ackee blockchain

Overnight Finance

Core

by Ackee Blockchain

3.3.2023

Contents

1. Document Revisions

0.1	Draft report	Feb 13, 2023
1.0	Final report	March 3, 2022
1.1	Fix review	March 3, 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
Jan Kalivoda	Lead 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

Overnight Finance is a protocol that presents yield-generating stablecoin pegged to USDC.

Revision 1.0

Overnight finance engaged Ackee Blockchain to perform a security review of the Core of the protocol with a total time donation of 10 engineering days in a period between January 23 and February 3, 2023 and the lead auditor was Jan Kalivoda.

The audit has been performed on the commit `291d5be` in ovnstable-core repository and the scope of the audit was the following files:

- Exchange.sol
- PortfolioManager.sol
- UsdPlusToken.sol
- Mark2Market.sol
- PayoutListener.sol
- Strategy.sol

We began our review by using static analysis tools, namely Slither and Woke. We then took a deep dive into the logic of the contracts. For testing and fuzzing, we have involved Woke testing framework. During the review, we paid special attention to:

- ensuring nobody can redeem/steal others' funds,
- checking if the code matches stablecoin's specification,
- checking correctness of the upgradeability pattern,

- ensuring the arithmetic of the system is correct,
- 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 14 findings, ranging from Info to Medium severity.

Ackee Blockchain recommends Overnight Finance:

- address all reported issues.

See Revision 1.0 for the system overview of the codebase.

Revision 1.1

The review was done between February 28 and March 3, 2023, on the given commit: `abfbc55` and the scope was only the raised issues from the Revision 1.0.

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: Unchecked return values for token transfers	Medium	1.0	Acknowledged
M2: Divison by zero if parameters are not set	Medium	1.0	Fixed
W1: Usage of `solc` optimizer	Warning	1.0	Acknowledged
W2: Wide Solidity pragma usage	Warning	1.0	Fixed
W3: For cycle in the `payout` function can revert	Warning	1.0	Acknowledged
I1: Inconsistent usage of `msg.sender` over `_msgSender`	Info	1.0	Fixed

	Severity	Reported	Status
I2: The lockfile can be overwritten	Info	1.0	Fixed
I3: Usage of hardcoded value instead of constant	Info	1.0	Fixed
I4: Unused function parameter	Info	1.0	Fixed
I5: The `payout` function could be external	Info	1.0	Fixed
I6: Contract id based validation	Info	1.0	Acknowledged
I7: Use pre-incrementation in for cycles	Info	1.0	Acknowledged
I8: Upgrader role is used inconsistently	Info	1.0	Fixed
I9: The `initSlippages` and `setSlippages` functions could be merged	Info	1.0	Fixed

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.

USDPlusToken

USD+ is the main token of the protocol, ERC20, and a rebased stablecoin. The contract is based on the OpenZeppelin ERC20 implementation and is upgradeable via the UUPSUpgradeable pattern.

Exchange

The exchange contract allows minting, redeeming of USD+ and provides payouts for the users from given strategies. The contract is upgradeable via the UUPSUpgradeable pattern. Swap of USD+ is done with a fixed rate (1:1) against USDC or other stablecoins, however, swaps include fees. Payouts are initially set to be done once a day. The contract is also pausable. There are several roles defined in the contract that have elevated privileges that are critical for the system to function properly. The roles are described in the Actors section. The contract is upgradeable via the UUPSUpgradeable pattern.

PortfolioManager

The portfolio manager receives Mark2Market data and compares the portfolio

structure to the added strategies. It is responsible for rebalancing the portfolio to meet the strategy via the `balance` function. It contains `withdraw` and `deposit` functions that are used to stake/unstake funds into strategies. Admin of the contract is responsible for setting various parameters, more in the Actors section. The `claimAndBalance` function is used from Exchange for payouts. The contract is upgradeable via the UUPSUpgradeable pattern.

Mark2Market

The contract obtains data directly from PortfolioManager about current assets managed by strategies and their value in USDC. The contract is upgradeable via the UUPSUpgradeable pattern.

Strategy

Strategy is an abstract contract that can have various implementations that generates yield for the protocol. The contract is upgradeable via the UUPSUpgradeable pattern.

