deployment + hacking** - the contracts are deployed locally and we try to attack the system and break it.
5. **Unit and fuzzy testing** - run unit tests to ensure that the system works as expected, potentially write missing unit or fuzzy tests.

## 2.3. Finding classification

A *Severity* rating of each finding is determined as a synthesis of two sub-ratings: *Impact* and *Likelihood*. It ranges from *Informational* to *Critical*.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Info*.

*Low* to *High* impact issues also have a *Likelihood*, which measures the probability of exploitability during runtime.

The full definitions are as follows:

### Severity

		<i>Likelihood</i>			
		High	Medium	Low	-
<i>Impact</i>	High	Critical	High	Medium	-
	Medium	High	Medium	Medium	-
	Low	Medium	Medium	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

Table 1. Severity of findings

## Impact

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.
- **Warning** - The issue cannot be exploited given the current code and/or configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.), but could be a security vulnerability if these were to change slightly. If we haven't found a way to exploit the issue given the time constraints, it might be marked as a "Warning" or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - The issue is on the borderline between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or configuration (see above) was to change.

## Likelihood

- **High** - The issue is exploitable by virtually anyone under virtually any circumstance.
- **Medium** - Exploiting the issue currently requires non-trivial preconditions.
- **Low** - Exploiting the issue requires strict preconditions.

## 2.4. Review team

Member's Name	Position
Jan Kalivoda	Lead Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

## 2.5. Disclaimer

We've put our best effort to find all vulnerabilities in the system, however our findings shouldn't be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.



## 3. Executive Summary

Zunami UZD is a stablecoin and a tokenized vault ([EIP-4626](#)) fork with Zunami LP token (ZLP) as an asset.

### Revision 1

Zunami engaged Ackee Blockchain to perform a security review of the Zunami UZD with a total time donation of 4 engineering days in a period between September 12 and September 16, 2022 and the lead auditor was Jan Kalivoda.

The full-repository audit has been performed on a private repository with commit [53dc20a](#).

We began our review by using static analysis tools, namely [Woke](#) and [Slither](#). [Woke](#) detectors didn't find any issue (see [Appendix C](#)) but generated control flow graphs helped us to identify the [H2: Daily deposit/withdrawal limits can be violated](#) issue.

Then we took a deep dive into the logic of the contracts. During the review, we paid special attention to:

- ensuring the price caching can not be exploited,
- deposit/withdrawal limits can not cause DoS,
- detecting possible reentrancies in the code,
- ensuring access controls are not too relaxed or too strict,
- looking for common issues such as data validation.

Our review resulted in 14 findings, ranging from Info to High severity. The most severe one is the possibility of DoS (see [H1: Anybody can cause DoS of the protocol if the limits are set](#)).

Ackee Blockchain recommends Zunami:

- fix all high severity issues since it is not recommended for deployment and use in this state,
- reconsider [Trust model](#) of the protocol as long as it heavily depends on [Owner](#),
- create documentation, including NatSpec code comments,
- address all other reported issues.

See the full report in chapter [Revision 1](#).

## Revision 1.1

Zunami provided an updated codebase that addresses some issues from this report. On September 23, 2022, [Ackee Blockchain](#) reviewed Zunami's fixes for the issues identified in this report. The updated commit was [335b852](#) and the scope was **only** related to the issues identified in this report.

The safety of the protocol now depends on the protocol administrators and the parameters they set up (such as withdraw/deposit limits), see the discussion of issue [H1: Anybody can cause DoS of the protocol if the limits are set](#).

See [Revision 1.1](#) for new changes and [Summary of Findings](#) for a status of each issue.

## 4. Summary of Findings

The following table summarizes the findings we identified during our review.

Unless overridden for purposes of readability, each finding contains:

- a *Description*,
- an *Exploit scenario*,
- a *Recommendation* and if applicable
- a *Solution*.

Many times, there might be multiple ways to solve or alleviate the issue, with varying requirements in terms of the necessary changes to the codebase. In that case, we will try to enumerate them all, making clear which solves the underlying issue better (albeit possibly only with architectural changes) than others.

