Brickblock [Phase 2] Audit

1 Summary

ConsenSys Diligence conducted a security audit on Brickblock's system of smart contracts for tokenizing real-world assets with a specific focus on real estate. The scope of the audit included Brickblock's upgradable system of smart contracts, encompassing three tokens, a pricing oracle, and other utilities, but with the understanding that one of the contracts, POAManager, was not frozen and would undergo further development. The objective of the audit was to discover issues that could threaten the funds held in or behaviour of the Brickblock system, including its future upgradability.

1.1 Audit Dashboard

Audit Details

- **Project Name:** Brickblock Audit
- **Client Name:** Brickblock
- **Client Contact:** Philip Paetz, Cody Lamson
- **Auditors:** Gonçalo Sá, Sarah Friend
- **GitHub :** https://github.com/brickblock-io/smart-contracts
- **Languages:** Solidity, Solidity Assembly, JavaScript
- **Date:** 8th June -

Number of issues per severity

Minor	Medium	Major	Critical
25	**4**	**0**	**0**

1.2 Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient and working according to its specifications. The audit activities can be grouped in the following three categories:

Security: Identifying security related issues within each contract and within the system of contracts.

Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:

- Correctness
- Readability
- Sections of code with high complexity
- Improving scalability
- Quantity and quality of test coverage

1.3 System Overview

Documentation

The following documentation was available to the audit team:

- The README which describes how to work with the contracts.
- The Ecosystem documentation gives an architectural overview and detailed information about the individual contracts.
- The Tests against Geth doc which explains how to run the tests against `geth` and not truffle's `ganache`.

Scope

The audit focus was on the smart contract files, and test suites found in the following repositories:

Repository	Commit hash	Commit date
brickblock-io/smart-contracts	f1f5b04722b9569e1d4c0b62ac4c490c0a785fd8	8th June 2018

The full list of smart contracts in scope of the audit can be found in chapter Appendix 1 - File Hashes.

Design

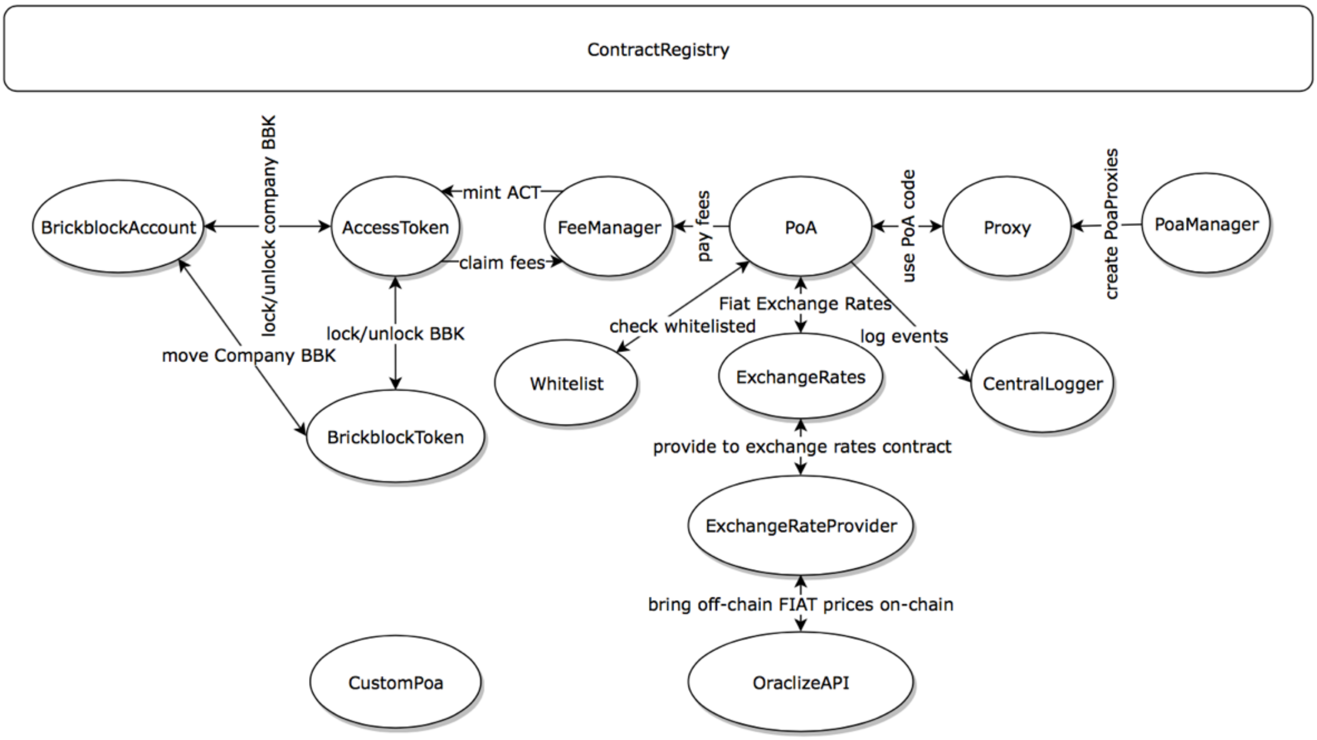
Brickblock is a system for managing the tokenization of assets, as well as custodianship/brokerage of and investment in those assets. The most important concepts of the Brickblock system are listed below:

- **Registrar**: is at the core of the system. It's an `ownable` contract that uses unstructured storage to manage an upgradeable directory of component contracts.
- **POA Tokens**: represents an asset that has been tokenized. It's called via a multitude of POAProxies that are deployed by `POAManager` via the delegate proxy factory pattern.
- **Exchange Price Oracle**: `ExchangeRateProvider` inherits from the Oraclize API and fetches current exchange rates, storing them in the `ExchangeRates` contract for use.
- **BBK Token**: is a double-entry paradigm token that can be locked for a period of time to collect ACT rewards.
- **ACT Token**: is another double-entry paradigm token that serves as a payout to locked-in BBK tokens and can be exchanged at a set-rate for Ether.

To better understand how all the components interact it is helpful to analyze the system diagram (source ecosystem):

1.4 Key Observations/Recommendations

Praises:



- The **system specification was thorough** from the day this phase of the audit was initiated and every design choice was well-founded.
- The Brickblock **team was interactive throughout and diligent in applying fixes** to presented issues.

Recommendations:

- **Last pass on specification**: it is recommended that the team does one last pass on the specification and documentation of the codebase. This includes comments in the codebase, as some of these have proved to be inconsistent with current code state.
- **Last pass on implementation**: akin to the last pass on specification/documentation it is recommended that stale parts of the codebase are identified and removed before deployment to mainnet.
- **Fix all issues:** It is recommended to fix all the issues listed in the below chapters, at the very least the ones with severity Critical, Major and Medium.

1.5 Revision

This section serves the sole purpose of acknowledging that the auditing team has approved all the changes coming into effect as a cause of the first revision of the report.

The team acknowledges that all issues have been closed either by changing the codebase to correctly address the problem at hand or by having the development team provide a precise explanation of why said issue was not addressed.

Remediation links are provided in each of the relevant issue sections. Non-fix explanations are found in this same section in the following table.

The commit hash agreed upon for checkpointing the codebase after all the fixes was: `99770100c9ae5ab7c8ac9d79e3fd0a8bce5f30b7`.

Non-addressed Issues Explanation

Issue Number	Explanation
3.4	Keep consistency of other calls where the event logger is used.
3.8	The power held by the owner of the system already enables other attacks.
3.15	Referenced code is just a stub for testing and so doesn't affect normal system operations.
3.16	The current design allows for great flexibility as well as keeping fee payments simple. There have been no issues found with this so far.
3.20	Said overflows will not happen for a **very large** period of time.
3.21	Said overflows will not happen for a **very large** period of time.
3.22	Due to the way we want to present balances to users during the crowdsale aspect, we want to ensure that the balance shows 0 for all users until a specific stage. There does not seem to be an easier way to do this. Additionally, the extra gas cost is not much.
3.22	The struct is the same shape for both PoaToken and Broker data. The rights access is controlled with modifiers on the public functions (addBroker, removeBroker, addToken, removeToken) and then make use of private functions to work with the abstract EntityState.

2 Issue Overview

The following table contains all the issues discovered during the audit. The issues are ordered based on their severity. A more detailed description of the levels of severity can be found in Appendix 2. The table also contains the Github status of any discovered issue.