Actors

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

USD+ Admin

The role is responsible for setting the Exchanger role and decimals of the token if they are not zero.

Exchanger

The address of the Exchange contract. It can mint/burn USD+ tokens and set the liquidity index in terms of the USDPlusToken contract. In terms of PortfolioManager, the role can call `withdraw`, `deposit` and `claimAndBalance` functions.

Portfolio Agent

The role sets fees for the Exchange contract, payout times, oracle loss, compensate loss and abroad parameters. Finally, the role can pause the Exchange contract. In terms of PortfolioManager, the role can set weight for strategies and call `balance` for portfolio rebalancing. In terms of Strategy, the role can set slippages for the strategies.

Portfolio Manager

The role can call `stake`, `unstake` and `claimRewards` functions of the Strategy contracts. It should be the address of the PortfolioManager contract.

Free rider

The role can bypass fees for the Exchange contract and bypass the `oncePerBlock` modifier.

Unit

The role can call the `payout` function of the Exchange contract.

Exchange Admin

The role is responsible for setting the Portfolio Agent role and possibly other roles when is in charge, because after `initialize` or `changeAdminRoles` is called, the role is no longer an admin role for Free rider and Unit. Otherwise, the role can set the tokens for Exchange, protocol's components addresses and profit recipient.

Mark2Market Admin

The role sets the PortfolioManager address.

PayoutListener Admin

The role sets the Exchanger address.

Strategy Admin

The role can set slippages for derived Strategy contracts and set the Portfolio Agent role that can be different from PortfolioManager contract (access modifier accepts both variants). The role also sets the PortfolioManager address.

Upgrader

The role that is responsible for upgrading the following contracts:

- USDPlusToken
- PortfolioManager
- Mark2Market
- PayoutListener

The Exchange, Strategy are upgradeable with the default admin role.

5.2. Trust model

Users of the protocol should trust that the users with elevated privileges will set the parameters correctly since there are multiple possible attack vectors in terms of trust. For example, changing the asset address on Exchange or changing the Exchanger address in USDPlusToken can lead to unlimited mint.

M1: Unchecked return values for token transfers

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	Exchange.sol, PortfolioManager.sol, Strategy.sol	Type:	Unchecked return value

Listing 1. Excerpt from Exchange._buy

```
312         IERC20(_asset).transferFrom(msg.sender,
    address(portfolioManager), _amount);
313         portfolioManager.deposit(IERC20(_asset), _amount);
314
315         uint256 buyFeeAmount;
316         uint256 buyAmount;
317         (buyAmount, buyFeeAmount) = _takeFee(usdPlusAmount, true);
318
319         usdPlus.mint(msg.sender, buyAmount);
```

Description

Transfers are not checking the return value. This can cause problems when the protocol contains tokens that don't match the expected behavior, such as tokens that don't revert on failed transfers.

Unsafe transfers can be found in the following functions:

- Exchange.`_buy`
- Exchange.`redeem`
- Exchange.`payout`
- PortfolioManager.`setCashStrategy`

- PortfolioManager.`deposit`
- PortfolioManager.`withdraw`
- PortfolioManager.`_balance`
- Strategy.`unstake`

Exploit scenario

The asset in Exchange does not revert on failed transfer and instead of that it returns false. As a result, USD+ is minted without transferring any asset.

Recommendation

Check for the return values of transfers or use SafeERC20 from OpenZeppelin.

Client's response

Acknowledged by the client.

> We only use trusted tokens in our protocol. These are tokens that can be borrowed on Aave. And with them there are no problems during the transfer. Therefore, in most cases, this check is redundant. But just in case, we decided to add extra. verification when investing in our protocol (method Exchange.buy). Also we will be use SafeERC20 in our strategies where can use "not popular" tokens.
>
> — Overnight Finance

Fix 1.1

The code is adjusted to check the contract's balance before and after the transfer.

```
uint256 _targetBalance = usdc.balanceOf(address(portfolioManager)) +
_amount;
usdc.transferFrom(msg.sender, address(portfolioManager), _amount);
require(usdc.balanceOf(address(portfolioManager)) == _targetBalance, 'pm
balance != target');
```

However, there is still a possible incompatibility with some tokens that do not respect ERC20 standard. For example, USDT will always revert to this type of transfer.