	Severity	Reported	Status
<a href="#">H1: Anybody can cause DoS of the protocol if the limits are set</a>	High	<a href="#">1.0</a>	Acknowledged
<a href="#">H2: Daily deposit/withdrawal limits can be violated</a>	High	<a href="#">1.0</a>	Fixed
<a href="#">H3: The <del>preview</del>Withdraw function does not include fee calculation</a>	High	<a href="#">1.0</a>	Fixed
<a href="#">M1: Fees can be set to 100% anytime</a>	Medium	<a href="#">1.0</a>	Fixed
<a href="#">M2: Two-phase transfer of ownership</a>	Medium	<a href="#">1.0</a>	Fixed

	Severity	Reported	Status
<a href="#">M3: Renounce ownership</a>	Medium	<a href="#">1.0</a>	Acknowledged
<a href="#">W1: Support for the meta-transactions</a>	Warning	<a href="#">1.0</a>	Acknowledged
<a href="#">W2: Variable shadowing of the owner variable</a>	Warning	<a href="#">1.0</a>	Fixed
<a href="#">W3: Floating pragma</a>	Warning	<a href="#">1.0</a>	Acknowledged
<a href="#">W4: Usage of solc optimizer</a>	Warning	<a href="#">1.0</a>	Acknowledged
<a href="#">W5: Missing package-lock.json</a>	Warning	<a href="#">1.0</a>	Fixed
<a href="#">I1: Unnecessary call for currentAssetPrice</a>	Info	<a href="#">1.0</a>	Fixed
<a href="#">I2: Functions that could be external</a>	Info	<a href="#">1.0</a>	Acknowledged
<a href="#">I3: Typos</a>	Info	<a href="#">1.0</a>	Fixed

Table 2. Table of Findings

## 5. Report revision 1.0

The first revision of the audit report.

### System Overview

This section contains an outline of the audited contracts. Note that this is meant for understandability purposes and does not replace project documentation.

#### Contracts

Contracts we find important for better understanding are described in the following section.

##### PricableAsset

Inherits from [Ownable](#). The contract is used to obtain a price for an asset. This obtained price is cached until a new block is emitted.

##### ElasticERC20

Inherits from [PricableAsset](#) and [Context](#). The contract is ERC-20 fork, and in addition, it has functions allowing to operate with cashed prices from the [PricableAsset](#). Moreover, it utilizes convert functions between the value of the asset derived from the price and the nominal value.

##### ElasticVault

Inherits from [ElasticERC20](#). The contract is a fork of [EIP-4626](#). Allows changing daily deposit and withdrawal limits by [Owner](#). Fees are present only on withdrawal that can also be set by [Owner](#) from zero to 100%.

#### Actors

## Owner

The owner of [ElasticVault](#). He/she can set the price oracle, daily deposit and withdrawal limits, the fee percent and the address where are fees accumulated.

## Trust model

Users have to trust [Owner](#) that he/she will initialize the contract with a non-malicious token. But moreover, he/she will not cause DoS of the vault by setting inappropriate deposit/withdrawal limits. Also, [Owner](#) can set the withdrawal fees to 100% which will cause the user on withdrawal will lose all of his/her funds (see [M1: Fees can be set to 100% anytime](#)).

# H1: Anybody can cause DoS of the protocol if the limits are set

*High severity issue*

Impact:	High	Likelihood:	Medium
Target:	ElasticVault.sol	Type:	DoS

## Description

The protocol has variables that limit deposits and withdrawals for a specified duration.

`dailyWithdrawCountingBlock`

- A block from it is the duration calculated

`dailyWithdrawDuration`

- Duration of the limit in blocks

`dailyWithdrawLimit`

- Limit in a nominal value that can not be crossed during one period

`dailyWithdrawTotal`

- Initialized as a zero
- Is set to zero when `changeDailyWithdrawParams` is called or when `block.number > dailyWithdrawCountingBlock + dailyWithdrawDuration`
- Otherwise, increments its value with each withdrawal

Respectively, deposit limits work the same.

Users can not make withdrawals if the `dailyWithdrawTotal > dailyWithdrawLimit`. If the attacker has enough funds or if he/she utilizes flash

loans, he/she can spend all the limits in one block and nobody can make any deposit or withdrawal.

The likelihood of the attack can be lower or higher, depending on specific parameters that can be changed at any time.

### Exploit scenario

The `dailyWithdrawLimit` is eg.  $1e18$  and there are no fees on withdrawal. Alice will take a flash loan at the beginning of each period and using front-running ensures that she will spend all the limits in the block before anyone else will be able to interact. Moreover, since the price is cached for the whole block, she can be sure that a deposit and withdrawal will proceed very precisely. As a result, nobody can do anything until the owner will intervene by raising fees to make the attack too pricy or by setting a higher limit so it will not be affected.