Chapter	Issue Title	Issue Status	Severity	Opt.
3.1	Unnecessary complexity in `toXLengthString` functions in `PoaCommon`	Closed	Medium	✓
3.2	No plan for how a physical tokenized asset would handle a chain split	Closed	Medium	
3.3	Usage of random storage slots in the Proxy adds too much complexity	Closed	Medium	
3.4	Unnecessary usage of low-level `.call()` method	Closed	Medium	
3.5	Withdraw method does not check if balance is sufficient for the withdrawal	Closed	Minor	
3.6	Can lock and unlock 0 BBK in AccessToken	Closed	Minor	
3.7	Precision in percent function can overflow	Closed	Minor	
3.8	Transaction order dependence issue in `ExchangeRates`	Closed	Minor	

Chapter	Issue Title	Issue Status	Severity	Opt.
3.9	Non-optimal ordering of instructions in `PoaProxy` and `PoaToken` fallback functions	Closed	Minor	✓
3.10	`ExchangeRateProvider`'s callback check for access control is non-optimal	Closed	Minor	
3.11	Inaccurate specification comment for `setFailed()` method in `PoaCrowdsale`	Closed	Minor	
3.12	Unnecessary fallback functions to refuse payments	Closed	Minor	✓
3.13	Comment about upgrade path is incorrect	Closed	Minor	
3.14	buyAndEndFunding ends by calling buyAndContinueFunding	Closed	Minor	
3.15	Unused variable has no dummy check-in `ExchangeRateProviderStub`	Closed	Minor	
3.16	FeeManager open-by-default design might introduce flaws in the token economy	Closed	Minor	
3.17	Unnecessary refund action in `PoaCrowdsale`	Closed	Minor	✓
3.18	`this` should be explicitly typecast to `address`	Closed	Minor	
3.19	Blocking conditions in buyFiat	Closed	Minor	
3.20	Use of ever-growing unsigned integers in `PoaToken` is dangerous	Closed	Minor	
3.21	Use of ever-growing unsigned integers in `AccessToken` is dangerous	Closed	Minor	
3.22	Non-optimal stage checking condition in `PoaToken`	Closed	Minor	
3.23	Contradicting comment on POAManager	Closed	Minor	
3.24	Inconsistent type used for decimals	Closed	Minor	
3.25	Inconsistent event naming	Closed	Minor	
3.26	Incorrect name of parameter in BBKUnlockedEvent	Closed	Minor	
3.27	Usage of `EntityState` for both brokers and tokens in `PoaManager` is an anti-separation-of-concerns pattern	Closed	Minor	

3 Issue Detail

3.1 Unnecessary complexity in `toXLengthString` functions in `PoaCommon`

Severity	Issue Status	GitHub Repo	Issue Link
Medium	Closed	brickblock-audit-report-2	issues/37

Description

Both the `toXLengthString` functions in `PoaCommon` are too complex and can be substituted by a simpler version with a single assembly block.

Remediation

```
    function to32LengthStringOpt(
        bytes32 _data
    )
        pure
        internal
        returns (string)
    {
        // create new empty bytes array with same length as input
        bytes memory _bytesString = new bytes(32);

        // an assembly block is necessary to change memory layout directly
        assembly {
            // we store the _data bytes32 contents after the first 32
bytes of
            // _bytesString which hold its length
            mstore(add(_bytesString, 0x20), _data)
        }

        // and now we measure the string by searching for the first
occurrence
        // of a zero'ed out byte
        for (uint256 _bytesCounter = 0; _bytesCounter < 32;
_bytesCounter++) {
            if (_bytesString[_bytesCounter] == hex"00") {
                break;
            }
        }

        // knowing the trimmed size we can now change its length directly
        assembly {
            // by changing the 32-byte-long slot we skipped over
previously
            mstore(_bytesString, _bytesCounter)
        }

        return string(_bytesString);
    }
    function to64LengthStringOpt(
        bytes32[2] _data
    )
        pure
        internal
        returns (string)
    {
        // create new empty bytes array with same length as input
        bytes memory _bytesString = new bytes(64);

        // an assembly block is necessary to change memory layout directly
        assembly {
            // we store the _data bytes32 contents after the first 32
bytes of
            // _bytesString which hold its length
            mstore(add(_bytesString, 0x20), mload(_data))
            mstore(add(_bytesString, 0x40), mload(add(_data, 0x20)))
        }

        // and now we measure the string by searching for the first
occurrence
        // of a zero'ed out byte
        for (uint256 _bytesCounter = 0; _bytesCounter < 64;
_bytesCounter++) {
            if (_bytesString[_bytesCounter] == hex"00") {
                break;
            }
        }

        // knowing the trimmed size we can now change its length directly
        assembly {
            // by changing the 32-byte-long slot we skipped over
previously
            mstore(_bytesString, _bytesCounter)
        }

        return string(_bytesString);
    }
```

