Zellic

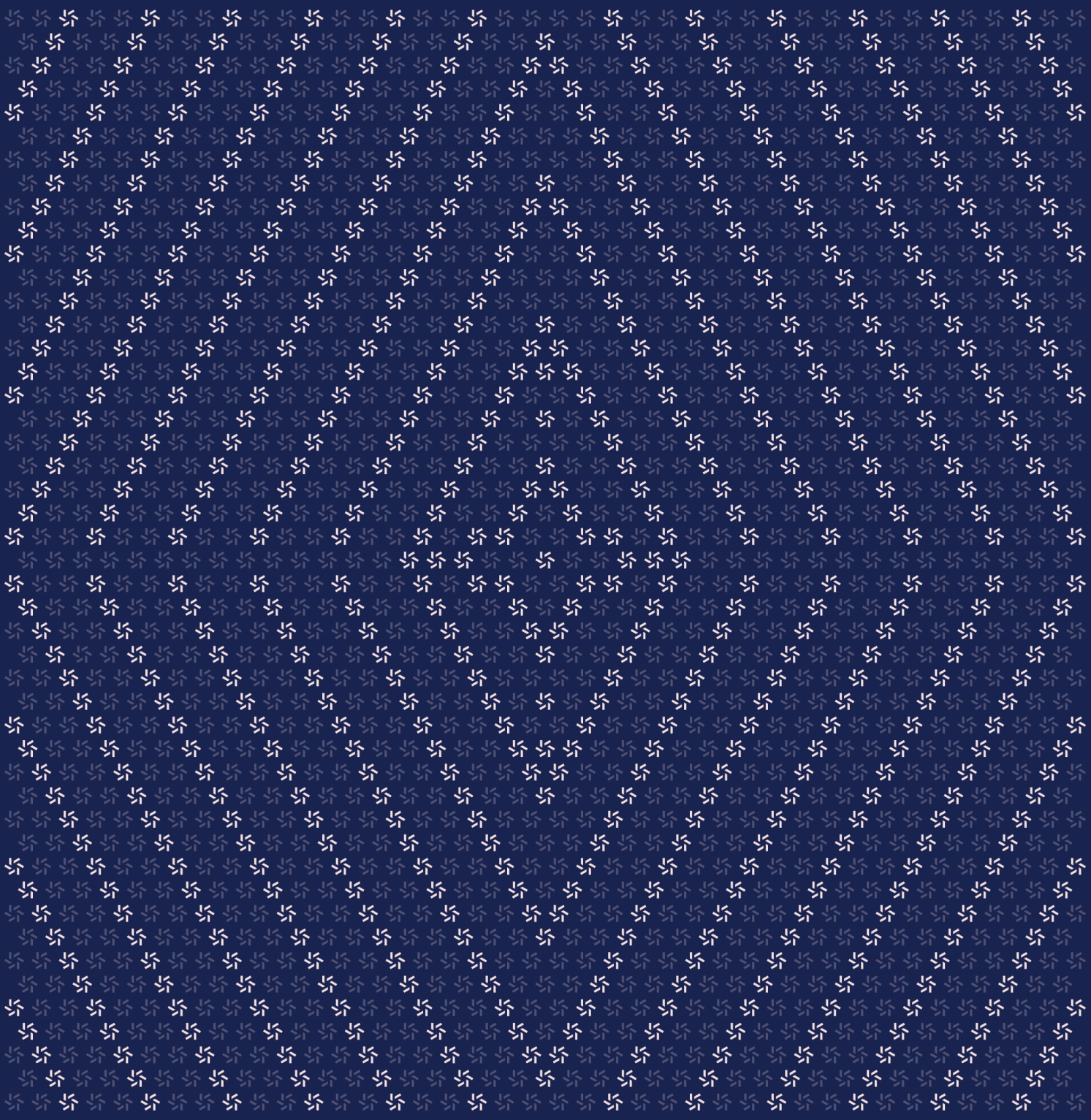
Prepared for
Henryk Sarat
Jacob Torres
Paxos

Prepared by
Qingying Jie
Seunghyeon Kim
Zellic

November 25, 2024

Paxos YBS

Smart Contract Security Assessment



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Paxos from November 18th to November 19th, 2024. During this engagement, Zellic reviewed Paxos YBS's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Do the PR changes cover the expected behavior?
- Do the PR changes cause DOS due to incomplete implementation?
- Are the PR changes implemented well, avoiding any bugs?
- Could an attacker do a financial attack because of the updated PRs?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

