Zellic

Prepared for
Henryk Sarat
Paxos

Prepared by
Katerina Belotskaia
Jaeeu Kim
Zellic

November 7, 2024

Paxos Stablecoin

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 October 14th to October 18th, 2024. During this engagement, Zellic reviewed Paxos Stablecoin'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:

- Is the storage layout compatible after the upgrade?
- Is Paxos's smart contract upgrade process safe?
- Are access controls implemented effectively to prevent unauthorized operations?
- Could an on-chain attacker exploit the system for unbounded minting?

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.

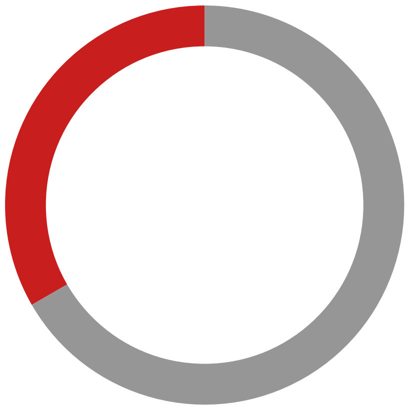
1.4. Results

During our assessment on the scoped Paxos Stablecoin contracts, we discovered three findings. One critical issue was found. 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	1
🟧 High	0
🟨 Medium	0
🟩 Low	0
⬜ Informational	2



2. Introduction

2.1. About Paxos Stablecoin

Paxos contributed the following description of Paxos Stablecoin:

> An ERC20 stablecoin contract that is used for all Paxos issued stablecoins. It centralizes all mint/burn to Paxos owned multi-sigs. Additionally, the stablecoin contract enables the delegation of mint/burn to external addresses up to a certain rate limit. Delegation to an external mint/burn will mainly be limited for warm minting/burning to a certain limit to not have to go through a cold signing process.

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 Stablecoin Contracts

Type	Solidity
Platform	EVM-compatible

Target	paxos-token-contracts
Repository	https://github.com/paxosglobal/paxos-token-contracts ↗
Version	`44992a908a222a8c453d86741fccba8f8b085e71`

Programs	`PaxosBaseAbstract`
	`RateLimit`
	`PYUSD`
	`USDP`
	`USDX`
	`BaseStorage`
	`PaxosTokenV2`
	`SupplyControl`
	`ECRecover`
	`EIP712Domain`
	`EIP3009`
	`ECRecover`
	`EIP2612`
	`EIP712`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of one and a half person-weeks. The assessment was conducted by two consultants over the course of one calendar week.

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:

Katerina Belotskaia
Engineer
kate@zellic.io ↗

Jaeeu Kim
Engineer
jaeeu@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

October 14, 2024	Kick-off call
October 14, 2024	Start of primary review period
October 18, 2024	End of primary review period

3. Detailed Findings

3.1. Frozen tokens available for cross-chain transfer

Target	PaxosTokenV2		
Category	Coding Mistakes	**Severity**	Critical
Likelihood	High	**Impact**	Critical

Description

The PaxosTokenV2 contract provides the `ASSET_PROTECTION_ROLE` with the ability to freeze an address balance. Tokens belonging to a frozen account are restricted in terms of transferability, and the allowance cannot be modified for such accounts.

Additionally, this contract supports cross-chain token transfers. The `decreaseSupplyFromAddress` function plays a crucial role in the process of transferring tokens between chains; tokens must be burned on the source chain before being transferred to the destination chain. This function reduces the balance of the account intending to send tokens. However, the issue is that this function does not verify whether the sender is a frozen account, which allows for bypassing the intended restrictions.

```
function decreaseSupplyFromAddress(uint256 value, address burnFromAddress)
    public virtual returns (bool success) {
    supplyControl.canBurnFromAddress(burnFromAddress, msg.sender);
    if (value > balances[burnFromAddress]) revert InsufficientFunds();

    balances[burnFromAddress] -= value;
    totalSupply_ -= value;
    emit SupplyDecreased(burnFromAddress, value);
    emit Transfer(burnFromAddress, address(0), value);
    return true;
}
```

Note: Paxos independently became aware of this issue. As this exists in the audit version, we document it in the report for completeness.