3.2 No plan for how a physical tokenized asset would handle a chain split

Severity	Issue Status	GitHub Repo	Issue Link
Medium	Closed	brickblock-audit-report-2	issues/48

Description

The brickblock contract system creates tokens for physical assets, but in the event of an unplanned contentious hard fork, there would be two blockchain assets for each physical one. This is a potentially catastrophic scenario.

Remediation

Plan possible scenarios for how the brickblock system would handle the split tokens, choose a fork to support, and/or deprecate a fork. Add the plans to `WORST-CASE-SCENARIOS.md`

3.3 Usage of random storage slots in the Proxy adds too much complexity

Severity	Issue Status	GitHub Repo	Issue Link
Medium	Closed	brickblock-audit-report-2	issues/21

Description

There is a big complexity in the codebase stemming from the use of a custom implementation of randomized storage slots for system-wide storage variables. This promotes dense code and may introduce unknown vulnerabilities.

Remediation

The set of PoA-related contracts could make use inherited storage instead of having addresses reside in random slots in storage. This would avoid such heavy use of inline assembly, therefore, maintaining readability and safety.

3.4 Unnecessary usage of low-level `.call()` method

Severity	Issue Status	GitHub Repo	Issue Link
Medium	Closed	brickblock-audit-report-2	issues/40

Description

Throughout the set of PoA-related contracts, there is an unnecessary and possibly dangerous usage of the low-level `.call()` method since every contract being called is known by the caller beforehand.

Remediation

Typecast the `address` variable returned by `ContractRegistry` and call the relevant member of the contract type without the use of `.call()` (this is especially relevant in https://github.com/brickblock-io/smart-contracts/blob/6360f5e1ba0630fa0caf82ff9b58b2dc5e9e1b53/contracts/PoaCommon.sol#L184).

3.5 Withdraw method does not check if balance is sufficient for the withdrawal

Severity	Issue Status	GitHub Repo	Issue Link
Minor	Closed	brickblock-audit-report-2	issues/29

Description

The withdrawEthFunds in BrickblockAccount does not check that balance is greater than the amount being requested, just that it's greater than zero

```
function withdrawEthFunds(
    address _address,
    uint256 _value
  )
    external
    onlyOwner
    returns (bool)
  {
    require(address(this).balance > 0);
    _address.transfer(_value);
    return true;
  }
```

Remediation

Consider switching `require(address(this).balance > 0);` to `require(address(this).balance >= _value);`

3.6 Can lock and unlock 0 BBK in AccessToken

Severity	Issue Status	GitHub Repo	Issue Link
Minor	**Closed**	brickblock-audit-report-2	issues/30

Description

This method is public and can be called by anyone with quantity zero

Remediation

Consider adding a validator to the function to eliminate a possible source of user error

`require( _value > 0);`

3.7 Precision in percent function can overflow

Severity	Issue Status	GitHub Repo	Issue Link
Minor	**Closed**	brickblock-audit-report-2	issues/46

Description

The public percent function in `PoaCrowdsale` takes precision as a parameter, does not validate it, and does not use safe math

`uint256 _safeNumerator = _numerator.mul(10 ** (_precision + 1));`

Remediation

Though the only place the brickblock contract system currently uses this function, precision is set at 18, using safe math here could prevent future error as the contract system evolves.

3.8 Transaction order dependence issue in `ExchangeRates`

Severity	Issue Status	GitHub Repo	Issue Link
Minor	**Closed**	brickblock-audit-report-2	issues/47

Description

Even though there is an access control layer applied to the whole contract, there's a transaction order dependence issue with the "owner" agent in `ExchangeRates`. When seeing a big buy transaction come in, "owner", basically controlling the exchange rate, could prepend a transaction (or multiple ones) of his own to get all the contribution for, practically, no tokens in exchange.

