Paxos Crosschain Messaging Enablement

Security Assessment

November 7, 2024

Prepared for:
Zach Petersen
Paxos

Prepared by: **Alexander Remie and Samuel Moelius**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
497 Carroll St., Space 71, Seventh Floor
Brooklyn, NY 11215
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

The sole canonical source for Trail of Bits publications is the Trail of Bits Publications page. Reports accessed through any source other than that page may have been modified and should not be considered authentic.

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Jeff Braswell, Project Manager
jeff.braswell@trailofbits.com

The following engineering director was associated with this project:

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultants were associated with this project:

Alexander Remie, Consultant
alexander.remie@trailofbits.com

Samuel Moelius, Consultant
samuel.moelius@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
September 9, 2024	Pre-project kickoff call
September 17, 2024	Status update meeting #1
September 24, 2024	Status update meeting #2
September 30, 2024	Delivery of report draft
October 9, 2024	Delivery of comprehensive report
October 28, 2024	Added Public Repositories section; updated Fix Review Results
November 7, 2024	Added Paxos’ comments to Fix Review Results

Executive Summary

Engagement Overview

Paxos engaged Trail of Bits to review the security of Paxos Crosschain Messaging Enablement.

A team of two consultants conducted the review from September 9 to September 27, 2024, for a total of four engineer-weeks of effort. Our testing efforts focused on the LayerZero integration, the updates to the existing upgradeable ERC20 token contract, and the new Solana program that represents a bridged token on the Solana blockchain. With full access to source code and documentation, we performed static and dynamic testing of the Paxos Crosschain Messaging Enablement repos, using automated and manual processes.

Observations and Impact

We only found issues of low and informational severity during this assessment.

On the Solidity side, the overall quality of the code is good, and we discovered no serious vulnerabilities during this engagement. The source-level documentation is also in a good state. The tests, however, can be improved with regard to LayerZero integration testing (TOB-PAXOSLZ-1). Furthermore, the usage of the `EnumerableSet` can be improved (TOB-PAXOSLZ-2, TOB-PAXOSLZ-10).

The Solana code could similarly benefit from improved documentation and testing. Specifically, the Solana code interacts with many accounts. The accounts are expected to bear several complex relationships, though those relationships are not documented. Furthermore, each Solana instruction is exercised by some test. However, the tests do not check for the existence of emitted events, nor whether they contain correct data.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that Paxos take the following steps:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.

- **Improve the LayerZero integration tests on both the Solidity and Solana side.** Such tests should be implemented whenever dealing with external protocols.

- **Document the ways that the Solana accounts relate to one another.** The relationships the accounts are expected to bear to one another are not obvious from the code. Documenting these relationships will save readers from having to discern them.

Finding Severities and Categories

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	0
Medium	0
Low	4
Informational	7
Undetermined	0

CATEGORY BREAKDOWN

Category	*Count*
Access Controls	2
Auditing and Logging	1
Data Validation	4
Testing	2
Undefined Behavior	2

Project Goals

The engagement was scoped to provide a security assessment of Paxos Crosschain Messaging Enablement. Specifically, we sought to answer the following non-exhaustive list of questions:

- Can LayerZero perform unbounded minting of a Paxos stablecoin?
- Can LayerZero break a Paxos stablecoin’s peg while minting within rate limits?
- Are there bugs in Paxos’s smart contract upgrade process?
- Can someone other than a supply controller perform supply controller functions?
- Could on-chain storage be incorrectly overwritten during an upgrade?
- Can Paxos contracts become bricked (via the upgrade process or otherwise)?
- Is there anything about the upgrade process that will make future upgrades more difficult?
- Can the token contract be safely upgraded from V1 to V2?
- Is the freezing mechanism implemented correctly?
- Do the updates to the EIP3009 and EIP2612 Solidity smart contracts pose any problems?

Project Targets

The engagement involved a review and testing of the targets listed below.

paxos-token-contract-internal

Repository	https://github.com/paxosglobal/paxos-token-contract-internal
Version	9fc474c23fcd5adb4450d12f51ab990720bd45c4
Type	Solidity
Platform	Ethereum

cross-chain-contracts-internal

Repository	https://github.com/paxosglobal/cross-chain-contracts-internal
Version	fe8604f936b87f9ca301a220b754dae3c0e1a10e 13ea95321c124a6123f049926d7d15a9ec62711f 0228cfc97dba51b48d1dfb2b789aa308eae2913b
Type	Solidity
Platform	Ethereum

solana-programs-internal

Repository	https://github.com/paxosglobal/solana-programs-internal
Version	b4f19114db91481262823cd1fbad5fc22a684f91
Type	Anchor
Platform	Solana

Public Repositories

The following are the public versions of the repositories we reviewed. We verified that the named commits match the fixed commits referenced in Fix Review Results (appendix D).

paxos-token-contract

Repository	https://github.com/paxosglobal/paxos-token-contract
Version	`ee1b21a93797c941e67c159ae2f376a4ce7e8843`

cross-chain-contracts

Repository	https://github.com/paxosglobal/cross-chain-contracts
Version	`826e367a5556efcb2fbc073e2924fbf6858b7e46`

solana-programs

Repository	https://github.com/paxosglobal/solana-programs
Version	`917a149327729d4f08893e79ec2c5a217590a717`

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- We verified the safety of upgrading the `XYZImplementationV1` Solidity smart contract to the `PaxosTokenV2` Solidity smart contract. We checked the storage layout of both contracts and all of their dependencies to verify that they align and do not overwrite incorrectly in terms of storage slot. We also reviewed the tests that are used to ensure that the upgrade between these two contracts is safe and sufficiently tested.

- We reviewed the deployment and configuration scripts present in the `cross-chain-contracts-internal` repository. These scripts will be used to deploy and configure the Solidity smart contracts. We looked for insufficient input validation and how that could negatively impact the script execution (TOB-PAXOSLZ-3). We looked for incorrect configuration of LayerZero (TOB-PAXOSLZ-4).

