ackee blockchain

Reserve

Protocol

by Ackee Blockchain

October 7, 2022

Contents

1. Document Revisions

0.1	Draft report	August 26, 2022
1.0	Final report	August 31, 2022
1.1	Fix-review	October 7, 2022

2. Overview

This document presents our findings in reviewed contracts.

2.1. Ackee Blockchain

Ackee Blockchain is an auditing company based in Prague, Czech Republic, specializing in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run free certification courses School of Solana, Summer School of Solidity and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, RockawayX.

2.2. Audit Methodology

1. **Technical specification/documentation** - a brief overview of the system is requested from the client and the scope of the audit is defined.
2. **Tool-based analysis** - deep check with automated Solidity analysis tools and Woke is performed.
3. **Manual code review** - the code is checked line by line for common vulnerabilities, code duplication, best practices and the code architecture is reviewed.
4. **Local deployment + hacking** - the contracts are deployed locally and we try to attack the system and break it.
5. **Unit and fuzzy testing** - run unit tests to ensure that the system works as expected, potentially write missing unit or fuzzy tests.

2.3. Finding classification

A *Severity* rating of each finding is determined as a synthesis of two sub-ratings: *Impact* and *Likelihood*. It ranges from *Informational* to *Critical*.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Info*.

Low to *High* impact issues also have a *Likelihood*, which measures the probability of exploitability during runtime.

The full definitions are as follows:

Severity

		Likelihood			
		High	**Medium**	**Low**	**-**
Impact	**High**	Critical	High	Medium	-
	Medium	High	Medium	Medium	-
	Low	Medium	Medium	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

Table 1. Severity of findings

Impact

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.
- **Warning** - The issue cannot be exploited given the current code and/or configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.), but could be a security vulnerability if these were to change slightly. If we haven't found a way to exploit the issue given the time constraints, it might be marked as a "Warning" or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - The issue is on the borderline between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or configuration (see above) was to change.

Likelihood

- **High** - The issue is exploitable by virtually anyone under virtually any circumstance.
- **Medium** - Exploiting the issue currently requires non-trivial preconditions.
- **Low** - Exploiting the issue requires strict preconditions.

2.4. Review team

Member's Name	Position
Štěpán Šonský	Lead Auditor
Jan Kalivoda	Auditor
Lukáš Böhm	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.5. Disclaimer

We've put our best effort to find all vulnerabilities in the system, however our findings shouldn't be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

3. Executive Summary

Reserve engaged Ackee Blockchain to perform a security review of the Reserve Protocol with a total time donation of 20 engineering days in a period between July 27 and August 25, 2022. Three auditors have performed the audit on the public repository with the following commits and files.

- https://github.com/reserve-protocol/protocol/tree/5cc6e94d9adfdab636a3cf3bfa72888bd6a6020d
  - contracts/p1/*.sol
  - contracts/interfaces/*.sol
  - contracts/plugins/assets/*.sol
  - contracts/plugins/trading/*.sol
  - contracts/libraries/*.sol

The Reserve Protocol introduces a unique concept of tokens backed by stable coins or other collateral assets. The protocol features a governance model, allowing many different trust models, from centralized to decentralized. The code quality is solid, and the protocol architecture is well designed. This audit was performed as a second audit on an already fixed codebase from the first audit that was performed by Trail of Bits. The project contains a large set of unit tests, which covers most of the possible use cases (154/154 passing using `yarn test` command). The unit test coverage is perfect for such a large project.

We began our review by using static analysis tools, namely Slither and the solc compiler. This resulted in some issue suspicions, which we investigated in detail. Most of these issues have been marked as false positives. We then took a deep dive into the logic of the contracts. During the review, we paid special attention to:

- understanding of the protocol architecture,
- line-by-line code review,
- check an upgradeability implementation,
- detecting possible reentrancies in the code,
- ensuring access controls are not too relaxed or too strict,
- looking for common issues such as data validation.

Our review resulted in 10 findings, ranging from Info to Medium severity. The three most severe (medium) issues M1: Unlimited allowance, M2: Downcasting overflow and M3: Insufficient data validation do not directly endanger the protocol in a reasonable timespan. During our review, we investigated some potentially severe issues, even one critical. None of the potentially severe issues was confirmed after writing an exploit script.