Impact

The frozen tokens can be transferred to another chain, which enables users to bypass intended restrictions.

Recommendations

Add a require statement to verify that the address is not frozen.

```
function decreaseSupplyFromAddress(uint256 value, address burnFromAddress)
    public virtual returns (bool success) {
   require(!_isAddrFrozen(burnFromAddress), "burnFromAddress frozen");
    supplyControl.canBurnFromAddress(burnFromAddress, msg.sender);
    if (value > balances[burnFromAddress]) revert InsufficientFunds();
    [...]
}
```

Remediation

This issue has been acknowledged by Paxos, and a fix was implemented in commit `1aeaaa82` ↗.

3.2. Allowing zero-address spender

Target	PaxosTokenV2		
Category	Code Maturity	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The OpenZeppelin ERC-20 implementation does not allow the `approve` function to approve the zero-address spender. However, the PaxosTokenV2 contract allows the `approve` function to approve the zero-address spender.

```
function approve(address spender, uint256 value)
    public whenNotPaused returns (bool) {
    if (_isAddrFrozen(spender) || _isAddrFrozen(msg.sender))
    revert AddressFrozen();
    _approve(msg.sender, spender, value);
    return true;
}

function _approve(address owner, address spender, uint256 value)
    internal override {
    allowed[owner][spender] = value;
    emit Approval(owner, spender, value);
}
```

Impact

This is not a security issue, but it would be a good practice to prevent the approval of the zero-address spender.

Recommendations

Add a check to prevent the approval of the zero-address spender.

Remediation

This issue has been acknowledged by Paxos, and a fix was implemented in commit `f401417c` ↗.

3.3. Gas optimization in `_removeAddressSet` function

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

Description

The `_removeAddressSet` function is used to clear `addressSet`. While the current implementation has O(n) complexity as each `remove` operation is O(1), it could be optimized for better gas efficiency by removing elements in reverse order. The current forward removal causes more storage updates due to array reordering, while removing elements in reverse order minimizes these operations.

```
function _removeAddressSet(EnumerableSet.AddressSet storage addressSet)
    private {
    uint256 length = EnumerableSet.length(addressSet);
    for (uint256 i = 0; i < length; ) {
        EnumerableSet.remove(addressSet, EnumerableSet.at(addressSet, i));
        unchecked {
            i++;
        }
    }
}
```

```
/**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function _remove(Set storage set, bytes32 value) private returns (bool) {
        // We read and store the value's index to prevent multiple reads from
    the same storage slot
        uint256 valueIndex = set._indexes[value];

        if (valueIndex != 0) {
            // Equivalent to contains(set, value)
            // To delete an element from the _values array in O(1), we swap the
    element to delete with the last one in
            // the array, and then remove the last element (sometimes called as
    'swap and pop').
```

```
            // This modifies the order of the array, as noted in {at}.

            uint256 toDeleteIndex = valueIndex - 1;
            uint256 lastIndex = set._values.length - 1;

            if (lastIndex != toDeleteIndex) {
                bytes32 lastValue = set._values[lastIndex];

                // Move the last value to the index where the value to delete is
                set._values[toDeleteIndex] = lastValue;
                // Update the index for the moved value
                set._indexes[lastValue] = valueIndex; // Replace lastValue's
    index to valueIndex
            }

            // Delete the slot where the moved value was stored
            set._values.pop();

            // Delete the index for the deleted slot
            delete set._indexes[value];

            return true;
        } else {
            return false;
        }
    }
```

Impact

This is not a security issue, but it would be a good practice to optimize the gas usage.