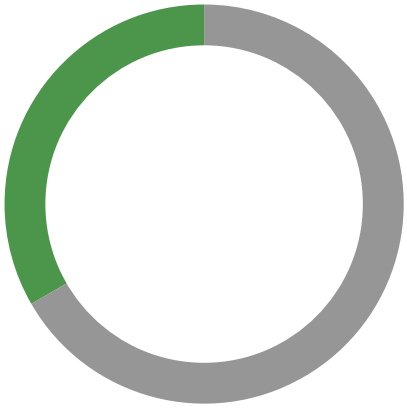
1.4. Results

During our assessment on the scoped Paxos YBS contracts, we discovered three findings. No critical issues were found. One finding was of low impact and the other findings were informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of Paxos in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	1
Informational	2



2. Introduction

2.1. About Paxos YBS

Paxos contributed the following description of Paxos YBS:

> The Paxos Yield Bearing Stablecoin (YBS) contract is a rebasing ERC20 token that allows paying out yield to end users by increasing the contract's rebase multiplier. This contract is currently used for Paxos International's Lift Dollar (USDL), issued on Ethereum and Arbitrum, to distribute yield from cash and cash equivalent reserves to its holders every day.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case

basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

Paxos YBS Contracts

Type	Solidity
Platform	EVM-compatible
Target	ybs-contract-internal changes for PRs #78, #80, and #81
Repository	https://github.com/paxosglobal/ybs-contract ↗
Version	`b22dc07af74081f3c28e58392c79f76788d7ef9a`

Programs	`YBSV1_1.sol`
	`wYBSV1.sol`
	`lib/*`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of three person-days. The assessment was conducted by two consultants over the course of two calendar days.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io

Chad McDonald
Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Qingying Jie
Engineer
qingying@zellic.io

Seunghyeon Kim
Engineer
seunghyeon@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

November 18, 2024	Kick-off call
November 18, 2024	Start of primary review period
November 19, 2024	End of primary review period

3. Detailed Findings

3.1. The `setNextMultiplier` has no upper limit

Target	YBSV1_1		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The `setNextMultiplier()` function has no upper bound on what can be set.

```
function setNextMultiplier(
    uint256 afterIncrMult_,
    uint256 multIncrTime_,
    uint256 expectedTotalSupply
) external onlyRole(REBASE_ADMIN_ROLE) {
    // Do not allow multIncrTime_ to be in the past.
    // If desired to only increase the multIncrTime use
    increaseRebaseMultiplier() with a zero rebaseRate.
    if (multIncrTime_ < block.timestamp) {
        revert RetroactiveRebase();
    }

    _setRebaseMultipliers(_getActiveMultiplier(), afterIncrMult_,
    multIncrTime_, expectedTotalSupply); // 2, 2, 0, any
}
```

This issue was reported by Paxos and we have included it for completeness.

Impact

If the rebase admin accidentally sets the multiplier to an extremely high value, Paxos would unintentionally pay out more than intended once it takes effect.

Recommendations

Consider adding logic to check that `rebaseRate` does not exceed the `maxRebaseRate`.

Remediation

Paxos added the logic for checking the rebase rate.

```
uint256 activeMult = _getActiveMultiplier();
if (afterIncrMult_ < activeMult) {
    revert InvalidRebaseMultiplier(afterIncrMult_);
}

uint256 rebaseRate = (afterIncrMult_ - activeMult) * _BASE / activeMult;
if (rebaseRate > maxRebaseRate) {
    revert InvalidRebaseRate(rebaseRate);
}
```

This issue has been acknowledged by Paxos, and a fix was implemented in commit d80ff009 ↗.