- We reviewed the `PaxosTokenV2` and `SupplyControl` Solidity smart contracts to look for flaws in the rate limiter, missing or incorrect access controls, incorrect usage of external libraries (TOB-PAXOSLZ-2, TOB-PAXOSLZ-10), missing or incorrect events, flaws in the freeze/unfreeze mechanism, and incorrect internal accounting.

- We reviewed the updates to the `EIP3009` and `EIP2612` Solidity smart contracts that added support for the freezing mechanism (TOB-PAXOSLZ-11). We looked for missing access controls and discrepancies of the access controls between these two contracts. We looked for ways the freezing mechanism interfered with the correct functioning of these contracts.

- We reviewed the tests for all three repositories to spot gaps in test coverage (TOB-PAXOSLZ-1, TOB-PAXOSLZ-9).

- We reviewed the `RateLimit` Solidity smart contract to look for ways the rate limit could be exceeded or skipped altogether without the “skip” being configured. We looked for incorrect storage updates of the `LimitConfig` struct. We reviewed the correctness of the `refill` function.

- We reviewed the minter controller program to check for general logic errors, misuse of Anchor constructs, accounts improperly marked as signing or writeable, account fields incorrectly populated, and seeds not sufficient to distinguish a PDA from others the program might use.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- On the Solana side, we were provided access to Paxos's minter controller program and LayerZero's sample Omnichain Fungible Token (OFT) implementation. However, the details of how they would integrate were not clear. Notably, Paxos told us that the OFT would not use the minter control program. Furthermore, we were provided no tests to verify that Paxos and LayerZero code could interoperate on the Solana blockchain.

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	The Solidity contracts use version 0.8.17 and therefore have automatic overflow checks. Regardless, there is barely any arithmetic in the implementation besides simple balance and approval updates. One exception is the `RateLimit.refill` function, but special care is taken to not revert if an overflow is detected; instead, the configured `limitCapacity` is returned.	**Satisfactory**
Auditing	Both the Ethereum and Solana contracts emit events with pertinent data at appropriate points in their execution. Note that some events currently emitted by Solana programs contain incorrect data. Furthermore, the Solana tests do not check for the existence of emitted events, nor whether they contain correct data.	**Satisfactory**
Authentication / Access Controls	We found no significant issues related to authentication or access controls. Nonetheless, administrative control of minter accounts does use a single-step transfer process, which could allow control of those accounts to be lost. We recommend that a two-step process be used.	**Satisfactory**
Complexity Management	The Solidity contracts are of low complexity and well documented.	**Satisfactory**
Decentralization	Paxos maintains administrative control over its contracts. However, this may be necessary to meet regulatory requirements. Put another way, it is not clear how decentralized Paxos could become while continuing to meet regulatory requirements.	**Weak**
Documentation	The minter controller program interacts with many accounts. However, the ways in which those accounts	**Moderate**

	relate to each other are not documented. For example, one account is called a “multisig.” However, the participants in that multisig, and what other roles they play, is not documented. On the Solidity side, all functions are well documented.	
Low-Level Manipulation	Both the Ethereum and Solana contracts use third-party libraries (e.g., OpenZeppelin and Anchor) that largely obviate the need for low-level manipulation.	**Satisfactory**
Testing and Verification	On the Ethereum side, LayerZero integration testing is minimal. We were not provided access to any LayerZero integration tests on the Solana side. As mentioned above, the Solana tests do not check for emitted events, nor whether they contain correct data.	**Weak**
Transaction Ordering	We found no significant issues related to transaction ordering.	**Satisfactory**

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Insufficient testing of cross-chain-contracts-internal contracts	Testing	Informational
2	EnumerableSet.add return value not checked	Data Validation	Low
3	Error-prone off-chain script	Data Validation	Low
4	setEnforcedOptions.js script contains outdated code	Data Validation	Informational
5	update_admin does not use a two-step transfer process	Access Controls	Informational
6	Account incorrectly marked optional	Data Validation	Informational
7	Unnecessary uses of mut	Undefined Behavior	Informational
8	minter_authority field is set incorrectly	Auditing and Logging	Low
9	Tests do not check emitted events	Testing	Informational
10	Reimplementation of library-provided function	Undefined Behavior	Informational
11	EIP3009 prohibits a frozen address from calling functions	Access Controls	Low

Detailed Findings

1. Insufficient testing of cross-chain-contracts-internal contracts

Severity: **Informational**	Difficulty: **Low**
Type: Testing	Finding ID: TOB-PAXOSLZ-1
Target: `cross-chain-contracts-internal/OFTProxySendTest.js`	

Description

The `cross-chain-contracts-internal` repository contains one test related to sending and receiving tokens. However, the test checks the balances stored within a single contract. Thus, the test does not exercise cross-chain functionality.

The relevant code appears in figure 1.1. Note that the balances checked both come from the `tokenFixture` contract. Preferably, the balances would come from two different contracts deployed to two different chains, even if those chains are simulated.

```
  95    // Fetching the final token balances of ownerA and ownerB
  96    const finalBalanceA = await tokenFixture.balanceOf(ownerA.address);
  97    const finalBalanceB = await tokenFixture.balanceOf(ownerB.address);
  98
  99    // Asserting that the final balances are as expected after the send
operation
 100    expect(finalBalanceA.eq(initialAmount.sub(tokensToSend))).to.be.true;
 101    expect(finalBalanceB.eq(tokensToSend)).to.be.true;
```

Figure 1.1: Excerpt of `OFTProxySendTest.js`
(`cross-chain-contracts-internal/test/OFTProxySendTest.js#95–101`)

For comparison, the corresponding lines from LayerZero's sample Hardhat test appear in figure 1.2.

```
// Fetching the final token balances of ownerA and ownerB
const finalBalanceA = await myOFTA.balanceOf(ownerA.address);
const finalBalanceB = await myOFTB.balanceOf(ownerB.address);

// Asserting that the final balances are as expected after the send operation
expect(finalBalanceA.eq(initialAmount.sub(tokensToSend))).to.be.true;
expect(finalBalanceB.eq(tokensToSend)).to.be.true;
```

Figure 1.2: Excerpt of LayerZero's sample Hardhat test

Exploit Scenario

Alice, a Paxos token user, tries to send her tokens from one chain to another. Alice's attempt fails because of a bug in Paxos's LayerZero integration. The bug might have been revealed through more thorough testing.

