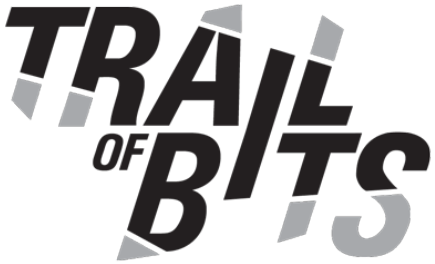

Paxos

Security Assessment

November 11, 2018

Prepared For:
Ralph Perrella | *Paxos*
rperrella@paxos.com

Prepared By:
Stefan Edwards | *Trail of Bits*
stefan.edwards@trailofbits.com

Robert Tonic | *Trail of Bits*
robert.tonic@trailofbits.com

Michael Colburn | *Trail of Bits*
michael.colburn@trailofbits.com

Changelog:
October 15, 2018: Initial report delivered
October 25, 2018: Added Appendix F
November 11, 2018: Added Appendix G

Executive Summary

Between October 1 and October 15, Trail of Bits assessed the Paxos Standard Solidity smart contracts. Three engineers conducted this assessment over the course of four person-weeks.

The assessment focused on the various smart contracts that made up the Paxos ecosystem. Special emphasis was applied to the Law Enforcement component of the Paxos ecosystem to ensure security and stability in the event of required intervention. Furthermore, guidance was applied and controls reviewed with a view towards compliance with all New York State (NYS) Department of Financial Services (DFS) New York Codes, Rules, and Regulations (NYCRR) 500 security controls.

The first week was spent on understanding the Paxos system's various components and their interactions. This entailed the review of the previous security assessment reports provided by the Paxos team, and a manual review of the Paxos token implementation.

The second week was spent finalizing manual review, recommendations, and development of automated tooling to help detect edge cases. The implemented tooling involved Echidna for property testing, and Manticore for symbolic execution.

The assessment uncovered one low-severity issue, concerning orphaned supply controller balances. Specifically, when a new supply controller is set, the previous supply controller's balance is not transferred to the new supply controller. The fix to this is detailed in the finding's recommendations in TOB-Paxos-04. An additional low-severity issue, TOB-PAXOS-06, was added to capture the state of audit logging in the contract, and recommend future directions that the development team may take in order to more easily meet NYSDFS NYCRR 500 compliance.

Informational-severity issues involved improper access controls and data exposure. The Paxos contract does not prevent accidental burn due to an implicit burn implementation through transfer to the supply controller. Law Enforcement's ability to influence on-chain representation of supply status could lead to incorrect trading values.

Engagement Goals & Scope

The goal of the engagement was to evaluate the security of the Paxos system and answer the following questions:

- Can attackers use leverage within the system to undermine the stability of the market?
- Does the design of the system introduce any risks at the architectural, code dependency, or contract level?
- Can attackers manipulate the various components of Paxos by front running transactions?
- Can the Law Enforcement or Supply Controller roles be manipulated?
- Is there any immediate concern regarding their upgradable contract usage?
- Does the application, its environment, and its security control implementations adhere to regulatory requirements?

Coverage

This review included all Solidity smart contracts used for the Paxos stable coin, found on Github at commit id f85b4a8, and the supporting upgradable contracts.

Contracts were reviewed for common Solidity flaws, such as integer overflows, re-entrancy vulnerabilities, and unprotected functions. Furthermore, contracts were reviewed with special consideration for the Law Enforcement role's potential impact on the Paxos market stability.

Additionally, the contracts were reviewed in light of the fact that Paxos itself is regulated by NYSDFS, which publishes NYCRR 500 "CYBERSECURITY REQUIREMENTS FOR FINANCIAL SERVICES COMPANIES." In furtherance of this process, Trail of Bits discussed design controls with the Paxos development and information security teams, to understand any inherited or procedural controls that may exist outside the scope of the smart contracts themselves. These discussions and review centered around audit logging, handling of key material and multi-factor authentication, and the steps necessary to comply with any regulatory guidelines of the same.