Ackee Blockchain recommends Reserve:

- be aware of malicious token implementations,
- remove code duplications,
- address or explain all reported issues,
- add Natspec documentation.

See Revision 1.0 for the system overview of the codebase.

3.1. Revision 1.1

The review was done on the given commit: `6559fcd` from October 6, 2022.

See Revision 1.1 for the review of the updated codebase and additional information we consider essential for the current scope.

4. Summary of Findings

The following table summarizes the findings we identified during our review. Unless overridden for purposes of readability, each finding contains:

- a *Description*,
- an *Exploit scenario*,
- a *Recommendation* and if applicable
- a *Solution*.

There might often be multiple ways to solve or alleviate the issue, with varying requirements regarding the necessary changes to the codebase. In that case, we will try to enumerate them all, clarifying which solves the underlying issue better (albeit possibly only with architectural changes) than others.

	Severity	Reported	Status
M1: Unlimited allowance	Medium	1.0	Acknowledged
M2: Downcasting overflow	Medium	1.0	Fixed
M3: Insufficient data validation	Medium	1.0	Fixed
W1: Code duplications	Warning	1.0	Acknowledged
W2: Basket nonce double increment	Warning	1.0	Fixed
W3: Enum to uint casting	Warning	1.0	Fixed
W4: Wrong revert message	Warning	1.0	Fixed
W5: Support for meta-transactions	Warning	1.0	Acknowledged
W6: Usage of `solc` optimizer	Warning	1.0	Acknowledged

	Severity	Reported	Status
I1: Unnecessary function override	Info	1.0	Acknowledged

Table 2. Table of Findings

5. Report revision 1.0

5.1. System Overview

This section contains an outline of the audited contracts. Note that this is meant for understandability purposes and does not replace project documentation.

Contracts

Contracts we find important for better understanding are described in the following section.

AssetRegistry.sol

The AssetRegistry contract contains a mapping from `IERC20` to `IAsset` and functions to manage them. E.g., `register`, `swapRegistered`, `unregister`, and other functions to convert `IERC20` to `IAsset` or `ICollateral`.

BackingManager.sol

`BackingManager` is crucial from the point of security because it holds and manages the backing assets of the `RToken`.

The contract inherits from `TradingP1` and holds collateral tokens for RToken (with started issuance but not yet vested). It contains `MAX_TRADING_DELAY` constant, which is set to 1 year, and a percentual value `MAX_BACKING_BUFFER = 1e18`. The `backingBuffer` variable is used to keep extra collateral value in the contract. The contract contains following external functions:

- `grantRTokenAllowance` gives `uint256.max` allowance to `rToken` contract for `erc20` parameter,
- `manageTokens` maintains the overall backing policy using recapitalization strategy or `handoutExcessAssets` when `basketHandler` is `fullyCapitalized`.

BasketHandler.sol

This file contains multiple entities dedicated to basket management. Structures `BackupConfig`, `BasketConfig` and `Basket`, library `BasketLib` and the contract `BasketHandlerP1`.

`BasketLib` contains essential util functions for the basket:

- `empty` to empty the basket,
- `copy` to copy one basket to another,
- `add` to add `weight` of the specific collateral token.

`BasketHandlerP1` manages the basket content, configuration and contains all necessary basket operations:

- `disable` disables the basket,
- `refresh` switches the basket (only `governance`),
- `setPrimeBasket` sets `ERC20` tokens and target amounts in the prime basket (only governance),
- `setBackupConfig` sets the backup config for `targetName`,
- `fullyCapitalized` returns `true` if the basket holds enough collateral,
- `status` returns the `CollateralStatus` of the basket,
- `quantity` returns the quantity of the specific `ERC20` token in the basket,
- `price` returns the basket total price according to oracle prices,
- `quote` calculates token quantities for the specific amount of basket unit,
- `basketsHeldBy` returns the amount of the basket units owned by the specific `account`,
- `_switchBasket` is called from the `refresh` function and recalculates the basket with assets that pass `goodCollateral` conditions,

- `goodCollateral` returns `true` if the collateral is registered and `status` is not `CollateralStatus.DISABLED`.