3.2. Public constant variables

Target	YBSV1_1		
Category	Optimization	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

There are some constant variables in the contract YBSV1_1 defined as public. The values of these variables can be read from the verified contract source code.

```
// [...]
bytes32 public constant SUPPLY_CONTROLLER_ROLE
    = 0x9c00d6f280439b1dfa4da90321e0a3f3c2e87280f4d07fea9fa43ff2cf02df2b;
// [...]
bytes32 public constant PAUSE_ROLE
    = 0x139c2898040ef16910dc9f44dc697df79363da767d8bc92f2e310312b816e46d;
// [...]
bytes32 public constant ASSET_PROTECTION_ROLE
    = 0xe3e4f9d7569515307c0cdec302af069a93c9e33f325269bac70e6e22465a9796;
// [...]
bytes32 public constant REBASE_ADMIN_ROLE
    = 0x1def088e742814a6c13355302c4cd95da961f82267b7106f2e38fbc5414a570e;
// [...]
bytes32 public constant REBASE_ROLE
    = 0x2cb8fee3430f011f8ea5df36a120dd5a293aa25c9ca88cc51159a94f41f768bb;
// [...]
bytes32 public constant WRAPPED_YBS_ROLE
    = 0x0d6cd32288790d7ef9cfeeb647381d8116dbc309cfa95f50cbb9e1956d87eb44;
```

Impact

A public storage variable has an implicit public function of the same name, which increases the size of the contract and causes more gas consumption in deployment.

Recommendations

Consider making the constant variables private.

Remediation

Paxos provided following message:

> Paxos intends to keep the variables public as our Operations team has processes that rely on the public methods to look up the role hash.

3.3. The WRAPPED_YBS_ROLE is not granted during the upgrade

Target	YBSV1_1		
Category	Business Logic	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The new version of the YBS contract introduces the WRAPPED_YBS_ROLE and adds checks in the _transfer function, in order to prevent direct transfers to the wrapped YBS contract.

```
function _transfer(address from, address to, uint256 amount)
    internal override {
    // [...]

    // To prevent inflation attacks on the wYBS contract, block direct
    transfers.
    if (hasRole(WRAPPED_YBS_ROLE, to) && (!hasRole(WRAPPED_YBS_ROLE,
    msg.sender) || hasRole(WRAPPED_YBS_ROLE, from)))
    revert WYBSTransferNotAllowed();

    // [...]
}
```

However, in the upgrade script upgrade.js, the WRAPPED_YBS_ROLE is not granted during the upgrade. An additional grant is required to enable the newly added checks.

```
const newContract = await ethers.getContractFactory('YBSV1_1');
const tx = await upgrades.upgradeProxy(PROXY_ADDRESS, newContract);
```

Impact

Malicious users might notice the newly added checks and front-run the role-grant transaction to transfer tokens directly to the wrapped YBS contract.

Recommendations

We recommend granting the WRAPPED_YBS_ROLE during the upgrade.

Remediation

Paxos provided following message:

> Paxos does not view this as a finding as the upgrade script is not being used and was not at the head of master when the audit started. Paxos uses an internal tool that can perform upgradeToAndCall, granting the role atomically during an upgrade.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Centralization Risk

There are certain centralized roles that exist in the system which we have highlighted below. Due to the nature of the protocol's design, privileged users with admin roles could induce changes that have financial repercussions for the system. We recommend Paxos outline the roles and abilities of privileged users in their documentation.

The roles and affected functions are below:

wYBSV1

- PAUSE_ROLE
  - pause
  - unpause
- ASSET_PROTECTION_ROLE
  - blockAccounts
  - unblockAccounts
  - seizeAssets
  - _withdraw
  - _beforeTokenTransfer

YBS1_1

- SUPPLY_CONTROLLER_ROLE
  - increaseSupply
  - decreaseSupply
- PAUSE_ROLE
  - pause
  - unpause
- ASSET_PROTECTION_ROLE
  - blockAccounts
  - blockAccountsFromReceiving
  - unblockAccounts
  - unblockAccountsFromReceiving
  - wipeBlockedAddress
- REBASE_ADMIN_ROLE
  - setRebasePeriod
  - setMaxRebaseRate
  - setNextMultiplier

- REBASE_ROLE
    - increaseRebaseMultiplier
- WRAPPED_YBS_ROLE
    - _transfer

5. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

5.1. Module: YBSV1_1.sol

Function: `blockAccounts(address[] addresses)`

This function adds the addresses as blacklisted.

Inputs

- `addresses`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Addresses to blacklist.