### Recommendation

Do not use a limit for the entire contract, but rather consider using a limit for each user, so that each user's limit is only affected by themselves.

### Solution (Revision 1.1)

The issue is acknowledged by the team.

#### Client's response

*"The limits were made to prevent an attack on the pool on Curve. An attacker can use flash loan to unbalance the pool, and take out UZD at a discounted price. We want to limit the potential losses by setting the limits and commission."*

[Go back to Findings Summary](#)



## H2: Daily deposit/withdrawal limits can be violated

*High severity issue*

Impact:	High	Likelihood:	Medium
Target:	ElasticVault.sol	Type:	Logic error

*Listing 1. Excerpt from /contracts/ElasticVault.sol#L206-L209[ElasticVault.\_withdraw]*

```
206         if (block.number > dailyWithdrawCountingBlock +
    dailyWithdrawDuration) {
207             dailyWithdrawTotal = 0;
208             dailyWithdrawCountingBlock = dailyWithdrawCountingBlock
    + dailyWithdrawDuration;
209         }
```

*Listing 2. Excerpt from /contracts/ElasticVault.sol#L165-L168[ElasticVault.\_deposit]*

```
165         if (block.number > dailyDepositCountingBlock +
    dailyDepositDuration) {
166             dailyDepositTotal = 0;
167             dailyDepositCountingBlock = dailyDepositCountingBlock +
    dailyDepositDuration;
168         }
```

### Description

The `dailyDepositCountingBlock` (resp. `dailyWithdrawCountingBlock`) variable is updated only in this way:

```
dailyDepositCountingBlock = dailyDepositCountingBlock +
dailyDepositDuration;
```

So if the `withdraw` and `deposit` functions are not called for a longer time (depends on set values), it will cause the user can bypass the limit repeatedly, because assigning the new value to `dailyDepositCountingBlock` is not linked with the current number of a block and is only incremented by `dailyDepositDuration`.

## Exploit scenario

The protocol starts at block 10 with `dailyWithdrawCountingBlock` set to 10 and `dailyWithdrawDuration` set to 10. The `dailyWithdrawLimit` is 100.

- Bob will withdraw 100 tokens at block 11, so he can't withdraw (and anyone else) until block 21.
  - `dailyWithdrawCountingBlock` = 10
  - `dailyWithdrawTotal` = 100
- At block 21 Bob will withdraw 30 tokens and wait or forget about the protocol (as anyone else)
  - `dailyWithdrawCountingBlock` = 20
  - `dailyWithdrawTotal` = 30
- At block 31 Bob will withdraw 100 tokens
  - `dailyWithdrawCountingBlock` = 30
  - `dailyWithdrawTotal` = 100
- At block 41 Bob will withdraw 100 tokens again
  - `dailyWithdrawCountingBlock` = 40
  - `dailyWithdrawTotal` = 100
- And can still do it again because the `dailyWithdrawCountingBlock` increments only by `dailyWithdrawDuration` that was passed multiple times

As a result, in one period Bob repeatedly bypassed the limit.

## Recommendation

Adjust the logic of updating the `dailyWithdrawCountingBlock` and `dailyDepositCountingBlock` variables to mitigate this risk. Consider using `block.number` or `block.timestamp` in the calculation for the assignment of the new value. More supposedly the logic of limits will need to be completely reworked (see [H1: Anybody can cause DoS of the protocol if the limits are set](#)).

## Solution (Revision 1.1)

The issue is fixed by assigning `block.number` to `dailyDepositCountingBlock` (resp. `dailyWithdrawCountingBlock`) on the expiry of the duration. The logic is moved to `_beforeDeposit` (resp. `_beforeWithdraw`) hook that is called on each deposit (resp. withdrawal). This is implemented in a new contract `ZunamiElasticVault` that inherits from [ElasticVault](#).

```
function _beforeDeposit(
    address caller,
    address,
    uint256 value,
    uint256
) internal override {
    if (dailyDepositDuration > 0 && !hasRole(REBALANCER_ROLE, caller))
    {
        if (block.number > dailyDepositCountingBlock +
dailyDepositDuration) {
            dailyDepositTotal = 0;
            dailyDepositCountingBlock = block.number;
        }
        dailyDepositTotal += value;
        require(dailyDepositTotal <= dailyDepositLimit, 'Daily deposit
limit overflow');
    }
}
```

However, the logic of how the `dailyDepositTotal` variable is incremented is changed. Previously it was using a nominal value (`nominal`), but now it uses a calculated value (`value`) instead. That means if we have for example `assetPrice` equal to `2e18`, then if we deposit 400 tokens, we will receive 800 tokens and the limit will be incremented by 800. With a previous behavior, it was incremented by 400. The same applies to the `dailyWithdrawTotal` variable.