Broker.sol

The `Broker` is a simple contract that deploys disposable trading contracts for `Traders`.

Deployer.sol

The `Deployer` is a factory contract for deploying the entire protocol.

Distributor.sol

Manages the distribution of `RevenueShare` for destinations.

Furnace.sol

Contract for slow, permissionless melting of `RTokens`. Melting speed is determined by `ratio` and `period` variables. `MAX_RATIO` is set to `1e18` = `1` and `MAX_PERIOD` to 1 year. Melting the vested amount can be performed by any external address using the `melt` function.

Main.sol

The central point of the whole system. Holds a reference to RSR token address.

RevenueTrader.sol

The trader component that converts all asset balances at its address to a single target asset and sends this asset to the `Distributor`.

RToken.sol

`RToken` is an `ERC20` token with an elastic supply and governable exchange rate to basket units. Inherits from `ComponentP1`, `ERC20PermitUpgradeable` contracts.

- `MIN_BLOCK_ISSUANCE_LIMIT` is set to 10000 `RTokens` per block.
- `MAX_ISSUANCE_RATE` is set to 1%.

The `issue` function collects the collateral tokens from the `msg.sender` and starts time delayed issuance of `RToken`. The queue can be skipped if the issuance can fit into the current block.

StRSR.sol

`StRSR` is an `ERC20` token with customized logic for staking `RSR`. Staking `RSR` works as insurance behind the `RToken`. Stakers receive a revenue share in `RSR` tokens. The contract contains the following external functions in addition to `ERC20` standard functions:

- `stake` stakes `RSR` amount and mints `StRSR` to the `msg.sender`,
- `unstake` begins a delayed unstaking of `StRSR`,
- `withdraw` completes the delayed unstaking for an `account` up to (excluding) `endId`,
- `seizeRSR` siezes at least `rsrAmount` of `RSR` or reverts,
- `exchangeRate` returns the exchange rate between `StRSR` and `RSR`,
- `endIdForWithdraw` returns the maximum `endId`, which should be working with the `withdraw` function
- `draftQueueLen` returns the `draftQueue` length for an `account` in an `era_`.

StRSRVotes.sol

`StRSRVotes` is an extension of `StRSR`, which is adding voting features. The implementation is based on OpenZeppelin `ERC20VotesUpgradeable`.

plugins/assets/*Collateral.sol

Basic contracts for different collateral types. The `Collateral` contract, which

sits in the `AbstractCollateral.sol` is used as a parent for all `*Collateral` contracts. All of these contracts use `OracleLib` for pricing.

plugins/assets/OracleLib.sol

A `library` for collateral pricing of assets using Chainlink.

libraries/Fixed.sol

`Fixed` is a fixed-point math library that uses `uint192` to represent 18-digit fixed-point numbers and provides all basic mathematical operations.

Actors

This part describes actors of the system, their roles, and permissions.

Owner / Governance

The owner or the governance has privileged permissions over the protocol. However, the trust model is well designed to allow completely decentralized governance and avoid a single point of failure regarding access controls. The governance has the following abilities:

- Set critical parameters of the system,
- Freeze/unfreeze the protocol,
- Register/unregister assets,
- Set backup config in the `BasketHandler`,
- Set distribution (revenue share) in the `Distributor`,
- Set `name`, `symbol`, `unstakinDelay`, `rewardPeriod` and `rewardRatio` in the `StRSR` contract.

Freezer

The freezer role can `oneshotFreeze` the protocol.

Pauser

The pauser role can pause/unpause the protocol.

User

The user means any external address which can interact with the protocol and use its features.

Deployer

The deployer deploys the protocol on the network and has total control over the deployed contract implementations, including malicious tokens.

Trust model

The whole protocol is based on an abstract governance model. Therefore the trust model strongly depends on the actual implementation. The owner can be a single address, multisig, voting governance, or anything else. Every `RToken` can have a different governance model. Also, the governance can transfer its role to another entity.