Remediation

A timelock could be implemented to give buyers a safe window on which to execute buy orders, but since the "owner" already holds so much power in the ACL structure, this may not be needed for the end user to feel safe buying tokens.

3.9 Non-optimal ordering of instructions in `PoaProxy` and `PoaToken` fallback functions

Severity	Issue Status	GitHub Repo	Issue Link
Minor	**Closed**	brickblock-audit-report-2	issues/45

Description

In `PoaProxy` and `PoaToken` fallback functions, the order of the instructions can be changed to achieve better gas optimization. There is no need to copy return data to memory if the call result is `false` and the call is going to be reverted anyway.

Remediation

Have the `iszero(result)` condition check reside before the `returndatacopy` instruction.

3.10 `ExchangeRateProvider`'s callback check for access control is non-optimal

Severity	Issue Status	GitHub Repo	Issue Link
Minor	**Closed**	brickblock-audit-report-2	issues/44

Description

Going against proposed COP (condition-oriented programming) patterns and the general code style present throughout the codebase, the `__callback` method of `ExchangeRateProvider` (v.

https://github.com/brickblock-io/smart-contracts/blob/6360f5e1ba0630fa0caf82ff9b58b2dc5e9e1b53/contracts/ExchangeRateProvider.sol#L100) does not use a modifier to check if the caller is authorized to run this function.

Remediation

Have this check: `require(msg.sender == oraclize_cbAddress());` reside in a properly named *onlyX* modifier.

3.11 Inaccurate specification comment for `setFailed()` method in `PoaCrowdsale`

Severity	Issue Status	GitHub Repo	Issue Link
Minor	**Closed**	brickblock-audit-report-2	issues/41

Description

The specification comment above the `setFailed()` method mentions scenarios that don't need this function to get to the "Failed" stage.

3.12 Unnecessary fallback functions to refuse payments

Severity	Issue Status	GitHub Repo	Issue Link
Minor	**Closed**	brickblock-audit-report-2	issues/42

Description

In `AccessToken`, `CentralLogger`, `ContractRegistry`, `ExchangeRates`, `FeeManager`, `PoaManager` and `Whitelist` the presence of the fallback function there defined is not needed because the default Solidity behavior is to disallow payments to contracts through their fallback function.

Remediation

Remove the fallback function definition from these contracts.

3.13 Comment about upgrade path is incorrect

Severity	Issue Status	GitHub Repo	Issue Link
Minor	**Closed**	brickblock-audit-report-2	issues/35

Description

This comment in AccessTokenUpgradeExample is incorrect. In the event of an upgrade, more than just inheritance will be required to access the state of the old contract.

```
* This is an example of how we would upgrade the AccessToken contract if
we had to.
```

```
 * Instead of doing a full data migration from ACTv1 to ACTv2 we could
make
 * use of inheritance and just access the state on the old contract.
```

Remediation

Remove the comment to prevent a source of possible future confusion.

3.14 buyAndEndFunding ends by calling buyAndContinueFunding

Severity	Issue Status	GitHub Repo	Issue Link
Minor	Closed	brickblock-audit-report-2	issues/39

Description

The function that ends a `PoaCrowdsale`, `buyAndEndFunding`, ends by calling `buyAndContinueFunding` - though there is no wrong functionality here, it is counterintuitive.

Remediation

Since `buyAndContinueFunding` has more than one use, consider renaming it - it provides no guarantees that funding continues.

3.15 Unused variable has no dummy check-in `ExchangeRateProviderStub`

Severity	Issue Status	GitHub Repo	Issue Link
Minor	Closed	brickblock-audit-report-2	issues/28

Description

There are unused variables in the `sendQuery` function in ExchangeRateProvider, generating a compiler warning. In ExchangeRateProviderStub on the same function, there's a comment about doing a dummy check is wrong, but no dummy check is done.

Remediation

Silence the compiler by mentioning the variables `_callInterval, _callbackGasLimit`

3.16 FeeManager open-by-default design might introduce flaws in the token economy

Severity	Issue Status	GitHub Repo	Issue Link
Minor	Closed	brickblock-audit-report-2	issues/18

Description

The `payFee` function in `FeeManager` is public and does not validate or restrict `msg.sender`

Remediation

While this is intentional, it also increases the attack surface of the system, since paying a fee to FeeManager effects the `totalSupply_` of ACT. Though at the moment any attack is likely prohibitively expensive, economic interference with the exchange rates of BBK to ACT is possible.