Recommendations

Short term, update `OFTProxySendTest.js` to perform simulated cross-chain transfers, like LayerZero's sample Hardhat test does. Doing so will increase confidence in the repository's contracts, and reduce the likelihood that they contain bugs.

Long term, consider using Foundry rather than Hardhat for testing. Foundry has certain advantages over Hardhat, such as logging from contracts, which can greatly aid in debugging. Since the `cross-chain-contracts-internal` repository is relatively young (currently, three commits), adapting it to use Foundry should not be terribly onerous.

References

- LayerZero: Testing Contracts

2. EnumerableSet.add return value not checked

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-PAXOSLZ-2
Target: `paxos-token-contract-internal/contracts/SupplyControl.sol`	

Description

The return value of the `EnumerableSet.add` function is not validated. As a result, trying to add a duplicate to the `mintAddressWhitelist` does not result in a revert, but instead the call will succeed and emit an incorrect event `MintAddressAddedToWhitelist`. This issue has no other effects besides an incorrect event emission.

Figure 2.1 shows that the `add` function (and the `_add` function) in the `EnumerableSet` library returns a Boolean that indicates if the addition was successful. If, for example, the addition failed because that value was already in the set, the `add` function will return `false`.

```
function add(AddressSet storage set, address value) internal returns (bool) {
    return _add(set._inner, bytes32(uint256(uint160(value))));
}

function _add(Set storage set, bytes32 value) private returns (bool) {
    if (!_contains(set, value)) {
        set._values.push(value);
        // The value is stored at length-1, but we add 1 to all indexes
        // and use 0 as a sentinel value
        set._indexes[value] = set._values.length;
        return true;
    } else {
        return false;
    }
}
```

Figure 2.1: The add and _add functions in OpenZeppelin's `EnumerableSet.sol`

Figure 2.2 shows that the return value from the `EnumerableSet.add` function is not checked. As a result, if the addition fails, this does not cause a revert. Instead, execution will continue and an event will be emitted that indicates that the `mintAddress` was added to the set. However, it was not added since it was already in the set before this call.

```
    function addMintAddressToWhitelist(
        address supplyController_,
        address mintAddress
    ) external onlyRole(SUPPLY_CONTROLLER_MANAGER_ROLE) onlySupplyController(supplyController_) {
        SupplyController storage supplyController = supplyControllerMap[supplyController_];
        EnumerableSet.add(supplyController.mintAddressWhitelist, mintAddress);
        emit MintAddressAddedToWhitelist(supplyController_, mintAddress);
    }
```

Figure 2.2: The `addMinterToWhitelist` function in `SupplyControl.sol`

Exploit Scenario

Alice, having the SUPPLY_CONTROLLER_MANAGER_ROLE, calls `addMintAddressToWhitelist` to add address 0x1 as minter. However, address `0x1` is already a registered minter. The transaction succeeds, even though no new minter was added.

Recommendations

Short term, validate the return value of the call to `EnumerableSet.add`, and revert if it is `false`. Alternatively, prepend the call to `EnumerableSet.add` with a call to `EnumerableSet.contains` to ensure that the value is not yet in the set (this pattern is used by the `removeMintAddressFromWhitelist` function).

Long term, read the source code of all external libraries/contracts to know how to write correct integrations. This helps to ensure that usage of external libraries/contracts is error-free.

3. Error-prone off-chain script

Severity: **Low**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-PAXOSLZ-3
Target: `cross-chain-contracts-internal/scripts/setConfig.js`	

Description

Missing validation of the variables destructured from `process.env` can lead to problems when executing `setConfig.js`, causing certain parts of the script to succeed while others fail. This would require re-running the script while commenting out parts that did succeed. Overall, this may lead to a very messy execution of this important configuration script, potentially even leading to incorrect configuration of the project.

```
const { ETH_OFT_ADDRESS, ETH_SEND_LIB_ADDRESS, ETH_RECEIVE_LIB_ADDRESS, ETH_ENDPOINT_CONTRACT_ADDRESS,
    SOLANA_EID, PAXOS_DVN_ADDRESS, LZ_DVN_ADDRESS, GOOGLE_DVN_ADDRESS, HORIZON_DVN_ADDRESS, NETHER_DVN_ADDRESS,
    CONFIRMATIONS, MAX_MESSAGE_SIZE, EXECUTOR_ADDRESS, NETWORK,
    NETWORK_URL, PRIVATE_KEY } = process.env;
```

Figure 3.1: Destructuring variables from `process.env` in `setConfig.js`

When destructuring variables from `process.env`, any unset environment variables will result in the value `undefined`. Depending on which variable, this could cause the script to break at some point or behave unexpectedly due to the coercion of `undefined`.

Exploit Scenario

The `setConfig.setLibraries` function is executed. Some environment variables are `undefined`. As a result, the `endpointContract.setSendLibrary` call succeeds but the `endpointContract.setReceiveLibrary` call fails. The script now needs to be re-executed with the `endpointContract.setSendLibrary` call commented out (since that was already executed successfully) so that only the `endpointContract.setReceiveLibrary` is executed.

Recommendations

Short term, validate all of the values destructured from `process.env`.

Long term, consider changing the framework of this repo from Hardhat to Foundry. Foundry is a much more versatile framework that allows for easier testing and management of complex projects. Foundry uses “scripts” written in Solidity to deploy and configure smart contracts. Foundry includes helper functions to read/write values from the local filesystem/environment, as well many other helpers that are useful for dealing with Solidity smart contracts.

References

- Foundry: Deploying contracts using Scripts
- Foundry: Helper functions for Scripts (and Tests)

4. setEnforcedOptions.js script contains outdated code	
Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-PAXOSLZ-4
Target: `cross-chain-contracts-internal/scripts/setEnforcedOptions.js`	

Description

The `setEnforcedOptions.js` script contains code that appears inconsistent. Paxos has acknowledged that the code was meant only for testing and should be removed.