Users need to trust the deployer in terms of malicious code deployment. Also, users have to trust the governance in case it is centralized.

M1: Unlimited allowance

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	p1/BackingManager.sol	Type:	Data validation

Listing 1. Excerpt from BackingManager.grantRTokenAllowance

```
47         erc20.approve(address(main.rToken()), type(uint256).max);
```

Description

The `BackingManager` contract grants unlimited (`type(uint256).max`) allowance to `main.rToken()` in the `grantRTokenAllowance` function.

Exploit scenario

A malicious `RToken` contract can spend or drain all user's funds.

Recommendation

Allow the contract to operate only with necessary, limited amounts.

Fix 1.1

Acknowledged. Client's response:

" We acknowledge that BackingManager grants unlimited allowance to main.rToken(), and that a malicious RToken contract could spend or drain all user's funds. However, an attacker with the ability to introduce malicious code to the RToken contract address also has the ability to introduce malicious code to the BackingManager contract; these two contracts are trusted contracts in the same security domain. While this does violate the heuristic of minimizing privileges, adding additional security boundaries

throughout our system would substantially increase the system's complexity, which in our judgment yields a more secure system. "

Go back to Findings Summary

M2: Downcasting overflow

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	p1/Furnace.sol	Type:	Overflow

Listing 2. Excerpt from Furnace.melt

```
41         uint32 numPeriods = uint32((block.timestamp) - lastPayout) /
   period;
```

Listing 3. Excerpt from StRSRP1._payoutRewards

```
354         uint32 numPeriods = (uint32(block.timestamp) - payoutLastPaid)
    / rewardPeriod;
```

Listing 4. Excerpt from GnosisTrade.stateTransition

```
76         endTime = uint32(block.timestamp) + auctionLength;
```

Listing 5. Excerpt from BasketLib.empty

```
55         self.timestamp = uint32(block.timestamp);
```

Listing 6. Excerpt from BasketLib.copy

```
68         self.timestamp = uint32(block.timestamp);
```

Listing 7. Excerpt from BasketLib.add

```
85         self.timestamp = uint32(block.timestamp);
```

Description

The protocol uses potentially dangerous downcasting to `uint32` in many contracts listed above.

Exploit scenario

Downcasting the `timestamp` to `uint32` will cause an overflow and protocol misbehavior on February 7, 2106.

Recommendation

Although the likelihood of this issue is low and the potential threat would appear decades in the future, we recommend using a bigger data type e.g. `uint40`. The gas cost would not increase much, and the protocol will be future-proof. Also, we believe gas costs will decrease after a few Ethereum upgrades.

Fix 1.1

Fixed.

Go back to Findings Summary

M3: Insufficient data validation

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	p1/Broker.sol, p1/RevenueTrader.sol, p1/Main.sol	Type:	Data validation

Description

The contracts have insufficient data validation for parameters that are passing addresses in their `init` functions.

- `tokenToBuy_` in RevenueTrader.sol
- `rsr_` in Main.sol
- `gnosis_` in Broker.sol
- `tradeImplementation_` in Broker.sol

Exploit scenario

By accident, an incorrect `tokenToBuy_` is passed to the `init` function. Instead of reverting, the call succeeds.

Recommendation

Add zero-address check for all mentioned parameters.

Fix 1.1

Fixed.

Go back to Findings Summary

W1: Code duplications

Impact:	Warning	Likelihood:	N/A
Target:	plugins/assets/*Collateral.sol	Type:	Best practices

Description

`Collateral` contracts contain a lot of duplicated code. Mainly in `refresh` and `status` functions. Code duplications can lead to bugs during development. Also, code duplications decrease the readability of the code and effective maintainability.

Recommendation

Refactor the architecture to decrease or completely avoid code duplications.