[Go back to Findings Summary](#)

### H3: The `previewWithdraw` function does not include fee calculation

*High severity issue*

Impact:	Medium	Likelihood:	High
Target:	ElasticVault.sol	Type:	Logic error

*Listing 3. Excerpt from /contracts/ElasticVault.sol#L117-L124[`ElasticVault.previewWithdraw`]*

```
117     /** @dev See {IERC4262-previewWithdraw}. */
118     function previewWithdraw(uint256 value) public view virtual
        override returns (uint256) {
119         return _convertToNominal(value, Math.Rounding.Up);
120     }
121
122     function _previewWithdrawCached(uint256 value) internal virtual
        returns (uint256) {
123         return _convertToNominalCached(value, Math.Rounding.Up);
124     }
```

#### Description

The `previewWithdraw` function should be inclusive of withdrawal fees. In the [ElasticVault](#) contract it isn't. As a result, users who want to preview their withdrawals are seeing different values than they get in real withdrawals.

#### Exploit scenario

Bob wants to preview his withdrawal. He finds out that he will withdraw 1000 tokens. Hence he will execute `withdraw`, but he will get only 800 tokens because the preview was without fees.

## Recommendation

Include the calculation of the withdrawal fees in the `previewWithdraw` function.

## Solution (Revision 1.1)

The issue is fixed by adding `_calcFee` function that is called on each `previewWithdraw` call. The function calculates the fee based on the current `fee` value and the amount of tokens that are going to be withdrawn.

[Go back to Findings Summary](#)

## M1: Fees can be set to 100% anytime

*Medium severity issue*

Impact:	High	Likelihood:	Low
Target:	ElasticVault.sol	Type:	Trust model

*Listing 4. Excerpt from /contracts/ElasticVault.sol#L65-L68[ElasticVault.changeWithdrawFee]*

```
65     function changeWithdrawFee(uint256 withdrawFee_) public onlyOwner {
66         require(withdrawFee_ <= FEE_DENOMINATOR, 'Bigger than 100%');
67         feePercent = withdrawFee_;
68     }
```

### Description

The percentage of fee per withdrawal can be set to 100% and that can be used against users to stop them from withdrawing or stealing funds from them.

### Exploit scenario

Bob wants to withdraw his funds, but Owner will front-run his transaction by setting fees to 100% and after Bob's transaction set it back. As a result, Bob loses all of his withdrawn funds.

### Recommendation

Create a new constant variable (eg. `MAXIMUM_FEE`) that is less than 100% and add another `require` statement to the `changeWithdrawFee` function.

### Solution (Revision 1.1)

The issue is fixed by adding `MAX_FEE` constant and proper `require` statement in

the `withdraw` function. This is implemented in a new contract `ZunamiElasticVault` that inherits from [ElasticVault](#).

```
uint256 public constant MAX_FEE = 50000; // 5%
```

[Go back to Findings Summary](#)



## M2: Two-phase transfer of ownership

*Medium severity issue*

Impact:	High	Likelihood:	Low
Target:	ElasticVault.sol, ElasticERC20.sol, PricableAsset.sol	Type:	Data validation

### Description

Multiple contracts in the codebase use the **owner** pattern for access control and also allow ownership transfer.

However, neither of the transfer functions has a robust verification mechanism for the new proposed owner. If a wrong owner address is passed to them, neither can recover from the error.

Thus passing a wrong address can lead to irrecoverable mistakes.

### Exploit scenario

The current owner Alice wants to transfer the ownership to Bob. Alice calls the **transferOwnership** function but supplies a wrong address by mistake. As a result, the ownership will be passed to a wrong address.

### Recommendation

One of the common and safer approaches to ownership transfer is to use a two-step transfer process.