Project Dashboard

Application Summary

Name	Paxos
Type	Decentralized application
Platform	Solidity, NodeJS

Engagement Summary

Dates	October 1 - October 15, 2018
Method	Whitebox
Consultants Engaged	3
Level of Effort	4 person-weeks

Vulnerability Summary

Total High Severity Issues	0	
Total Medium Severity Issues	0	
Total Low Severity Issues	2	■■
Total Informational Severity Issues	4	■■■■
Total	6	

Category Breakdown

Access Controls	3	■■■
Data Exposure	1	■
Data Validation	1	■
Auditing and Logging	1	■
Total	6	

Recommendations Summary

Short Term

❑ **Prevent `supplyController` from being frozen or set to a frozen address.** Require the address being frozen to be different from the `supplyController` address.

❑ **Be aware that Law Enforcement can influence the supply of PAX.** Expansion and contraction of PAX should take into account any projected `lawEnforcementRole` operations.

❑ **Be aware that Law Enforcement actions are highly visible.** Assignation of the `lawEnforcementRole` could be an indicator to a malicious user to liquidate PAX.

❑ **Transfer the `supplyController` balance when migrating the role to a new address.** Modify the `setSupplyController` function to transfer the previous supply controller's balance to the new supply controller.

❑ **Prevent the `supplyController` from receiving PAX.** This reduces the likelihood of accounting errors when PAX is minted, burned or distributed by the supply controller.

❑ **Perform role address transfers in two steps.** This ensures appropriate ownership of both addresses and prevents accidental loss of control of the contract.

❑ **Consider removing 'require' conditions which operate on `SafeMath` values.** This helps to reduce gas usage and increase readability.

❑ **Monitor all reverts surrounding the Paxos contract.** This will allow Paxos defenders to determine when abnormal and potentially malicious interactions with the contract occur, and formulate an appropriate response if necessary.

Long Term

❑ **Ensure user actions are scoped appropriately.** Documentation should indicate overlapping functionality.

❑ **Consider implementing escrow functionality for wiped frozen accounts.** This allows for balances to be wiped naturally with expansion and contraction.

❑ **Consider generalizing the Law Enforcement role into an administrative role.** This helps obfuscate the intent of a newly appointed administrator if that administrator is a Law Enforcement agency.

❑ **Implement and practice incident response scenarios.** Ensure functions are implemented to quickly resolve incident response scenarios. Practice the designed disaster and incident response plans.

❑ **Implement a method to explicitly burn PAX without redistribution.** By explicitly defining a burn, restriction can occur in the transfer function to prevent accidental burning by users.

❑ **Re-architect the contract to include explicit logging event emission.** This should include successful path emissions as the contract currently does, but also failure path emissions, such that the Paxos response team may easily reconstruct an attacker's path through the application, and determine when attacks have been launched.

Findings Summary

#	Title	Type	Severity
1	lawEnforcementRole can freeze the supplyController	Access Controls	Informational
2	lawEnforcementRole can decrease supply	Access Controls	Informational
3	lawEnforcementRole operations are highly visible	Data Exposure	Informational
4	supplyController address changes result in orphaned balances	Data Validation	Low
5	Users can send PAX to supplyController	Access Controls	Informational
6	Insufficient Logging	Audit & Logging	Low

1. lawEnforcementRole can freeze the supplyController

Severity: Informational
Type: Access Controls
Target: `PAXImplementation.sol`

Difficulty: High
Finding ID: TOB-Paxos-01

Description
The `LawEnforcementRole` is able to freeze the `supplyController` balance. This makes possible expansion and contraction through `increaseSupply` and `decreaseSupply`, but the `supplyController` will no longer be able to distribute newly minted PAX using the standard `ERC-20` functions, since they all require the destination and sender to be unfrozen.

The `transfer`, `transferFrom`, and `approve` functions all check to ensure an address is not frozen. These methods are also used by the `supplyController` to distribute the minted PAX. Because there is no protection against freezing the `supplyController`, and the `supplyController` cannot perform transfers while frozen, distribution can be paused temporarily.

Exploit Scenario
Alice is assigned to be the `lawEnforcementRole`. Bob compromises Alice gaining access to her role's abilities. Bob then freezes the `supplyController`. Newly minted PAX cannot be distributed.