3.17 Unnecessary refund action in `PoaCrowdsale`

Severity	Issue Status	GitHub Repo	Issue Link
Minor	**Closed**	brickblock-audit-report-2	issues/43

Description

In the `buyAndEndFunding()` method of `PoaCrowdsale` there's a `transfer` action being executed every time even if the refund is equal to *0* or not even requested/needed.

Remediation

Only `transfer` if `refundAmount > 0`.

3.18 `this` should be explicitly typecast to `address`

Severity	Issue Status	GitHub Repo	Issue Link
Minor	**Closed**	brickblock-audit-report-2	issues/19

Description

`this` is implicitly used as an address, which is forbidden in newer versions of solidity

Remediation

Every instance of `this` should now be explicitly typecast to the `address` type

3.19 Blocking conditions in buyFiat

Severity	Issue Status	GitHub Repo	Issue Link
Minor	**Closed**	brickblock-audit-report-2	issues/38

Description

There is an edge case where the difference between `fundingGoalInCents` and fundedAmountInCentsDuringFiatFunding is less than 100, causing this earlier check `require(_amountInCents >= 100);` to block reaching the funding goal

In addition, there is a logical error in the function: Because of the check `if (fundingGoalInCents().sub(_newFundedAmount) >= 0)`, the second check `if` `(fundedAmountInCentsDuringFiatFunding() >= fundingGoalInCents())` can never be greater than, only less than or equal to.

Remediation

Though this can be unblocked by moving on to the third stage and funding with Ether, the gas fees to do so will likely be more than the remaining needed funding amount. Possible mitigations include removing the `require(_amountInCents >= 100);`, validating that `fundingGoalInCents % 100 == 0`, or otherwise changing the logical flow.

3.20 Use of ever-growing unsigned integers in `PoaToken` is dangerous

Severity	Issue Status	GitHub Repo	Issue Link
Minor	**Closed**	brickblock-audit-report-2	issues/36

Description

Just like `AccessToken`, this contract makes use of unsigned integer variables that can only increase and create an attack surface for DoS attacks, as well as a scalability limitation.

Remediation

Even though from a very careful analysis we could see that any attack would be hugely costly this presents an opportunity for a possible extension over this token, in the future, to overlook this nature of said variables and for this to become an actual attack vector.

Similarly to `AccessToken`, the results of balanceOf calls could be validated

3.21 Use of ever-growing unsigned integers in `AccessToken` is dangerous

Severity	Issue Status	GitHub Repo	Issue Link
Minor	**Closed**	brickblock-audit-report-2	issues/33

Description

In both the `balanceOf` and `distribute` functions the math behind makes use of `uint256` variables that are ever-growing (can only increase and never decrease, per specification), this creates an attack surface for DoS attacks.

Remediation

Even though from a very careful analysis we could see that any attack would be hugely costly this presents an opportunity for a possible extension over this token, in the future, to overlook this nature of said variables and for this to become an actual attack vector.

The possibility of attack or accidental DOS can be prevented by using the results of balanceOf function in an overflow check

```
uint256 newRecipientBalance = balanceOf(_to).add(_value);
uint256 tempSpent = spentBalances[_to];
require(tempSpent.add(newRecipientBalance));
```

3.22 Non-optimal stage checking condition in `PoaToken`

Severity	Issue Status	GitHub Repo	Issue Link
Minor	Closed	brickblock-audit-report-2	issues/34

Description

The check of whether PoaToken is in stage 4 is implemented in the `startingBalance` function which, in turn, is used in the `balanceOf` function which is transversal to a lot of other functions.

Besides creating an extra piece of bytecode that will get executed even in the `transferFrom` and `currentPayout` functions, it is buried down in the logic which makes it harder to assess the certainty of the specification: "the token is only tradeable after stage 4".

Remediation

The use of a modifier on the `transfer` function alone would achieve the same effect and produce more readable and extensible code.