Suppose Alice wants to transfer the ownership to Bob. The two-step process would have the following steps: Alice proposes a new owner, namely Bob. This proposal is saved to a variable **candidate**. Bob, the candidate, calls the

`acceptOwnership` function. The function verifies that the caller is the new proposed candidate, and if the verification passes, the function sets the caller as the new owner. If Alice proposes a wrong candidate, she can change it. However, it can happen, though with a very low probability that the wrong candidate is malicious (most often it would be a dead address). An authentication mechanism can be employed to prevent the malicious candidate from accepting the ownership.

### Solution (Revision 1.1)

The protocol is using Openzeppelin's [AccessControl](#) contract instead of the [Ownable](#) contract. The [AccessControl](#) contract is more flexible and allows for multiple roles to be defined.

A two-phase transfer of ownership is not implemented, but the described issue can be mitigated with the following flow. The `DEFAULT_ADMIN_ROLE` role can set this role to a new admin while the old admin still preserves his admin role and thus if there will be a mistake in granting the role, the old owner can revoke the new admin and add it to another one. If the new admin is correct then he can revoke himself.

The flow is different from the two-phase transfer of ownership, but in this case, it solves the problem and shouldn't affect the trust model, since access control rules are operated by the team of one organization.

[Go back to Findings Summary](#)

## M3: Renounce ownership

*Medium severity issue*

Impact:	High	Likelihood:	Low
Target:	ElasticVault.sol, ElasticERC20.sol, PricableAsset.sol	Type:	Data validation

### Description

All of the contracts are inherited from the [Ownable](#) contract. Due to that, the ownership can be renounced by the owner.

### Exploit scenario

The owner accidentally calls `renounceOwnership()`. Then nobody will ever be able to change the parameters of the protocol.

### Recommendation

Override the `renounceOwnership()` method to disable this feature if it is not intended. Otherwise, ignore this issue.

### Solution (Revision 1.1)

According to the client's response, it is a wanted feature to renounce the owner in the future.

[Go back to Findings Summary](#)

## W1: Support for the meta-transactions

Impact:	Warning	Likelihood:	N/A
Target:	**/*	Type:	Identity forgery

### Description

The protocol is using OpenZeppelin `Context` for potential support for meta-transactions in the future. Meta-transaction support creates a new attack surface via the `_msgSender` function. In a traditional smart contract, users can rely upon the `msg.sender` value is returning the expected value. However, when a body of the `_msgSender` function is adjusted with some additional logic, it can cause different behavior.

For example, in the following case, if a user is the trusted forwarder, he/she can pass any address to the `msg.data` and thus impersonate anyone.

*Listing 5. Example of the malicious `_msgSender` function implementation*

```
function _msgSender() internal override virtual view returns (address
ret) {
    if (msg.data.length >= 20 && isTrustedForwarder(msg.sender)) {
        // At this point we know that the sender is a trusted
forwarder,
        // so we trust that the last bytes of msg.data are the verified
sender address.
        // extract sender address from the end of msg.data
        assembly {
            ret := shr(96,calldataload(sub(calldatasize(),20))) ①
        }
    } else {
        ret = msg.sender;
    }
}
```

① This line will cut the last 20 bytes (address or another payload) from `msg.data` and return it

Ackee Blockchain wrote [a blog post](#) about this issue, and it is also mentioned in [EIP-2771](#) in the Security Considerations section.

## Recommendation

Pay special attention to the `_msgSender` function. If meta-transactions will be supported in the future, ensure the trusted forwarder address is securely handled.

[Go back to Findings Summary](#)

## W2: Variable shadowing of the owner variable

Impact:	Warning	Likelihood:	N/A
Target:	**/*	Type:	Variable shadowing

### Description

The owner variable from the [Ownable](#) contract is shadowed in several functions.

It can potentially cause issues in further development if the context will be accidentally swapped ([Owner](#) of the contract and local variable).

### Recommendation

Rename the global variable or function arguments.

### Solution (Revision 1.1)

The issue is fixed by replacing the [Ownable](#) contract with the [AccessControl](#) contract.

[Go back to Findings Summary](#)

## W3: Floating pragma

Impact:	Warning	Likelihood:	N/A
Target:	**/*	Type:	Compiler configuration

### Description

The project uses solidity floating pragma: ^0.8.0. Floating pragma allows using all the higher versions of the solidity compiler.