Recommendation
Short term, prevent `supplyController` from being frozen or set to a frozen address. This can be done through requiring the address being frozen to be different from the `supplyController`.

Long term, ensure user actions are scoped appropriately and documentation indicates overlapping functionality.

2. lawEnforcementRole can decrease supply

Severity: Informational
Type: Access Controls
Target: `PAXImplementation.sol`

Difficulty: High
Finding ID: TOB-Paxos-02

Description
When a `lawEnforcementRole` invokes `wipeFrozenAddress`, the tokens of a particular user are effectively burned.

```
    function wipeFrozenAddress(address _addr) public onlyLawEnforcementRole
{
        require(frozen[_addr], "address is not frozen");
        uint256 _balance = balances[_addr];
        balances[_addr] = 0;
        totalSupply_ = totalSupply_.sub(_balance);
        emit FrozenAddressWiped(_addr);
        emit SupplyDecreased(_addr, _balance);
        emit Transfer(_addr, address(0), _balance);
    }
```

Figure 1: The `wipeFrozenAddress` *function definition*

Due to the effect `wipeFrozenAddress` has on the `totalSupply_`, the `lawEnforcementRole` is able to control contractions through freeze functionality.

Exploit Scenario
Bob has a significant balance in two separate accounts (A and B). Account A is used in a very blatantly malicious way, enticing Law Enforcement to intervene. Account B is completely clean, with no malicious activity, and no visible association to Bob. Alice is appointed the `lawEnforcementRole`, and subsequently freezes and wipes Bob's account A balance. Due to the significant balance of Bob's account A, Alice has indirect influence over the contraction of PAX. Bob exploits this artificial contraction by selling account B's PAX when it has a higher value.

Recommendation
Short term, expansion and contraction of PAX should take into account any projected `lawEnforcementRole` operations.

Long term, consider implementing escrow functionality for frozen accounts which have been wiped. This allows for balances to be wiped naturally with expansion and contraction.

3. lawEnforcementRole operations are highly visible

Severity: Informational
Type: Data Exposure
Target: `PAXImplementation.sol`

Difficulty: Low
Finding ID: TOB-Paxos-03

Description
When a `lawEnforcementRole` is assigned, an event including the previous and new `lawEnforcementRole` address is emitted.

```
   function setLawEnforcementRole(address _newLawEnforcementRole) public {
        require(msg.sender == lawEnforcementRole || msg.sender == owner,
"only lawEnforcementRole or Owner");
        emit LawEnforcementRoleSet(lawEnforcementRole,
_newLawEnforcementRole);
        lawEnforcementRole = _newLawEnforcementRole;
    }
```

Figure 1: The `setLawEnforcement` *function definition*

Knowledge of the new `lawEnforcementRole`'s presence could lead a malicious user to preemptively liquidate PAX before a `LawEnforcementRole` is able to freeze a malicious user.

Exploit Scenario
Bob has been exploiting PAX. Alice is assigned as the `lawEnforcementRole` with the intent of freezing Bob's account. Bob observes the `lawEnforcementRoleSet` event, and preemptively liquidates his PAX before Alice can freeze his account.

Recommendation
Short term, ensure Law Enforcement is aware that assignation to the `lawEnforcementRole` could be an indicator to a malicious user to liquidate PAX.

Long term, consider generalizing the law enforcement role into an administrative role. This helps obfuscate the intent of a newly appointed administrator if that administrator is a Law Enforcement agency.

4. supplyController address changes result in orphaned balances

Severity: Low
Type: Data Validation
Target: `PAXImplementation.sol`

Difficulty: High
Finding ID: TOB-Paxos-04

Description
The `setSupplyController` function assigns a new supply controller without transferring the balance of the previous supply controller. There are currently no functions implemented to transfer PAX to the new supply controller without control of the previous supply controller.

You are only able to mint/burn tokens on the new supply controller through `increaseSupply` and `decreaseSupply`, or use the `lawEnforcementRole` to freeze and then wipe the previous supply controller.

```
   function setSupplyController(address _newSupplyController) public {
        require(msg.sender == supplyController || msg.sender == owner,
"only SupplyController or Owner");
        require(_newSupplyController != address(0), "cannot set supply
controller to address zero");
        emit SupplyControllerSet(supplyController, _newSupplyController);
        supplyController = _newSupplyController;
    }
```

Figure 1: The `setSupplyController` *function definition*

Exploit Scenario
Alice is the owner of the PAX contract. The `supplyController` address is compromised, so Alice uses `setSupplyController` to assign a new `supplyController` address. The new `supplyController` is set, but the old `supplyController` address still has a balance that is now orphaned and controlled by an attacker.