3.23 Contradicting comment on POAManager

Severity	Issue Status	GitHub Repo	Issue Link
Minor	Closed	brickblock-audit-report-2	issues/24

Description

The `addToken()` function in POAManager has a comment saying it initializes the entity with `_active` as true but actually sets it false.

Remediation

Verify that this is the correct behaviour in code, and correct the comment

3.24 Inconsistent type used for decimals

Severity	Issue Status	GitHub Repo	Issue Link
Minor	Closed	brickblock-audit-report-2	issues/25

Description

An inconsistent type is used for decimals. In POAToken uint256 is used, in AccessToken uint8 is used.

Remediation

Consider which type is preferable for this parameter and use it uniformly throughout all tokens. `uint8` is more commonly seen in standards

3.25 Inconsistent event naming

Severity	Issue Status	GitHub Repo	Issue Link
Minor	Closed	brickblock-audit-report-2	issues/26

Description

Throughout the contract system - somewhat inconsistent event naming conventions, for example, Burn and BurnEvent

```
event BurnEvent(address indexed burner, uint256 value);

event Burn(address indexed burner, uint256 value);
```

Remediation

Decide on a naming convention and use it throughout the system. The `BurnEvent` pattern may be the stronger choice, as it follows the `Differentiate functions and events` best practice

3.26 Incorrect name of parameter in BBKUnlockedEvent

Severity	Issue Status	GitHub Repo	Issue Link
Minor	Closed	brickblock-audit-report-2	issues/27

Description

In AccessToken, wrong variable name, the second uint256 is actually the unlockedAmount

```
event BBKUnlockedEvent(
  address indexed locker,
  uint256 lockedAmount,
  uint256 totalLockedAmount
);
```

Remediation

Correct the variable name:

```
    event BBKUnlockedEvent(
      address indexed locker,
      uint256 unlockedAmount,
      uint256 totalLockedAmount
    );
```

3.27 Usage of `EntityState` for both brokers and tokens in `PoaManager` is an anti-separation-of-concerns pattern

Severity	Issue Status	GitHub Repo	Issue Link
Minor	**Closed**	brickblock-audit-report-2	issues/32

Description

The use of the `doesEntityExist` modifier and the `addEntity`, `removeEntity`, and `setEntityActiveValue` to manipulate both brokers and tokens in the contract's state is an anti-pattern regarding separation of concerns.

Since these functions are reused across two very different domains of logic state, this means that in the unlikely event of a public function related to brokers having a vulnerability there's a non-zero probability that tokens are compromised as well. Given the importance of the prior and latter lists, this is a clear escalation in the severity of a vulnerability.

Remediation

Create specific functions to handle each one of the different entities (e.g. `addToken`, `removeBroker`) or implement the ***add***, ***remove*** and ***set active*** logics for each entity in the public functions themselves instead of having shared private functions for that.

4 Tool based analysis

The issues from the tool based analysis have been reviewed and the relevant issues have been listed in chapter 3 - Issues.

4.1 Mythril

Mythril is a security analysis tool for Ethereum smart contracts. It uses concolic analysis to detect various types of issues. The tool was used for automated vulnerability discovery for all audited contracts and libraries. More details on Mythril's current vulnerability coverage can be found here.

The raw output of the Mythril vulnerability scan can be found here. It was thoroughly reviewed for possible vulnerabilities, and all the results stemming out of such analysis were included in the final issues report.

4.2 Sūrya

Surya is a utility tool for smart contract systems. It provides a number of visual outputs and information about the structure of smart contracts. It also supports querying the function call graph in multiple ways to aid in the manual inspection and control flow analysis of contracts.

A complete list of functions with their visibility and modifiers can be found here.

4.3 Odyssey

Odyssey is an audit tool that acts as the glue between developers, auditors, and tools. It leverages Github as the platform for building software and aligns to the approach that quality needs to be addressed as early as possible in the development life cycle and small iterative security activities spread out through development help to produce a more secure smart contract system. In its current version Odyssey helps communicate audit issues to development teams better and to close them successfully.

Appendix 1 - File Hashes

The SHA1 hashes of the source code files in scope of the audit are listed in the table below.