The relevant code appears in figure 4.1. The `setEnforcedOptions` function is called with four different "enforced options." Two of them are for `ENDPOINT_EID`, and two are for `SOLANA_EID`. However, according to LayerZero documentation, the type of the destination chain (e.g., Ethereum or Solana) determines whether the options are interpreted as `(gas_limit, msg.value)` or `(compute_units, lamports)`. It seems unlikely that a single pair of values would make sense interpreted both ways.

```
    const GAS_OPTION = Options.newOptions().addExecutorLzReceiveOption(300000, 0).toHex() //0x00030100110100000000000000000000000000000493E0
    ...
      const enforced_options= [
        [
            ENDPOINT_EID,
            SEND_MSG_TYPE,
            GAS_OPTION
        ],
        [
            ENDPOINT_EID,
            SEND_AND_CALL_MSG_TYPE,
            GAS_OPTION
        ],
        [
            SOLANA_EID,
            SEND_MSG_TYPE,
            GAS_OPTION
        ],
        [
            SOLANA_EID,
            SEND_AND_CALL_MSG_TYPE,
            GAS_OPTION
        ]

      ]
      const transaction = await oftContract.setEnforcedOptions(enforced_options)
```

Figure 4.1: Excerpt of `setEnforcedOptions.js`
(`cross-chain-contracts-internal/scripts/setEnforcedOptions.js#9–39`)

Exploit Scenario

Alice, a Paxos admin, uses the `setEnforcedOptions.js` script to set Paxos endpoint options. However, for some of the endpoints, the settings are incorrect. Contracts that use the endpoints no longer work correctly.

Recommendations

Short term, remove the entries from `enforced_options` involving `ENDPOINT_EID`. Paxos has acknowledged that those entries were meant only for testing. Removing them will allow the remaining parts of the script to work correctly.

Long term, ensure that scripts with production impacts are tested regularly. For example, the script in 4.1 could be called using real or simulated endpoints, and then verifying that the endpoints work correctly. Taking these steps will help to ensure that scripts do not contain bugs.

References

- LayerZero: Solana Execution Options

5. update_admin does not use a two-step transfer process

Severity: **Informational**	Difficulty: **High**
Type: Access Controls	Finding ID: TOB-PAXOSLZ-5

Target: `solana-programs-internal/programs/minter-controller/src/instructions/update_admin.rs`

Description

The `update_admin` instruction performs an admin transfer in one step. The existing administrator could accidentally transfer control to a nonexistent address, effectively revoking all administrative control of the associated `Minter` account.

```
35    pub fn update_admin(ctx: Context<UpdateAdmin>, new_admin: Pubkey) -> Result<()> {
36        ctx.accounts.minter.admin = new_admin;
37        emit!(AdminUpdated{
38            minter_authority: ctx.accounts.minter.minter_authority,
39            mint_account: ctx.accounts.minter.mint_account,
40            admin: new_admin
41        });
42        Ok(())
43    }
```

Figure 5.1: Implementation of the `update_admin` instruction (`solana-programs-internal/programs/minter-controller/src/instructions/update_admin.rs#35–43`)

Exploit Scenario

Bob is an admin of a Paxos `Minter` account. Bob tries to transfer administrative control of the account to Alice, but mistypes her address. The `Minter` account's rate limit can no longer be updated, nor can new whitelisted addresses be added to the account. A new `Minter` account must be created, and all code that used the old account must be migrated to use the new one.

Recommendations

Short term, perform administrative transfers using a two-step process. That is, require the new administrator to invoke an instruction to accept administrative control. This will reduce the likelihood that administrative control of a `Minter` account is unintentionally lost.

Long term, write tests to ensure that the two-step process works correctly. This will help increase confidence in the process's correctness.

References

- Helius: A Hitchhiker's Guide to Solana Program Security

6. Account incorrectly marked optional

Severity: **Informational**	Difficulty: **Undetermined**
Type: Data Validation	Finding ID: TOB-PAXOSLZ-6
Target: `solana-programs-internal/programs/minter-controller/src/instructions/mint_token.rs`	

Description

In the `mint_token` instruction, the whitelist is marked optional (figure 6.1). The account's existence indicates that tokens can be minted to the associated `to_address`. Thus, the account's presence in the transaction should be mandatory and not optional.

```
    pub whitelist: Option<Account<'info, WhitelistedAddress>>,
```

Figure 6.1: Declaration of `whitelist` in `mint_token`'s `Context` struct (`solana-programs-internal/programs/minter-controller/src/instructions/mint_token.rs#44`)

Note: This finding is informational because we were unable to invoke the instruction without the account present. In other words, exploiting the issue may not be possible. Nonetheless, the instruction should be updated to prevent the possibility that the account is omitted.

Exploit Scenario

Alice, a Paxos admin, invokes the `mint_tokens` instruction but forgets to include the whitelisted account. Unbeknownst to her, the `to_address` to which Alice is trying to mint tokens is no longer whitelisted. Her transaction succeeds nonetheless.

Recommendations

Short term, make the whitelist account not optional in the `mint_tokens` instruction. This will help to ensure that the account is not unintentionally omitted when the `mint_tokens` instruction is called.

Long term, continue to test with non-whitelisted accounts, as the code does now. This will help to ensure that an error is produced when someone tries to mint to a non-whitelisted account.

7. Unnecessary uses of mut

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-PAXOSLZ-7
Target: Various source files in `solana-programs-internal/programs/minter-controller/src/instructions`	

Description

The code features many uses of `mut` on accounts that do not need to be writable. While this does not currently appear to be a vulnerability, the unnecessary uses of `mut` should be removed to help ensure the accounts are not unintentionally modified.