Recommendation
Short term, modify the `setSupplyController` function to transfer the previous supply controller's balance to the new supply controller.

Long term, consider the steps to disaster and incident response and ensure functions are implemented to allow these steps to be taken quickly. Practice the designed disaster and incident response plans.

5. Users can send PAX to supplyController

Severity: Informational
Type: Access Controls
Target: `PAXImplementation.sol`

Difficulty: Low
Finding ID: TOB-Paxos-05

Description
While there are checks to ensure a user cannot transfer PAX to a frozen or 0x0 address, there is no check to ensure a user cannot send PAX to the `supplyController`.

```
   function transfer(address _to, uint256 _value) public whenNotPaused
returns (bool) {
        require(_to != address(0), "cannot transfer to address zero");
        require(!frozen[_to] && !frozen[msg.sender], "address frozen");
        require(_value <= balances[msg.sender], "insufficient funds");


        balances[msg.sender] = balances[msg.sender].sub(_value);
        balances[_to] = balances[_to].add(_value);
        emit Transfer(msg.sender, _to, _value);
        return true;
    }
```

Figure 1: The `transfer` *function definition*

Exploit Scenario
Bob receives PAX from the `supplyController`. Bob subsequently transfers PAX back to the `supplyController`. The `supplyController` redistributes Bob's newly transferred PAX in the subsequent distribution, resulting in a double spend.

Recommendation
Short term, prevent the `supplyController` from receiving PAX.

Long term, implement a method to explicitly burn PAX without redistribution.

6. Insufficient Logging

Severity: Low
Type: Auditing and Logging
Target: `PaxImplementation.sol`

Difficulty: High
Finding ID: TOB-Paxos-06

Description
The contract uses `require` for halting a transaction in certain instances, such as an authentication failure. However, the contract did not emit an event; security-relevant logging information must be reconstructed from the blockchain, which is cumbersome to perform during a regulatory audit or to detect when an intrusion has occurred.

```
    function wipeFrozenAddress(address _addr) public onlyLawEnforcementRole
{
        require(frozen[_addr], "address is not frozen");
        uint256 _balance = balances[_addr];
        balances[_addr] = 0;
        totalSupply_ = totalSupply_.sub(_balance);
        emit FrozenAddressWiped(_addr);
        emit SupplyDecreased(_addr, _balance);
        emit Transfer(_addr, address(0), _balance);
    }
```

Figure 1: The `wipeFrozenAddress` *function definition*

For example, unless Paxos defensive teams are watching the blockchain for reverts, the team would be unaware that an attacker is attempting to interact with this contract, which may violate NYSDFS 500.03(n) and 500.06(a).

Exploit Scenario
An attacker is probing Paxos contracts, prior to or during a breach. As part of regulatory compliance, NYSDFS requests Paxos' intrusion detection logs, especially around Law Enforcement functionality. Paxos must then reconstruct those events from on-chain reverts, which risks missing regulatory compliance windows and results in fines and reputation loss.

Recommendation
In the short term, monitor the chain for reverts surrounding sensitive administrative and Law Enforcement functionality, as these may signal an attacker probing Paxos contracts for weaknesses.

Longer term, consider adding visible logging to all endpoints with an explicit `emit`, rather than a `require/revert`. This will allow the Paxos response team to monitor for abnormal access patterns, and assess if a breach or potential breach is about to occur.