Branches and code coverage

Intended branches

- Add blacklisted accounts.
  - ☑ Test coverage

Negative behavior

- The caller must have the `ASSET_PROTECTION_ROLE`.
  - ☑ Negative test

Function: `increaseRebaseMultiplier(uint256 rebaseRate, uint256 expectedTotalSupply)`

This function increases the next multiplier and sets the increase time.

Inputs

- `rebaseRate`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be less than `maxRebaseRate`.

    - **Impact**: The increase rate for the next multiplier.
- `expectedTotalSupply`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None.
    - **Impact**: The expected total supply after the `rebaseRate` is applied.

Branches and code coverage

Intended branches

- Increase the next multiplier.
    - ☑ Test coverage
- Set the increase time.
    - ☑ Test coverage

Negative behavior

- The caller must have `REBASE_ROLE`.
    - ☑ Negative test
- The `multIncrTime` must be less than `block.timestamp`.
    - ☑ Negative test
- The given `rebaseRate` must be less than `maxRebaseRate`.
    - ☑ Negative test

Function: `setNextMultiplier(uint256 afterIncrMult_, uint256 multIncrTime_, uint256 expectedTotalSupply)`

This function sets the next rebase multiplier and multiplier's time increase. This function will be used in the following scenarios:

- Corrective actions when a pending increase is set (i.e., `multIncrTime` is in the future) — the `beforeIncrMult` should be active in this case and should not change.
- Explicitly setting the next multiplier and multiplier's time increase — the `afterIncrMult` should be active in this case and roll to `beforeIncrMult`.

Inputs

- `afterIncrMult_`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None.
    - **Impact**: The contract rebase multiplier after increase.
- `multIncrTime_`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Must be greater than or equal to `block.timestamp`.

- **Impact**: The multiplier's time increase.

- `expectedTotalSupply`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: The expected total supply after the increase based on `afterIncrMult_`.

Branches and code coverage

Intended branches

- Set the next rebase multiplier.
  - ☑ Test coverage

Negative behavior

- The given `multIncrTime_` must be greater than or equal to `block.timestamp`.
  - ☑ Negative test
- The caller must have `REBASE_ADMIN_ROLE`.
  - ☑ Negative test

Function: `unblockAccounts(address[] addresses)`

This function removes the addresses from the blacklist.

Inputs

- `addresses`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Addresses to blacklist.

Branches and code coverage

Intended branches

- Remove the addresses from the blacklist.
  - ☑ Test coverage

Negative behavior

- The caller must have the `ASSET_PROTECTION_ROLE`.
  - ☑ Negative test

5.2. Module: wYBSV1.sol

Function: `blockAccounts(address[] addresses)`

This function adds the addresses as blacklisted.

Inputs

- `addresses`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Addresses to blacklist.

Branches and code coverage

Intended branches

- Add blacklisted accounts.
  - ☑ Test coverage

Negative behavior

- The caller must have the `ASSET_PROTECTION_ROLE`.
  - ☑ Negative test

Function: `seizeAssets(address blockedAddr, address receiverAddr)`

This function wipes the shares of a blocked address and transfers assets to a receiver address.

Inputs

- `blockedAddr`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: The given address must be blocked.
  - **Impact**: Account to burn the shares.
- `receiverAddr`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: The given address must not be blocked.
  - **Impact**: Address to get the assets.

Branches and code coverage

Intended branches

- Burn the shares from the given blacklisted address.
  - ☑ Test coverage
- Transfer the assets to the given receiver address.
  - ☑ Test coverage

Negative behavior

- The caller must have the `ASSET_PROTECTION_ROLE`.
  - ☑ Negative test
- The `blockedAddr` must be blacklisted.
  - ☑ Negative test
- The `receiverAddr` must not be blacklisted.
  - ☑ Negative test

Function: `unblockAccounts(address[] addresses)`

This function removes the addresses from the blacklist.

Inputs

- `addresses`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Addresses to blacklist.

Branches and code coverage

Intended branches

- Remove the addresses from the blacklist.
  - ☑ Test coverage

Negative behavior

- The caller must have the `ASSET_PROTECTION_ROLE`.
  - ☑ Negative test

6. Assessment Results

At the time of our assessment, the reviewed code was not deployed to Ethereum Mainnet.

During our assessment on the scoped Paxos YBS contracts, we discovered three findings. No critical issues were found. One finding was of low impact and the other findings were informational in nature.

6.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.