Examples appear in figures 7.1 and 7.2. When adding a minter, the mint account does not change. Similarly, when minting tokens, the whitelist account does not change. Thus, neither account should be marked as writeable.

```
18    #[account(mut)]
19    pub mint_account: Account<'info, Mint>,
```

Figure 7.1: Declaration of `mint_account` in the `add_minter` instruction's `Context` struct (`solana-programs-internal/programs/minter-controller/src/instructions/add_minter.rs#18–19`)

```
39    #[account(
40        mut,
41        seeds = [b"mint-whitelist", minter_authority.key().as_ref(),
mint_account.key().as_ref(), to_address.key().as_ref()],
42        bump = whitelist.bump,
43    )]
44    pub whitelist: Option<Account<'info, WhitelistedAddress>>,
```

Figure 7.2: Declaration of `whitelist` in the `mint_token` instruction's `Context` struct (`solana-programs-internal/programs/minter-controller/src/instructions/mint_token.rs#39–44`)

Exploit Scenario

Alice, a Paxos developer, is tasked with implementing a new feature in the Paxos mint controller program. Alice introduces a bug that causes an unintentional change in one of the program's accounts. The bug goes unnoticed because the account was incorrectly marked `mut`.

Recommendations

Short term, remove all unnecessary uses of `mut`. This will help ensure that unintentional changes are not made to the annotated accounts.

Long term, regularly review all uses of `mut`. This will help to identify cases where the annotation is used unnecessarily.

8. minter_authority field is set incorrectly

Severity: **Low**	Difficulty: **Low**
Type: Auditing and Logging	Finding ID: TOB-PAXOSLZ-8
Target: `solana-programs-internal/programs/minter-controller/src/instructions/{add_minter.rs`, `add_whitelisted_address.rs}`	

Description

In the `add_minter` instruction, the newly created `Minter` account's `minter_authority` field is set incorrectly. Similarly, in the `add_whitelisted` instruction, the newly created `WhitelistedAccount` account's `minter_authority` field is set incorrectly. The incorrectly set fields cause incorrect events to be emitted.

```
 14        #[account()]
 15        pub minter_authority: Signer<'info>,
 ...
 21        #[account(
 22           init,
 23           payer = payer,
 24           space = 8 + 32 + 32 + 32 + 8 + 8 + 8 + 8 + 1, //8 discriminator + 32
minter_authority + 32 mint account + 32 admin + 8 capacity + 8 tokens + 8
refill_per_second + 8 last_refill_time + 1 bump
 25           seeds = [b"mint-authority", minter_authority.key().as_ref(),
mint_account.key().as_ref()], bump
 26       )]
 27        pub minter: Account<'info, Minter>,
 ...
 32        ctx.accounts.minter.minter_authority = ctx.accounts.minter.key();
```

Figure 8.1: Setting of the `minter_authority` field in the `add_minter` instruction (`solana-programs-internal/programs/minter-controller/src/instructions/add_minter.rs#14–32`)

```
 18       #[account()]
 19        pub minter_authority: UncheckedAccount<'info>,
 ...
 25        #[account(
 26           mut,
 27           has_one = admin,
 28           seeds = [b"mint-authority", minter_authority.key().as_ref(),
mint_account.key().as_ref()], bump = minter.bump
 29       )]
 30        pub minter: Account<'info, Minter>,
 ...
```

```
        ctx.accounts.whitelisted_address.minter_authority =
ctx.accounts.minter.key();
```

Figure 8.2: Setting of the `minter_authority` field in the `add_whitelisted_address` instruction (`solana-programs-internal/programs/minter-controller/src/instructions/add_whitelisted_address.rs#18–50`)

For both `Minter` and `WhitelistedAddress` accounts, the `minter_authority` field is used only to generate events. Nonetheless, the problem should be corrected to ensure that future code changes do not introduce more severe bugs.

Exploit Scenario

Alice runs off-chain software that monitors the Paxos minter controller program. The incorrect events generated by the program cause Alice's software to behave incorrectly.

Recommendations

Short term, in both the `add_minter` and `add_whitelisted_address` instructions, set the `minter_authority` field to the address of the `minter_authority` account. This will fix a bug that causes incorrect events to be emitted.

Long term, for instructions that use `Minter` accounts, add checks to ensure that the `minter_authority` and `mint_account` fields are set correctly, e.g., like in figure 8.3. Take similar steps for instructions that use `WhitelistedAddress` accounts. Such actions will help prevent similar bugs from arising. See also TOB-PAXOSLZ-9.

```
    #[account(
        has_one = minter_authority,
        has_one = mint_account,
        has_one = admin,
        seeds = [b"mint-authority", minter_authority.key().as_ref(),
mint_account.key().as_ref()],
        bump = minter.bump
    )]
    pub minter: Account<'info, Minter>,
```

Figure 8.3: Sample code for checking `minter_authority` and `mint_account` fields

9. Tests do not check emitted events

Severity: **Informational**	Difficulty: **Low**
Type: Testing	Finding ID: TOB-PAXOSLZ-9
Target: `solana-programs-internal/tests/minter_controller.ts`	

Description

The minter controller's tests do not check for emitted events, nor the data contained in them. Events are crucial for many off-chain applications. The minter controller should include tests to ensure the correctness of the events it emits.

Exploit Scenario

See the exploit scenario from TOB-PAXOSLZ-8.

Recommendations

Short term, for each instruction that emits an event, ensure that there is at least one test that checks for that event, and that verifies the correctness of the event's data. This will help ensure the correct behavior of software monitoring those events. Sample code for checking for an event appears in figure 9.1.

```
let foundEvent = false
try {
  const tx = await minterControllerProgram.methods
  .addWhitelistedAddress()
  .accounts({
    payer: payer.publicKey,
    minterAuthority: mintAuthorityKeypair.publicKey,
    toAddress: payer.publicKey,
    admin: adminKeypair.publicKey,
    mintAccount: mintPDA
  })
  .signers([adminKeypair])
  .rpc(confirmOptions);
  let t = await provider.connection.getTransaction(tx, {
    commitment: "confirmed",
    maxSupportedTransactionVersion: 0
  });
  const eventParser = new EventParser(minterControllerProgram.programId, new BorshCoder(minterControllerProgram.idl));
  const events = [...eventParser.parseLogs(t.meta.logMessages)];
  foundEvent = events.some((event) => event.name === 'whitelistedAddressAdded'
    && event.data.minterAuthority.toString() ===
mintAuthorityKeypair.publicKey.toString()
        && event.data.mintAccount.toString() === mintPDA.toString()
        && event.data.toAddress.toString() === payer.publicKey.toString()
      );
    } catch (err) {
      console.log(err)
      assert.fail('Error not expected while adding whitelisted_address')
    }
    assert.isTrue(foundEvent)
```

Figure 9.1: Sample Anchor code for checking for an event

Long term, run all tests (including any newly introduced ones) in CI. This will ensure that the tests are run regularly, which will increase their chances of exposing bugs.