A. Classifications

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices or software
Cryptography	Related to protecting the privacy or integrity of data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing system failure
Error Reporting	Related to the reporting of error conditions in a secure fashion
Numerics	Related to numeric calculations
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Timing	Related to race conditions, locking or order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth
Undetermined	The extent of the risk was not determined during this engagement
Low	The risk is relatively small or is not a risk the customer has indicated is important
Medium	Individual user's information is at risk, exploitation would be bad for client's reputation, moderate financial impact, possible legal implications for client
High	Large numbers of users, very bad for client's reputation, or serious legal or financial implications

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploit was not determined during this engagement
Low	Commonly exploited, public tools exist or can be scripted that exploit this flaw
Medium	Attackers must write an exploit, or need an in-depth knowledge of a complex system
High	The attacker must have privileged insider access to the system, may need to know extremely complex technical details or must discover other weaknesses in order to exploit this issue

B. Code Quality Recommendations

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

General Recommendations

- Address transfers should be performed in two steps, ensuring appropriate ownership of both addresses.
- Consider removing `require` conditions which operate on SafeMath values. This helps to reduce gas usage and increase readability.
- Ensure events for `lawEnforcement`, `owner`, and `supplyController` assignment are emitted when deploying the Paxos contract.

C. Manticore Testing

We reviewed the feasibility of proving correctness properties of the Paxos contracts with Manticore, a open-source EVM analysis tool that takes advantage of symbolic execution.

Symbolic execution explores program behavior in a broader way than classical testing methods, such as fuzzing. Trail of Bits used this to seek potential edge cases in the `SafeMath` library used by the Paxos contracts. While no edge cases were identified, the test script used has been included below for reference and usage.

To use the script, ensure Manticore is installed, then execute:

```
$ python3.6 <script_name>.py --add --sub
```

The results will then be added to an `mcore_` prefixed directory in the directory of execution.

```
import argparse
from manticore.ethereum import ManticoreEVM

def main(args):
    # If no function is specified, fail early to prevent the mcore
directory
    # from being created.
    if not (args.add or args.sub):
        print("No function specified.")
        return

    # The extracted SafeMath functions.
    contract_src="""
    contract SafeMath {

        function sub(uint256 a, uint256 b) public pure returns (uint256) {
            require(b <= a);
            uint256 c = a - b;

            return c;
        }

        function add(uint256 a, uint256 b) public pure returns (uint256){
            uint256 c = a + b;
            require(c >= a);

            return c;
        }
```

```
    }
    """

    # Define environment to perform symbolic execution
    m = ManticoreEVM()

    user_account = m.create_account(balance=1000)
    contract_account = m.solidity_create_contract(contract_src,

owner=user_account,

balance=0)

    # Define the symbolic variables to use.
    a = m.make_symbolic_value()
    b = m.make_symbolic_value()

    # Symbolically execute either/both add & sub functions.
    if args.add:
        contract_account.add(a, b)
    if args.sub:
        contract_account.sub(a, b)

    # Output states to the mcore_* directory in the current directory.
    m.finalize()

if __name__ == "__main__":
    parser = argparse.ArgumentParser()

    # Define the CLI options.
    parser.add_argument(
        "--add",
        action="store_true",
        help="""
        Perform symbolic execution against the SafeMath.add function.
        """
    )
    parser.add_argument(
        "--sub",
        action="store_true",
        help="""
        Perform symbolic execution against the SafeMath.sub function.
        """
    )

    main(parser.parse_args())
```

Figure 1: Manticore testing script for the `SafeMath` *library.*

D. Echidna Testing

Trail of Bits used Echidna, our property-based testing framework, to find security issues and logic errors in the Solidity components of Paxos. During the assessment we used Echidna to generate sequences of transactions intended to break important properties within the `PAXImplementation.sol` contract.

No issues were discovered through its use. The script below has been included as an example of the testing Trail of Bits performed against the Paxos contracts, and as a base for future property test development.