Contract File Name	SHA1 hash
stubs/RemoteContractStub.sol	c2da2c57d0502a68acc9cafa134ffb62dfdc8446
stubs/RemoteContractUserStub.sol	b4d9811cca3c8c2516d521f315945f18e1ca488c
stubs/ExchangeRateProviderStub.sol	bce06f04ad4ae358e2802198484a95d7091cbdfb
stubs/BrokenRemoteContractStub.sol	76d0cd9bcb809cd26255fcbf0aca5aae593fdd13
stubs/PoaManagerStub.sol	886dd9d3f890acf7f6cf3c802a02e28dfcb38795
stubs/UpgradedPoa.sol	7ddde558f506efec77488ba958fc1db714d1df4d
stubs/BrickblockFountainStub.sol	1fcd2643e33cf0fa76644dd2203b0fa697701ed5
PoaProxy.sol	2359f57c3503608f206195372e220d3673a127f2
PoaManager.sol	0022d2a65065359ef648d05fc1a01b049dd32ff3
ExchangeRates.sol	dd4c7a19d798a5a097d12e7fd2146f18705d5e6c
tools/WarpTool.sol	c2e2f5b46c2382d5919a6a11852d8bd3718ea238
CustomPOAToken.sol	bc8a19f076450c44a8c1cb175626e9ca5b21c712
OraclizeAPI.sol	974d293678647f934864c4eef21469c322e60f19
CentralLogger.sol	63d7facdd2fd969f798b7eef4f3eb89392f817ea
FeeManager.sol	ba1fa0085716b524424a8b1ba366fde272b03842
BrickblockAccount.sol	2c8cf3c8a6c8ce68044c89afaa1b30e5392f1b0c
AccessToken.sol	9ea080dade42bf75787805d87be7aa7d3cdf2f11
Migrations.sol	cfc2c3229aa8d50eb038dbdad89b79c10aa76e81

Contract File Name	SHA1 hash
Whitelist.sol	0059355f7b70aefcae1e00293717c5547bf4c9f2
BrickblockToken.sol	1dc072c4a388eb02a8e5ff94e53170266b3986cd
PoaToken.sol	7115dd663666c65344d60530cb7f3a1f2439a4a9
ContractRegistry.sol	2bad3f21834b921e00a2c69e70976f49b8f0b828
AccessTokenUpgradeExample.sol	4934bdfbf573caed91b947c4ce33fdd13525759a
ExchangeRateProvider.sol	55ae134887bf0ec8b6436dd32026f69f384abf8b
interfaces/IWhitelist.sol	c1f79ab4dfe09e739142cba10bf5e8cb8c7cae00
interfaces/IAccessToken.sol	86ed15fbf886c084deec249dfb47286cfac1d328
interfaces/IBrickblockToken.sol	98db90ef02f16a9bf2097b7f7cbbdaef74e6c39d
interfaces/IPoaToken.sol	0a00f80a0e25d19a9615247ed3f58c79cee592ed
interfaces/IExchangeRates.sol	9f27b08adff3d6451689f6f2eaf60e7f79241676
interfaces/IFeeManager.sol	cc418992580a2b7e471461c0aa71c554edc44206
interfaces/IRegistry.sol	33620967a81de0ecd2b82356eb8ed2eb1e3523cf
interfaces/IExchangeRateProvider.sol	61f0a6d1f06f85f501d755c45f6ab2517a716472
interfaces/IPoaManager.sol	1d09eb035efbf7d087b4e6d60d25480cacf0d1d7

Appendix 2 - Severity

A.2.1 - Minor

Minor issues are generally subjective or potentially deal with topics like "best practices" or "readability". In general, minor issues do not indicate an actual problem or bug in the code.

The maintainers should use their own judgment as to whether addressing these issues improves the codebase.

A.2.2 - Medium

Medium issues are generally objective but do not represent actual bugs or security problems.

These issues should be addressed unless there is a clear reason not to.

A.2.3 - Major

Major issues are things like bugs or security vulnerabilities. These issues may not be directly exploitable or may require a certain condition to arise to be exploited.

Left unaddressed these issues are highly likely to cause problems with the operation of the contract or lead to a situation which allows the system to be exploited in some way.

A.2.4 - Critical

Critical issues are directly exploitable bugs or security vulnerabilities.

Left unaddressed these issues are highly likely or guaranteed to cause critical problems or potentially a full failure in the operations of the contract.

Appendix 3 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer or any other areas beyond Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) -- on its Github account (https://github.com/GNSPS). CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorse the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assume no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.