10. Reimplementation of library-provided function

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-PAXOSLZ-10
Target: `paxos-token-contract-internal/contracts/SupplyControl.sol`	

Description

The `SupplyControl` contract implements `_getAddressSet` to convert an `EnumerableAddressSet` into an array. However, the `EnumerableSet` library itself provides a `values` function to convert the set to an array, which is also more gas efficient.

```
    function _getAddressSet(EnumerableSet.AddressSet storage addressSet) private view returns (address[] memory) {
        uint256 length = EnumerableSet.length(addressSet);
        address[] memory addresses = new address[](length);
        for (uint256 i = 0; i < length; ) {
            addresses[i] = EnumerableSet.at(addressSet, i);
            unchecked {
                i++;
            }
        }
        return addresses;
    }
```

Figure 10.1: The `_getAddressSet` function in `SupplyControl.sol`

```
    function values(AddressSet storage set) internal view returns (address[] memory) {
        bytes32[] memory store = _values(set._inner);
        address[] memory result;

        /// @solidity memory-safe-assembly
        assembly {
            result := store
        }

        return result;
    }
```

Figure 10.2: The `values` function in OpenZeppelin's `EnumerableSet.sol`

Recommendations

Short term, remove the `_getAddressSet` function and instead use the `EnumerableSet.values` function to turn the set into an array.

Long term, read the source code of all external libraries/contracts to know how to write correct integrations. This helps to ensure that usage of external libraries/contracts is error-free.

11. EIP3009 prohibits a frozen address from calling functions

Severity: **Low**	Difficulty: **Medium**
Type: Access Controls	Finding ID: TOB-PAXOSLZ-11
Target: `paxos-token-contract-internal/contracts/lib/EIP2616.sol`	

Description

Unlike the `EIP2612` contract, the `EIP3009` contract disallows a “frozen” account from calling any of its functions. This limitation does not seem necessary since these calls are only to relay a call from another account, that other account cannot be frozen, and such checks exist. Furthermore, Paxos explained that such a limitation should not exist.

```
    function cancelAuthorization(
        address authorizer,
        bytes32 nonce,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external whenNotPaused {
        if (_isAddrFrozen(msg.sender) || _isAddrFrozen(authorizer)) revert AddressFrozen();
```

Figure 11.1: Excerpt from the `cancelAuthroization` function in `EIP3009.sol`

```
    function _transferWithAuthorization(
        bytes32 typeHash,
        address from,
        address to,
        uint256 value,
        uint256 validAfter,
        uint256 validBefore,
        bytes32 nonce,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) internal {
        if (block.timestamp <= validAfter) revert AuthorizationInvalid();
        if (block.timestamp >= validBefore) revert AuthorizationExpired();

        if (_isAddrFrozen(msg.sender)) revert AddressFrozen();
```

Figure 11.2: Excerpt from the `_transferWithAuthorization` function in `EIP3009.sol`

Exploit Scenario

Carol's account is frozen by the Paxos team due to a real-world legal order. Bob authorizes a transfer. Carol calls `transferWithAuthorization` to relay Bob's authorized transfer. The transaction reverts.

Recommendations

Short term, remove the `_isAddrFrozen(msg.sender)` checks highlighted in figures 11.1 and 11.2 from the implementation.

Long term, add tests that ensure that all state-changing functions that are only meant to relay a signed message do not revert if the caller's account is frozen.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels

Severity	Description
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels

Difficulty	Description
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage
Transaction Ordering	The system’s resistance to transaction-ordering attacks

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.

Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Non-Security-Related Recommendations

The following recommendations are not associated with specific vulnerabilities. However, implementing them may enhance code readability and prevent the introduction of vulnerabilities in the future.

- **Adjust the following comments to prevent potential misunderstandings.**
    - The comment in figure C.1 suggests the value returned by `getRemainingAmount` would be added to `limitStorage.remainingAmount`. However, the value returned is actually the new `limitStorage.remainingAmount`.

```
 * @dev Gets remaining amount that can be added for the window
```

Figure C.1: Comment in `RateLimit.sol`
(`paxos-token-contract-internal/contracts/lib/RateLimit.sol#59`)

-
    - The comment in figure C.2 suggests that `isFrozen` returns a list. However, it actually only allows querying the addresses that are in the list.

```
and as such we expect to happen extremely rarely. The list of frozen addresses is available
in `isFrozen(address who)`.
```

Figure C.2: Comment in `README.md`
(`paxos-token-contract-internal/README.md#103–104`)

-
    - The comment in figure C.3 suggests that `canBurnFromAddress` uses a whitelist. However, it does not.

```
 * @dev Function which checks that `burnFromAddress` is in the whitelisted map for msg.sender.
```

Figure C.3: Comment in `SupplyControl.sol`
(`paxos-token-contract-internal/contracts/SupplyControl.sol#400`)

- **Rename the `OFTProxy` contract to `OFTWrapper` or something similar (figure C.4).** Use of the term "proxy" suggests upgradeability to Ethereum developers, and is likely to cause confusion.

```
/**
 * @title OFTProxy
 * @dev This contract is a proxy around LayerZero's OFT standard.  The OFTProxy can be called
 * by LayerZero to mint and burn tokens like normal.  The OFTProxy then
```

```
forwards those requests
 12    * to the underlying token.  The underlying token must grant this contract
permission to mint and burn.
 13    */
 14    contract OFTProxy is OFTCore {
```

Figure C.4: Excerpt of `OFTProxy.sol`
(`cross-chain-contracts-internal/contracts/OFTProxy.sol#8–14`)