To run the tests, ensure Echidna has been installed, then run:

```
$ echidna-test <echidna_test_contract.sol> <echidna_test_contract.sol>:TEST
                        --config="<config_name>.yml"
```

```
import "./contracts/PAXImplementation.sol";

contract TEST is PAXImplementation {
    address testerAddr = 0x00a329c0648769a73afac7f9381e08fb43dbea70;
    address otherAddr = 0x67518339e369ab3d591d3569ab0a0d83b2ff5198;
    address frozenAddr = 0x57518339e369ab3d591d3569ab0a0d83b2ff5198;
    uint256 initial_totalSupply;

    function TEST() {
        unpause();

        initial_totalSupply = 6000000000 * (10**(uint256(decimals)));
        increaseSupply(initial_totalSupply);
        transfer(testerAddr, initial_totalSupply/3);
        transfer(otherAddr, initial_totalSupply/3);
        transfer(frozenAddr, initial_totalSupply/3);
        frozen[frozenAddr] = true;

        owner = address(0);
        lawEnforcementRole = address(0);
        supplyController = address(0);
    }

    // the zero address cannot have a balance
    function echidna_zero_address_balance() returns (bool) {
        return balanceOf(address(0)) == 0;
    }
```

```
    // an unprivileged user cannot increaseSupply, decreaseSupply or
    // wipeFrozenAddress (i.e. change totalSupply)
    function echidna_constant_supply() returns (bool) {
        return totalSupply() == initial_totalSupply;
    }

    // an unprivileged user cannot make themselves owner
    function echidna_become_owner() returns (bool) {
        return !(owner==testerAddr);
    }

    // an unprivileged user cannot make themselves lawEnforcementRole
    function echidna_become_LE() returns (bool) {
        return !(lawEnforcementRole==testerAddr);
    }

    // an unprivileged user cannot make themselves supplyController
    function echidna_become_SC() returns (bool) {
        return !(supplyController==testerAddr);
    }

    // a frozen user's balance remains constant
    function echidna_frozen_balance() returns (bool) {
        return balanceOf(frozenAddr) == initial_totalSupply/3;
    }
}
```

Figure 1: Echidna testing script for the Paxos `ERC-20` *compatible contract.*

```
testLimit: 10000
epochs: 1
range: 10
printCoverage: false
solcArgs: "--allow-paths ."
addrList: [0x0, 0x00a329c0648769a73afac7f9381e08fb43dbea70,
0x67518339e369ab3d591d3569ab0a0d83b2ff5198,
0x57518339e369ab3d591d3569ab0a0d83b2ff5198]
returnType: Success
```

Figure 2: Echidna configuration file to use when running the script.

E. NYSDFS Compliance

During the weekly status call, Paxos confirmed that they were audited by NYSDFS, which includes a New York Codes, Rules, and Regulations (NYCRR) policy regarding cyber security.

Trail of Bits reviewed the contract in light of these regulatory guidance controls, and discussed inherited and environmental controls with the Paxos development and information security teams. This list of controls include:

- 500.6 - Audit Trail: Finding TOB-Paxos-006 discusses methods and designs to strengthen audit controls within the contract and its surrounding ecosystem.
- 500.7 - Access Control: Trail of Bits discussed the access control policy within the contract and its surrounding ecosystem. Paxos will formalize the currently informal policy matrix with access controls, and reflect on the processes surrounding the Law Enforcement role, both within the contract as well as the web applications that internal Paxos signers will use.
- 500.12 - Multi-factor Authentication. The contract itself did not utilize multi-signature functionality to prevent a privileged attacker from exercising functionality. However, Paxos implemented key sharding outside of the contract to ensure that keys were not easily stolen, and that a single, malicious employee could not impact the system without the help of conspirators.
- 500.13 - Limitations on Data Retention: The contract did not store Nonpublic Information at the time of the assessment.
- 500.15 - Encryption of Nonpublic Information: The contract did not store Nonpublic Information at the time of the assessment.
- 500.16 - Incident Response Plan: The contract included log emission for most events that could signal an incident had occurred. However, the Paxos team will continue to monitor and improve any IR plans in place to note further when functionality has been exercised at abnormal times or by malicious actors.

F. Paxos responses to findings

Paxos accepted the risk of all findings and code quality issues identified in this report. Responses from Paxos for each outstanding issue are included as quotes below.