Following is the gas-usage simulation result when iterating 100 elements:

```
[PASS] testGasComparison() (gas: 8009931)
Logs:
  Forward removal gas used: 244059
  Backward removal gas used: 165719
  Gas saved using backward removal: 78340
```

Reverse removal is more gas-efficient than forward removal.

Recommendations

Use reverse removal to optimize the gas usage.

```
function _removeAddressSet(EnumerableSet.AddressSet storage addressSet)
    private {
    uint256 length = EnumerableSet.length(addressSet);
    for (uint256 i = length; i > 0; ) {
        unchecked {
            i---;
        }
        EnumerableSet.remove(addressSet, EnumerableSet.at(addressSet, i));
    }
}
```

Remediation

This issue has been acknowledged by Paxos, and a fix was implemented in commit `f401417c` ↗.

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 risks

This section documents the roles defined in Paxos Stablecoin and highlights the potential risks associated with them according to the contract and the specifications:

SupplyControl

- `DEFAULT_ADMIN_ROLE`
    - **Privileges:** Granting and revoking roles and contract upgrade.
    - **Restrictions:** Two-step process to transfer the `DEFAULT_ADMIN_ROLE` — the delay between these steps is set to three hours.
    - **Key-custody solution:** Assigned to a multi-sig wallet.
- `SUPPLY_CONTROLLER_MANAGER_ROLE`
    - **Privileges:** Managing `SUPPLY_CONTROLLER_ROLE` roles using `addSupplyController` and `removeSupplyController` functions, updating limit config through `updateLimitConfig`, and managing whitelist of accounts for minting actions using `addMintAddressToWhitelist` and `removeMintAddressFromWhitelist`.
    - **Restrictions:** Has no restrictions.
    - **Key-custody solution:** Assigned to a multi-sig wallet.
- `SUPPLY_CONTROLLER_ROLE`
    - **Privileges:** Increasing and decreasing total supply of the PaxosTokenV2 and minting and burning for the specified user.
    - **Restrictions:** The `SUPPLY_CONTROLLER_ROLE` role with `allowAnyMintAndBurnAddress` flag can burn tokens for any arbitrary user. Whitelists are used for minting, but the supply controller with the `allowAnyMintAndBurnAddress` flag can mint tokens for any arbitrary user. The minting limit is applied, but if `refillPerSecond` is zero, the limit check is skipped.
    - **Key-custody solution:** Assigned to a multi-sig wallet.
- `TOKEN_CONTRACT_ROLE`
    - **Privileges:** Can execute the `canMintToAddress` to verify that `msg.sender` actually can mint the specified amount of tokens for the provided account.
    - **Restrictions:** Has access to only one function, `canMintToAddress`.
    - **Key-custody solution:** Assigned to the PaxosTokenV2 token contract.

PaxosTokenV2

- `DEFAULT_ADMIN_ROLE`
  - **Privileges:** Granting and revoking roles, reclaiming all tokens at the current contract address, and updating the SupplyControl address.
  - **Restrictions:** Two-step process to transfer the `DEFAULT_ADMIN_ROLE`.
  - **Key-custody solution:** Assigned to a multi-sig wallet.
- `PAUSE_ROLE`
  - **Privileges:** Pausing and unpausing contract.
  - **Restrictions:** Has no restrictions.
  - **Key-custody solution:** Assigned to a multi-sig wallet.
- `ASSET_PROTECTION_ROLE`
  - **Privileges:** Can freeze an address balance from being transferred and can also unfreeze balance and wipe the balance of a frozen address.
  - **Restrictions:** Has no restrictions.
  - **Key-custody solution:** Assigned to a multi-sig wallet.

One additional safeguard that Paxos could consider implementing is the introduction of timelocks for sensitive operations such as assigning a new account for the `SUPPLY_CONTROLLER_ROLE` role, granting other roles, updating the whitelist, and unpausing. Timelocks are generally implemented by introducing a two-step process on dangerous operations, often with two separate functions.