### Vulnerability scenario

A mistake in deployment can cause a version mismatch and thus an unexpected bug. Floating pragma also allows higher vulnerable versions (0.8.14) to be used in the project, which can also cause unintended behavior.

### Recommendation

Stick to one version, lock the pragma in all contracts, or set the range more precisely to avoid using unsafe versions. More information can be found at [SWC registry](#).

[Go back to Findings Summary](#)

## W4: Usage of solc optimizer

Impact:	Warning	Likelihood:	N/A
Target:	**/*	Type:	Compiler configuration

### Description

The project uses solc optimizer. Enabling solc optimizer [may lead to unexpected bugs](#). More significantly when it is used with the recent, not battle-tested versions, like the one specified in the Hardhat configuration file (0.8.16).

The Solidity compiler was audited in November 2018, and the audit [concluded](#) that the optimizer may not be safe.

### Vulnerability scenario

A few months after deployment, a vulnerability is discovered in the optimizer. As a result, it is possible to attack the protocol.

### Recommendation

Until the solc optimizer undergoes more stringent security analysis, opt-out using it. This will ensure the protocol is resilient to any existing bugs in the optimizer.

[Go back to Findings Summary](#)



## W5: Missing package-lock.json

Impact:	Warning	Likelihood:	N/A
Target:	**/*	Type:	Version control

### Description

The project repository does not contain the dependency lockfile (`package-lock.json`). Since the `package.json` file is not using strict versions then package versions can be different from the intended or tested ones. This can cause unknown behavior.

### Exploit scenario

The project was tested on some specific set of package versions. When the project is deployed it is deployed with different ones and it causes an unexpected bug in the protocol.

### Recommendation

Add `package-lock.json` to the repository and always use `npm ci` instead of `npm i` to ensure the lockfile is used.

### Solution (Revision 1.1)

The lockfile is added to the repository.

[Go back to Findings Summary](#)

## I1: Unnecessary call for currentAssetPrice

Impact:	Info	Likelihood:	N/A
Target:	PricableAsset.sol	Type:	Gas optimization

*Listing 6. Excerpt from /contracts/PricableAsset.sol#L45-L49[PricableAsset.assetPriceCached]*

```
45         uint256 currentAssetPrice = assetPrice();
46         if (_cachedAssetPrice < currentAssetPrice) {
47             _cachedAssetPrice = assetPrice();
48             emit CachedAssetPrice(_cachedBlock, _cachedAssetPrice);
49         }
```

### Description

The `assetPrice` function output is saved to the `currentAssetPrice` variable on the line 45 (see [Listing 6](#)), but on the line 47 is the function called again.

### Recommendation

Replace the call with the `currentAssetPrice` variable.

### Solution (Revision 1.1)

The call is replaced by the variable.

[Go back to Findings Summary](#)

## I2: Functions that could be external

Impact:	Info	Likelihood:	N/A
Target:	ElasticVault.sol	Type:	Gas optimization

### Description

The following functions are declared public even though they are not called internally anywhere.

- `changeDailyDepositParams`
- `changeDailyWithdrawParams`
- `changeWithdrawFee`
- `changeFeeDistributor`

### Recommendation

If functions are not called internally, they should be declared external. At least change the listed above. It helps to optimize gas consumption because function arguments do not have to be copied into memory.

[Go back to Findings Summary](#)

## I3: Typos

Impact:	Info	Likelihood:	N/A
Target:	PricableAsset.sol	Type:	Quality assurance

*Listing 7. Excerpt from /contracts/PricableAsset.sol#L32-L37[PricableAsset.assetPriceChahedParams]*

```

32     function assetPriceChahedParams()
33         public
34         view
35         virtual
36         returns (uint256 cachedBlock, uint256 cachedAssetPrice)
37     {

```

### Description

- [Listing 7](#) - The function name `assetPriceChahedParams` should be probably `assetPriceCachedParams`.

### Recommendation

Fix the typo.

### Solution (Revision 1.1)

Typos are fixed.

[Go back to Findings Summary](#)

## 6. Report revision 1.1

The following issues were changed:

- [H2: Daily deposit/withdrawal limits can be violated](#)
- [H3: The `previewWithdraw` function does not include fee calculation](#)
- [H3: The `previewWithdraw` function does not include fee calculation](#)
- [M1: Fees can be set to 100% anytime](#)
- [M2: Two-phase transfer of ownership](#)
- [W2: Variable shadowing of the `owner` variable](#)
- [W5: Missing `package-lock.json`](#)
- [I1: Unnecessary call for `currentAssetPrice`](#)
- [I3: Typos](#)

## System Overview

There are new contracts in the protocol:

- `ZunamiElasticVault` (inherits from [ElasticVault](#) and utilizes [AccessControl](#)),
- `ELT` (same as `UZD`, just different parameters for [ElasticERC20](#) constructor),

and several things were changed or moved to these contracts (see [changelog](#) above).