TOB-Paxos-01: lawEnforcementRole can freeze the supplyController

> *Operationally, Paxos executive wallets will control the owner, admin, and lawEnforcement roles. The supplyController is controlled by an Operations wallet that has strictly lower privileges than the other three roles. Although it is unlikely Paxos would ever want to freeze the supply controller, the relationship between these roles is consistent with our intent.*

TOB-Paxos-02: lawEnforcementRole can decrease supply

> *The lawEnforcementRole can wipe frozen addresses to allow for the seizure of the backing assets in the event of a legal requirement to do so. Freezing and wiping balances is something Paxos expects to be rare. The value of PAX is fixed by the backing assets, so as long as PAX and U.S. dollars are removed from the supply together, the value of the token will remain the same.*
>
> *After a freeze event, the frozen assets are held in the holding address. This avoids the need to escrow the crypto assets in a Paxos wallet, and allows for the possibility of unfreezing the address. The supply decreases done by the lawEnforcementRole are intentionally separate from the standard expansion/contraction done by the supply controller, distinguishing backing assets that are being seized by authorities from supply changes due to the purchase and redemption of PAX at Paxos.*

TOB-Paxos-03: lawEnforcementRole operations are highly visible

> *lawEnforcementRole operations are intentionally highly visible. Paxos has always been transparent about the lawEnforcementRole as part of the regulated Paxos Standard Token and does not plan to obfuscate the role in any way. To help avoid any concern about setting the lawEnforcmentRole causing a market disturbance, Paxos has set the role.*

TOB-Paxos-04: supplyController address changes result in orphaned balances

Changing the supplyController does not also move the supplyController's PAX balance. It also does not move the Eth balance needed to pay for transactions. This is not an issue for the Paxos Standard since the stablecoin is fully backed, which operationally means that supply changes happen in response to purchases and redemptions of PAX. The purchases have an immediate destination, so there is no reason for the supplyController to carry a standing balance.

TOB-Paxos-05: Users can send PAX to supplyController

Operationally, Paxos will not simply burn tokens it does not know the intent for, so tokens will not be unintentionally burned. However, in general Paxos cannot stop token holders from sending their tokens to addresses that they do not control, which in effect leads to a loss in the circulating supply of PAX. The fully-backed model of the stablecoin ensures that this does not pose a threat to the stability of the token.

TOB-Paxos-06: Insufficient Logging

Ethereum smart contract events are a standard way to track events on the blockchain. However, Paxos has kept the focus of contract events primarily to ones that might be of interest to the token holders and users. Operationally, Paxos reads the chain directly to confirm balances for redemptions, and similarly, events related to exploits are monitored on the chain directly.

Code Quality Recommendations

- Address transfers should be performed in two steps, ensuring appropriate ownership of both addresses.

  Operationally, Paxos validates ownership of target addresses by signing and broadcasting a test transaction.

- Consider removing require conditions which operate on SafeMath values. This helps to reduce gas usage and increase readability.

  The require statement checks use very little gas (30 or less, which is less than 0.1% of the gas cost for a transaction on the smart contract). Because the smart contracts are

already deployed, Paxos has decided not to publish an update via an upgrade of the implementation contract to address this recommendation at this time.

- Ensure events for lawEnforcement, owner, and supplyController assignment are emitted when deploying the Paxos contract.

  Since the contract is already deployed, changing the deployment events will not change the history of the deployment events and therefore the contract code has been left unchanged.

G. ERC20 approve race condition

Trail of Bits noted that the Paxos token is exposed to the well-known ERC20 race condition during this engagement. This issue was discussed with Paxos but was not fully investigated due to time constraints. Trail of Bits continued their investigation after the engagement concluded and identified scenarios with possible security impacts to Paxos. This appendix reviews the issue, describes the impact, and recommends mitigations.

Issue Description
Paxos conforms to the ERC20 token standard, which contains an unavoidable race condition. Paxos's compliance with ERC20 inherently introduces this race condition. This race condition is only exploitable by sophisticated attackers, but could result in loss of funds for Paxos users.

It is not a smart contract correctness bug, but rather a consequence of the API design and Ethereum's unique execution model. The bug is quite subtle and difficult to understand. Normally, people think of the transaction model as completely separate from the code it executes, but this bug requires a nuanced understanding of their interaction to precisely understand its impact.

Specifically, the ERC20 standard requires two functions, `approve` and `transferFrom`, which allow users to designate other trusted parties to spend funds on their behalf. Calls to any Ethereum function, including these, are visible to third parties prior to confirmation on-chain. In addition to these calls' visibility prior to confirmation, a sophisticated attacker can "front-run" them and insert their own transactions to occur *before* the observed calls.