Fix 1.1

Acknowledged. Client's response:

" Acknowledged, and this degree of code duplication is intentional. We're aiming to keep the plugin semantics as simple and intuitive as possible, and each plugin as readable as possible. Previous versions, where we'd aimed to reduce code duplication, were overall less readable and less maintainable than what we currently have. "

Go back to Findings Summary

W2: Basket nonce double increment

Impact:	Warning	Likelihood:	N/A
Target:	p1/BasketHandler.sol	Type:	Data validation

Description

The `copy` function increments the `Basket.nonce` by 2. The first increase happens in the `empty` function:

Listing 8. Excerpt from BasketLib.empty

```
54          self.nonce++;
```

Then the second increase in the `copy` function itself:

Listing 9. Excerpt from BasketLib.copy

```
67          self.nonce++;
```

Recommendation

Check whether this is intended behavior; otherwise, fix this issue.

Fix 1.1

Fixed. Nonce were removed from the BasketLib library.

Go back to Findings Summary

W3: Enum to uint casting

Impact:	Warning	Likelihood:	N/A
Target:	p1/BasketHandler.sol	Type:	Best practices

Listing 10. Excerpt from BasketHandlerP1.status

```
201                 if (uint256(s) > uint256(status_)) status_ = s;
```

Description

Casting `enum` to `uint` ordinal values can be dangerous, and comparing them using `<` and `>` operators even more.

Vulnerability scenario

Unaware developer changes the enum values or order, which can lead to critical system misbehaviors.

Recommendation

Use explicit `==` conditions for enums.

Fix 1.1

Function `worseThans` is now used to compare the values.

Go back to Findings Summary

W4: Wrong revert message

Impact:	Warning	Likelihood:	N/A
Target:	p1/BasketHandler.sol	Type:	Data validation

Listing 11. Excerpt from BasketHandlerP1.refreshBasket

```
120         require(!main.pausedOrFrozen() || main.hasRole(OWNER,
    _msgSender()), "paused or frozen");
```

Description

The `require` statement reverts with the message `"paused or frozen"` when the protocol is not `pausedOrFrozen` and `main.hasRole(OWNER, _msgSender())` returns `false`. This message does not reflect the real reason for the revert.

Also the statement `main.hasRole(OWNER, _msgSender())` is duplicated `governance` modifier.

Recommendation

Use the `governance` modifier on the `refreshBasket` function and change the `require` statement to following:

```
require(!main.pausedOrFrozen(), "paused or frozen");
```

Alternatively, split the `require` statement into two separate `require` conditions with self-describing revert messages.

```
require(main.hasRole(OWNER, _msgSender()), "not the owner");
require(!main.pausedOrFrozen(), "paused or frozen");
```

Fix 1.1

The error message has been changed to ”basket unrefreshable”. The `governance` modifier was not added to the `refreshBasket` function.

Go back to Findings Summary

W5: Support for meta-transactions

Impact:	Warning	Likelihood:	N/A
Target:	`**/*`	Type:	Identity forgery

Description

The protocol is using OpenZeppelin `Context` for potential support for meta-transactions in the future. Meta-transaction support creates a new attack surface via the `_msgSender` function. In a traditional smart contract, users can rely upon the `msg.sender` value is returning the expected value. However, when a body of the `_msgSender` function is adjusted with some additional logic, it can cause different behavior.

For example, in the following case, if a user is the trusted forwarder, he/she can pass any address to the `msg.data` and thus impersonate anyone.

Listing 12. Example of the malicious `_msgSender` function implementation

```
    function _msgSender() internal override virtual view returns (address
ret) {
        if (msg.data.length >= 20 && isTrustedForwarder(msg.sender)) {
            // At this point we know that the sender is a trusted
forwarder,
            // so we trust that the last bytes of msg.data are the verified
sender address.
            // extract sender address from the end of msg.data
            assembly {
                ret := shr(96,calldataload(sub(calldatasize(),20))) ①
            }
        } else {
            ret = msg.sender;
        }
    }
```

① This line will cut the last 20 bytes (address or another payload) from `msg.data` and return it

Ackee Blockchain wrote a blog post about this issue, and it is also mentioned in EIP-2771 in the Security Considerations section.

Recommendation

Pay special attention to the `_msgSender` function. If meta-transactions will be supported in the future, ensure the trusted forwarder address is securely handled.