5. System Design

This provides a description of the high-level components of the system and how they interact, including details like a function's externally controllable inputs and how an attacker could leverage each input to cause harm or which invariants or constraints of the system are critical and must always be upheld.

Not all components in the audit scope may have been modeled. The absence of a component in this section does not necessarily suggest that it is safe.

5.1. SupplyController

The SupplyController contract is responsible for controlling the supply of the Paxos tokens. It is used to add/remove SupplyController, update the rate limit, update the white address, and hook mint/burn functions.

Rate-limit system

The RateLimit library contract is used to limit the number of mints/burns in a short time. The rate limit is configurable by the controller manager. The rate limit is defined by the following parameters:

```
struct Storage {
    // Limit configuration
    LimitConfig limitConfig;
    // Remaining amount for the time period
    uint256 remainingAmount;
    //Timestamp of last event
    uint256 lastRefillTime;
}
struct LimitConfig {
    // Max amount for the rate limit
    uint256 limitCapacity;
    // Amount to add to limit each second up to the limitCapacity
    uint256 refillPerSecond;
}
```

Every `refillPerSecond` seconds, the `remainingAmount` will be filled up to `limitCapacity`. The `remainingAmount` will be decreased when minting/burning. If the `remainingAmount` is less than the amount to mint/burn, the mint/burn will be reverted. This rate limit protects stablecoins from minting/burning too much in a short time.

White address

SupplyController could have a list of white addresses that are allowed to mint. This could be updated by the controller manager. This whitelist will be used in hooking the mint function to check if the target address is allowed to mint.

Hooking mint/burn

SupplyController is used to hook mint and burn functions to check if the address is allowed to mint/burn. For example, the Paxos token contract could call the `canMintToAddress` function to check if the target address is allowed when minting.

```
function canMintToAddress(
    address mintToAddress,
    uint256 amount,
    address sender
) external onlySupplyController(sender) onlyRole(TOKEN_CONTRACT_ROLE) {
    SupplyController storage supplyController = supplyControllerMap[sender];
    if (
        !supplyController.allowAnyMintAndBurnAddress &&
        !EnumerableSet.contains(supplyController.mintAddressWhitelist,
    mintToAddress)
    ) {
        revert CannotMintToAddress(sender, mintToAddress);
    }
    RateLimit.Storage storage limitStorage
    = supplyController.rateLimitStorage;
    RateLimit.checkNewEvent(block.timestamp, amount, limitStorage);
}

function canBurnFromAddress(address burnFromAddress, address sender)
    external view onlySupplyController(sender) {
    SupplyController storage supplyController = supplyControllerMap[sender];
    if (!supplyController.allowAnyMintAndBurnAddress && sender !=
    burnFromAddress) {
        revert CannotBurnFromAddress(sender, burnFromAddress);
    }
}
```

5.2. PaxosTokenV2 and ERC-20 differences

PaxosTokenV2 supports the ERC-20 implementation, but it is not inherited from ERC-20.

Extended to EIP-2612 and EIP-3009

PaxosTokenV2 has extended to support EIP-2612 and EIP-3009. EIP-2612 is used to support the permit function, which allows users to sign a message to approve the token transfer. EIP-3009 is used to support batch transfer with authorization, which allows users to transfer multiple tokens with authorization.

Controlled mint/burn

PaxosTokenV2 holds `supplyControl`, which is a SupplyController instance. When minting or burning, PaxosTokenV2 will call `canMintToAddress` or `canBurnFromAddress` to check if the address is allowed to mint/burn and that the caller has the permission to mint/burn.

Some permission control

PaxosTokenV2 has some permissioned functions, such as `freeze`, `unfreeze`, and `wipeFrozenAddress`. These functions could be called by the `ASSET_PROTECTION_ROLE` role.

6. Assessment Results

At the time of our assessment, the reviewed code was deployed to Ethereum Mainnet.

During our assessment on the scoped Paxos Stablecoin contracts, we discovered three findings. One critical issue was found. 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.