- **In the `mint_tokens` instruction, propagate any errors that occur (e.g., from CPI) to the caller.** Experiments suggest that a failed CPI call results in a reverted transaction. Nonetheless, propagating errors will help ensure that errors resulting from CPI calls are not intentionally ignored.

```
 80    let _ = anchor_lang::solana_program::program::invoke_signed(
 81        &ix,
 82        &[
 83            ctx.accounts.associated_token_account.to_account_info(),
 84            ctx.accounts.mint_account.to_account_info(),
 85            ctx.accounts.mint_multisig.to_account_info(),
 86            ctx.accounts.minter.to_account_info()
 87        ],
 88        signer_seeds,
 89    );
```

Figure C.5: Ignored result in the `mint_tokens` instruction
(`solana-programs-internal/programs/minter-controller/src/instructions/mint_token.rs#80–89`)

- **Correct the typo in figure C.6.** “existant” should be “existent.”

```
387    it('Cannot remove non existant whitelisted_address', async () => {
```

Figure C.6: Typo in test name (“existant” should be “existent”)
(`solana-programs-internal/tests/minter_controller.ts#387`)

- **Add an assertion to verify that an exception is thrown in the “Cannot mint if rate limit exceeded” test (figure C.7).** Currently, the test will pass if an exception is not thrown. An example where an assertion is used to verify that an exception is thrown appears in figure C.8.

```
460    try {
461      const mintTokenSignature = await minterControllerProgram.methods
462      .mintToken(capacity)
463      .accounts({
464        payer: mintAuthorityKeypair.publicKey,
465        minterAuthority: mintAuthorityKeypair.publicKey,
466        toAddress: payer.publicKey,
467        associatedTokenAccount: associatedTokenAccountAddress,
468         mintAccount: mintPDA,
469         mintMultisig: mintMultisigAddr
470       })
471       .signers([mintAuthorityKeypair])
472       .rpc();
473     } catch (err) {
474       assert.equal(err.error.errorCode.code, 'LimitExceeded')
475     }
```

Figure C.7: Excerpt of the "Cannot mint if rate limit exceeded" test. We recommend that an assertion be added between lines 472 and 473. (*`solana-programs-internal/tests/minter_controller.ts#460–475`*)

```
    await minterControllerProgram.methods
      .addMinter(capacity, refillPerSecond, adminKeypair.publicKey)
      .accounts({
        minter: mintAuthorityPDA,
        payer: provider.wallet.publicKey,
        minterAuthority: mintAuthorityKeypair.publicKey,
        mintAccount: mintPDA
      })
      .signers([mintAuthorityKeypair])
      .rpc()
      assert.fail('Expected adding duplicate mint authority to fail')
```

Figure C.8: Example where an assertion is used to verify that an exception is thrown (*`solana-programs-internal/tests/minter_controller.ts#201–211`*)

- **Write seeds and bump annotations on separate lines.** Writing them on the same line can cause the bump annotation to be overlooked. An example where the annotations are written on the same line appears in figure C.9.

```
    seeds = [b"mint-authority", minter_authority.key().as_ref(),
mint_account.key().as_ref()], bump
```

Figure C.9: Example where seeds and bump annotations are written on the same line (*`solana-programs-internal/programs/minter-controller/src/instructions/add_minter.rs#25`*)

- **In the `remove_whitelisted_address` instruction, use the whitelisted address's stored bump seed rather than recompute the bump seed (figure C.10).** The value is stored in the account (figure C.11). Thus, recomputing the bump seed is unnecessary.

```
    #[account(
       mut,
       close = payer,
       seeds = [b"mint-whitelist", minter_authority.key().as_ref(),
mint_account.key().as_ref(), to_address.key().as_ref()], bump
    )]
    pub whitelisted_address: Account<'info, WhitelistedAddress>,
```

Figure C.10: Declaration of `whitelisted_address` in `remove_whitelisted_address`'s `Context` struct (`solana-programs-internal/programs/minter-controller/src/instructions/remove_whitelisted_address.rs#35–40`)

```
#[account]
pub struct WhitelistedAddress{
    pub minter_authority: Pubkey,
    pub mint_account: Pubkey,
    pub to_address: Pubkey,
    pub bump: u8,
}
```

Figure C.11: Structure of a `WhitelistedAddress` account (`solana-programs-internal/programs/minter-controller/src/state/whitelisted_address.rs#5–11`)

- **Correct the typo in the comment in figure C.12.** "Mint" should be "Minter." Note that the same typo appears in several other places.

```
/// CHECK: Mint authority
```

Figure C.12: Comment with a typo ("Mint" should be "Minter") (`solana-programs-internal/programs/minter-controller/src/instructions/add_whitelisted_address.rs#17`)

- **Change all uses of `AccountInfo` to `UncheckedAccount`.** `UncheckedAccount` is considered the more modern convention and should be preferred. An example where both `AccountInfo` and `UncheckedAccount` are used is in figure C.13.

```
/// CHECK: Mint authority
#[account()]
pub minter_authority: UncheckedAccount<'info>,
...
/// CHECK: the wallet address to receive the token
#[account()]
pub to_address: AccountInfo<'info>,
```

Figure C.13: Example where `AccountInfo` is used. `AccountInfo` should be changed to `UncheckedAccount`. (`solana-programs-internal/programs/minter-controller/src/instructions/add_whitelisted_address.rs#17–34`)

D. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

From October 4 to October 5, 2024, Trail of Bits reviewed the fixes and mitigations implemented by the Paxos team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue. The following are the final, fixed commits for the repositories we reviewed:

- `paxos-token-contract-internal`: `a5a9df13e4df6713fb83b408daa666f5541be4a2`
- `cross-chain-contracts-internal`: `85c6b11a03ba45398eb421ef985713ba28d445e5`
- `solana-programs-internal`: `1630b535959c437c69131b2a2822b8bacd928f18`

Trail of Bits does not reevaluate code maturity as part of a fix review. As such, Paxos requested that the following comments be included in this appendix:

- Regarding Testing and Verification:

  Paxos has improved the quality and coverage of the Ethereum and Solana tests, achieving near 100% coverage. Event verification has been added to the Solana tests and integration tests have been added for the LZ Ethereum code. As for Solana integration testing, Paxos did not implement any custom code related to LZ for Solana. However, Paxos has extensively tested the configuration of the Eth and Solana pathway.