## Trust model

Owner can set fees only to 5% so [M1: Fees can be set to 100% anytime](#) is not applicable.

## Appendix A: How to cite

Please cite this document as:

[Ackee Blockchain](#), Zunami: UZD, 26.9.2022.

## Appendix B: Glossary of terms

The following terms might be used throughout the document:

### **Superclass/Ancestor of C**

A contract that C inherits/derives from.

### **Subclass/Child of C**

A contract that inherits/derives from C.

### **Syntactic contract**

A Solidity contract. May have an inheritance chain, and may be deployed.

### **Deployed contract**

An EVM account with non-zero code. If its source was written in Solidity, it was created through at least one syntactic contract. If that contract had superclasses (parents), it would be composed of multiple syntactic contracts.

### **Init/initialization function**

A non-constructor function that serves as an initializer. Often used in upgradeable contracts.

### **External entryptpoint**

A **public** or **external** function.

### **Public/Publicly-accessible function/entryptpoint**

An **external** or **public** function that can be successfully executed by any network account.

### **Mutating function**


A non-**view** and non-**pure** function.

## Appendix C: Woke outputs

This appendix shows the outputs from the static analyzer tool [Woke](#).

### Detectors

The following detectors were tested and all successfully passed.

 **Woke**

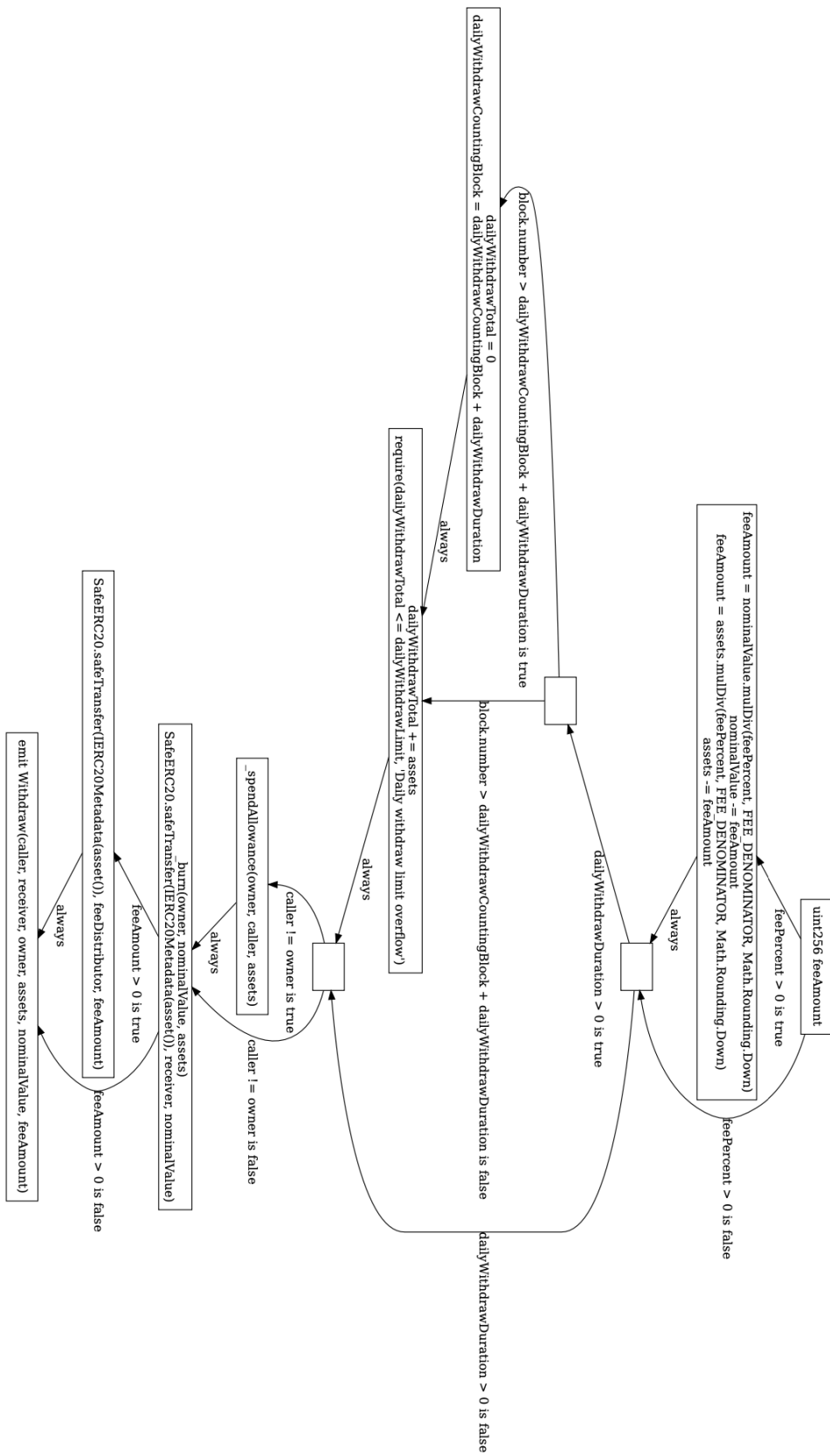
Using the following detectors:

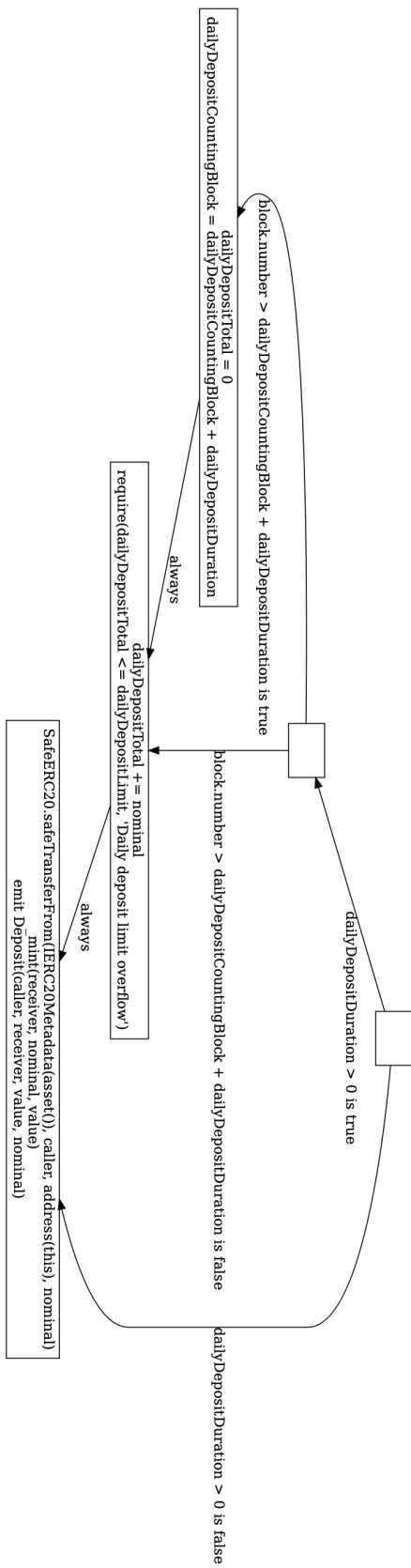
- **function-call-options-not-called**  
Function with call options actually is not called, e.g. `this.externalFunction(value: targetValue)`.
- **old-gas-value-not-called**  
Function with gas or value set actually is not called, e.g. `this.externalFunction.value(targetValue)`.
- **reentrancy**  
Detects re-entrancy vulnerabilities.
- **unchecked-function-return-value**  
Return value of a function call is ignored.
- **unsafe-address-balance-use**  
Address.balance is either written to a state variable or used in a strict comparison (`==` or `!=`).
- **unsafe-delegatecall**  
Delegatecall to an untrusted contract.
- **unsafe-selfdestruct**  
Selfdestruct call is not protected.

### Control flow graphs

The withdraw and deposit functions were chosen for analysis using control flow graphs.







# Thank You

Ackee Blockchain a.s.



Prague, Czech Republic



hello@ackeeblockchain.com



<https://discord.gg/z4KDUbuPxq>