Fix 1.1

Acknowledged. Client's response:

" It seems true that governance should be warned against this type of failure mode. We will include such a warning in our documentation for governance, but do not intend to modify the code. "

Go back to Findings Summary

W6: Usage of `solc` optimizer

Impact:	Warning	Likelihood:	N/A
Target:	`**/*`	Type:	Compiler configuration

Description

The project uses `solc` optimizer (determined by dotfile). Enabling `solc` optimizer may lead to unexpected bugs.

The Solidity compiler was audited in November 2018, and the audit concluded that the optimizer may not be safe.

Vulnerability scenario

A few months after deployment, a vulnerability is discovered in the optimizer. As a result, it is possible to attack the protocol.

Recommendation

Until the `solc` optimizer undergoes more stringent security analysis, opt-out using it. This will ensure the protocol is resilient to any existing bugs in the optimizer.

Fix 1.1

Acknowledged. Client's response:

" Yes, we acknowledge the additional technical risk that results from using solc's optimization features. We have measured the gas savings from optimizations, and have concluded that:

- Many of our contracts exceed the contract size limit without optimization. We prefer the risks associated with using the optimizer to the risks

associated with the increasing complexity that would arise from breaking single, coherent contracts into multiple deployments.

- The gas savings on our three major user functions (RToken issue, vest, and redeem) are 14%, 11%, and 18%. We expect that our users will benefit more from these gas savings than they are likely to lose from the risk of optimizer failure.

We mitigate (but do not eliminate) this risk in three main ways: - We compile our contracts using the default runs value of 200 in all of our core system contracts. This ensures that the optimizations that we are using are the most commonly-used optimizations, and thus optimizer bugs are marginally less likely. - We are not using the very newest version of solc. Instead, we've chosen our compiler version to be as old as possible, while still including the language features on which we rely and any known security patches that may affect the correctness of our code. Thus, it's marginally less likely for serious compiler bugs to exist in that version. - Our system is upgradable and contains an emergency-stop mechanism. While this cannot eliminate the possibility of zero-day attacks due to as-yet unknown optimizer bugs, if the community learns about optimizer bugs that affect the code we've already deployed, it can likely pause the whole system more quickly than the system can be attacked, and then it should be relatively straightforward to upgrade the whole system using contracts compiled differently. "

Go back to Findings Summary

I1: Unnecessary function override

Impact:	Info	Likelihood:	N/A
Target:		Type:	Best practices

Description

The contract `Collateral` overrides the function `price` from `Asset` and `IAsset`, but the `Asset` contract already implements this function using the same logic.

Listing 13. Excerpt from Collateral.price

```
37     function price() public view virtual override(Asset, IAsset) returns
   (uint192) {
38         return chainlinkFeed.price(oracleTimeout);
39     }
```

Listing 14. Excerpt from Asset.price

```
49     function price() public view virtual returns (uint192) {
50         return chainlinkFeed.price(oracleTimeout);
51     }
```

Another occurrence of this finding is situated in the `Main` contract. The `hasRole` function overrides `IAccessControlUpgradeable` and `AccessControlUpgradeable`, but calls only `super.hasRole(role, account)`.

Listing 15. Excerpt from MainP1.hasRole

```
59     function hasRole(bytes32 role, address account)
60         public
61         view
62         override(IAccessControlUpgradeable, AccessControlUpgradeable)
63         returns (bool)
64     {
65         return super.hasRole(role, account);
```

```
66     }
```

Recommendation

Remove unnecessary function overridings if they have no intended reason. Otherwise, ignore this informational finding.

Fix 1.1

Acknowledged. Client's response:

" We're not overriding the named functions to change their semantics, we're overriding this function because the contract type signatures demand it. We're not happy about it either, but (a) the system won't compile without this explicit override, and (b) we consider it out-of-scope for the current project to improve the language. "

Go back to Findings Summary

6. Report revision 1.1

6.1. System Overview

No significant changes were performed in the contracts. All the changes are responding to reported issues.

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, Reserve: Protocol, October 7, 2022.

ackee

Thank You

Ackee Blockchain a.s.


Prague, Czech Republic


hello@ackeeblockchain.com


https://discord.gg/z4KDUbuPxq