- Regarding Documentation:

  The multi-sig wallet is not documented due to Paxos security standards and processes.

In summary, of the 11 issues described in this report, Paxos has resolved all 11 issues. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Insufficient testing of cross-chain-contracts-internal contracts	Resolved
2	EnumerableSet.add return value not checked	Resolved

3	Error-prone off-chain script	Resolved
4	setEnforcedOptions.js script contains outdated code	Resolved
5	update_admin does not use a two-step transfer process	Resolved
6	Account incorrectly marked optional	Resolved
7	Unnecessary uses of mut	Resolved
8	minter_authority field is set incorrectly	Resolved
9	Tests do not check emitted events	Resolved
10	Reimplementation of library-provided function	Resolved
11	EIP3009 prohibits a frozen address from calling functions	Resolved

Detailed Fix Review Results

TOB-PAXOSLZ-1: Insufficient testing of cross-chain-contracts-internal contracts
Resolved in commit b4fadf2. The specific problem mentioned in TOB-PAXOSLZ-1 was addressed. Nonetheless, we recommend developing additional tests to verify that the Paxos contracts correctly interoperate with LayerZero.

TOB-PAXOSLZ-2: EnumerableSet.add return value not checked
Resolved in commit 1a63af8. An additional check was added to ensure no duplicates can be added to the EnumerableSet.

TOB-PAXOSLZ-3: Error-prone off-chain script
Resolved in commit a0089d6. A validation function was added that ensures that all environment variables are “set” before the script executes.

TOB-PAXOSLZ-4: setEnforcedOptions.js script contains outdated code
Resolved in commit 85c6b11. The script has been generalized so that it will work correctly if the destination is an Ethereum or Solana chain.

TOB-PAXOSLZ-5: update_admin does not use a two-step transfer process
Resolved in commit 2ca7156. The instruction update_admin was replaced with two instructions, start_admin_transfer and accept_admin_transfer. Additionally, nine tests were added to verify the new instructions’ functionality:

- Can successfully start admin transfer
- Can successfully start admin transfer with existing pending admin
- Cannot start admin transfer with None pending admin
- Cannot start admin transfer with missing admin signature
- Cannot start admin transfer with invalid admin
- Cannot accept admin transfer with missing pending admin signature
- Cannot accept admin transfer with invalid pending admin signature
- Cannot accept admin transfer with invalid pending admin
- Can successfully accept admin transfer

Note that we are unsure of the value of the “Cannot start admin transfer with None pending admin” test. The error generated during the test is:

```
TypeError: Cannot read properties of null (reading 'toBuffer')
```

Note that this is a JavaScript error, not an error generated by the Paxos contracts. While having the test certainly does not hurt, Paxos may have intended for the test to do something else.

Paxos requested that the following comment be included in this report:

> *The "Cannot start admin transfer with None pending admin" unit test has been removed in commit 58c423d370da65510f4d6b01ffaadeb203299329*

TOB-PAXOSLZ-6: Account incorrectly marked optional
Resolved in commit `ab9b81c`. The use of `Option` was removed.

TOB-PAXOSLZ-7: Unnecessary uses of mut
Resolved in commit `313e606`. All but four uses of `mut` were removed. The four remaining uses appear to be minimal. Removing any of them results in either code that does not compile, or a failing test.

TOB-PAXOSLZ-8: minter_authority field is set incorrectly
Resolved in commit `9e15c36`. The `Minter` and `WhitelistedAccount`'s `minter_authority` fields are now correctly set to the address of the `minter_authority` account. Also, the `add_whitelisted_address`, `remove_whitelisted_address`, `update_admin`, and `update_rate_limit` instructions were updated to check their `Minter` account's `minter_authority` field.

However, the `get_remaining_amount` and `mint_token` instructions were not updated like the other instructions. Furthermore, the `accept_admin_transfer` instruction added as part of the fix for TOB-PAXOSLZ-5 does not have these checks. While we consider the issue resolved, we recommend adding these checks to the `get_remaining_amount`, `mint_token`, and `accept_admin_transfer` instructions.

Paxos requested that the following comment be included in this report:

> *The additional has_one checks have been added in f51bef34b2c35f248b52a12b5f74d46a6aa4d91d*

TOB-PAXOSLZ-9: Tests do not check emitted events
Resolved in commit `0239564`. The tests were updated to check for each of the seven kinds of emitted events. In each case, we verified that the check includes each of the event's fields, and that the associated `foundEvent` variable set by the check is itself checked (see figure D.1 for an example).

```
    foundEvent = events.some((event) =>
      event.name === 'rateLimitUpdated'
      && event.data.minterAuthority.toString() === mintAuthorityKeypair.publicKey.toString()
      && event.data.mintAccount.toString() === mintPDA.toString()
      && event.data.capacity.toString() === newCapacity.toString()
      && event.data.refillPerSecond.toString() === newRefillPerSecond.toString()
    )

    let mintAuthorityPda = await minterControllerProgram.account.minter.fetch(mintAuthorityPDA)
    expect(mintAuthorityPda.rateLimit['capacity'].toString()).to.equal(newCapacity.toString())
    expect(mintAuthorityPda.rateLimit['refillPerSecond'].toString()).to.equal(newRefillPerSecond.toString())
    expect(foundEvent).to.be.true
```

Figure D.1: Example use of a `foundEvent` variable in the `RateLimitUpdated` event check (`solana-programs-internal/tests/minter_controller.ts#872–883`)

TOB-PAXOSLZ-10: Reimplementation of library-provided function

Resolved in commit `afa0a4a`. The `_getAddressSet` function was removed, and the built-in `EnumerableSet.values` function is now used in its place.

TOB-PAXOSLZ-11: EIP3009 prohibits a frozen address from calling functions

Resolved in commit `a5a9df1`. The check that prohibited the caller from being frozen has been removed. As a result, a frozen caller can now call the functions to relay pre-signed data from non-frozen accounts.

E. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.