The `approve` function is defined to take an address and an amount, and set that address's "allowance" to the specified amount. Then, that address can call `transferFrom` and move up to their allowance of tokens as if they were the owners. Here's the issue: `approve` is specified to be idempotent. It sets the approval to a new value regardless of its prior value, it doesn't modify the allowance.

Exploit Scenario
In a scenario where a malicious party is approved for some amount and then the approving party wants to update the amount, the malicious party could end up with significantly more funds than the approving party intended.

Suppose Alice, a non-malicious user, has previously approved Bob, a malicious actor, for $100 for Paxos. She wishes to increase his approval to $150. Bob observes the `approve(bob, 100)` transaction prior to its confirmation and front-runs it with a

`transferFrom(alice, bob, 100)`. Then, as soon as the new approval is in, his allowance is set to $150 and he can call `transferFrom(alice, bob, 150)`.

In this scenario, Alice believes she's setting Bob's allowance to $150, and he can only spend 150 tokens. Due to the race condition, Bob can spend $250. This is effectively a theft of tokens. Bob can then use these stolen tokens at an exchange that accepts Paxos. Even if Paxos directly modifies balances to refund Alice her tokens, the participating exchange is left with the liability since Bob has already traded the Paxos for another cryptocurrency.

Likelihood of Exploitation
As mentioned above, only sophisticated attackers can exploit this bug, and only in very specific circumstances. The attack requires dedicated infrastructure to monitor and quickly react to transactions. Performing it consistently may require collaboration with an Ethereum mining pool. In addition, it is only possible when the attacker has already been approved for some allowance. Even then, the value an attacker can steal is limited (it cannot be more than the initial allowance).

Due to the degree of effort required, the minimal reward, and the unlikely circumstances required (e.g., the vast majority of ERC20 token holders never use `approve` and `transferFrom` in the first place, let alone with untrusted parties), Trail of Bits is unaware of this bug ever having been exploited in the wild. It simply has not proven profitable to exploit. It has been widely known for quite some time now and most large tokens elect to remain standards-compliant rather than mitigate it.

Nonetheless, any issue that could result in loss of funds as well as loss of confidence in Paxos is very important, and must be addressed seriously and comprehensively to the extent possible. Just because it has not been exploited in the past does not mean it never will be in the future. As attackers grow more serious and well-resourced, bugs this subtle and difficult to exploit merit thorough consideration.

Available Mitigations
This vulnerability can be largely mitigated. Tokens can add `increaseApproval` and `decreaseApproval` functions, which are not idempotent and, therefore, do not suffer the above issue. Users who exclusively use these functions will not be vulnerable.

Alternatively, users can ensure that when they update an allowance, they either set it to 0 or verify that it was 0 prior to the update. In the above example, the user would call `approve(0)` then `approve(150)` instead of just the latter.

Notably, both outlined mitigations require users to use the API with some care. The solution is not just modifying code, but creating and publishing documentation. Since this issue is in the standard and Trail of Bits cannot recommend that Paxos remove it entirely

(by modifying the approve function), it is critical that users of this functionality are informed of the risk and understand the best practices for avoiding it.

Paxos response to ERC20 race condition

> *Recommendations on careful use of these ERC20 methods have been added to our Readme where we already mention the ERC20 approve/transferFrom flow. In practice users can be advised to set the approval to zero (and confirm that change was not front-run) before setting the to another amount.*
>
> *We plan to maintain full compatibility with the ERC20 specification, meaning the second workaround is out-of-bounds. As long as we continue to support ERC20 the root issue is communicating with users that increasing approval in one step using the ERC20 methods can only be done with trusted parties.*
>
> *We were aware of this issue and did originally consider the increaseApproval and decreaseApproval methods from open zeppelin (as they were called at the time). However, we determined that they were not strictly necessary, and we were trying to minimize the contract code as much as possible, so we left it out. We also understand that there has to date not been a known successful front-running attack on this vulnerability. Weighing this against the security risk posed by a smart contract upgrade Paxos has chosen to leave the smart contract implementation code as it currently stands, and to rely on communicating the best practices to our users.*