Go back to Findings Summary

M2: Divison by zero if parameters are not set

Medium severity issue

Impact:	Medium	Likelihood:	Low
Target:	Exchange.sol	Type:	Math error, DoS

Description

If `oracleLossDenominator` or `compensateLossDenominator` are not set, the `payout` function is unable to proceed on negative rebase and the transaction is reverted.

Exploit scenario

Portfolio Agent did not call `setOracleLoss` and `setCompensateLoss` functions. Unit calls the `payout` function and it reverts on division by zero.

Recommendation

Ensure that these values are initialized to a non-zero value or adjust the logic to handle the case when they are zero.

Fix 1.1

The variables are now initialized in the constructor.

Go back to Findings Summary

W1: Usage of `solc` optimizer

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

Description

The project uses `solc` optimizer. 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.

Client's response

Acknowledged by the client.

> We tried to remove optimizations, but got a 2-fold increase in the size of contracts.
>
> — Overnight Finance

Go back to Findings Summary

W2: Wide Solidity pragma usage

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

Description

The contracts are using a wide range of Solidity versions. In case of USDPlusToken it is `>=0.5.0 <0.9.0`, for the rest of the files it is `>=0.8.0 <0.9.0`. This can cause unexpected behavior if the version of the compiler used to compile the contracts is different from the one that was properly tested.

Recommendation

Choose a single version of Solidity and use it consistently across all the contracts.

Fix 1.1

The pragma is set to `>=0.8.0 <0.9.0` in all of the contracts. This shouldn't be a problem if the deployment is handled correctly.

Go back to Findings Summary

W3: For cycle in the `payout` function can revert

Impact:	Warning	Likelihood:	N/A
Target:	`**/*`	Type:	Gas optimization, DoS

Listing 2. Excerpt from Exchange.payout

```
527         for (; block.timestamp >= nextPayoutTime - payoutTimeRange;) {
528             nextPayoutTime = nextPayoutTime + payoutPeriod;
529         }
```

Description

Calling the `payout` function can run out of gas if there will be a lot of iterations or at least be pricy on gas.

Recommendation

Consider using a different approach to avoid the `for` loop while preserving security. For example, on each payout, the `payout` function will be locked for one day.

Client's response

Acknowledged by the client.

> We need this cycle because allows you to keep a fixed payout execution time in case we failed to complete the payout on time. This is a business requirement. Since we are responsible for the execution of the payout, and not the users, we are ready to bear the loss of gas in the event of a delay in the execution of the payout.

> — Overnight Finance

Go back to Findings Summary

I1: Inconsistent usage of `msg.sender` over `_msgSender`

Impact:	Info	Likelihood:	N/A
Target:	USDPlusToken.sol	Type:	Logic error

Description

The protocol is using OpenZeppelin `Context` that defines `_msgSender` and `_msgData` functions. This makes it easy to switch their semantics, e.g. if developers decides to support meta-transactions in the future. If a contract inherits from `Context`, uses of `msg.data` and `msg.sender` should be replaced by `internal` calls to `_msgData` and `_msgSender`, respectively. This will ensure that if the semantics is changed in the future, the codebase will remain consistent. There are currently no uses of `msg.data`, but a few uses of `msg.sender`.

Since the `msg.sender` occurrences are only used in the admin functions, the issue is considered informational.

Recommendation

Replace all instances of `msg.sender` with `_msgSender` if you are planning to change the semantics. Otherwise, use `msg.sender` consistently.

Fix 1.1

The `msg.sender` occurrences were replaced with `_msgSender`.

Go back to Findings Summary

I2: The lockfile can be overwritten

Impact:	Info	Likelihood:	N/A
Target:	`**/*`	Type:	Package management

Description

According to the documentation, the packages are installed via the `yarn` command. This execution can overwrite the lockfile and thus it can lead to unexpected behavior due to different package versions.

Recommendation

Use `yarn --frozen-lockfile` instead of `yarn` to avoid overwriting the lockfile.

Fix 1.1

The documentation was updated accordingly.

Go back to Findings Summary

I3: Usage of hardcoded value instead of constant

Impact:	Info	Likelihood:	N/A
Target:	UsdPlusToken.sol	Type:	Constants

Listing 3. Excerpt from UsdPlusToken.approve

```
278     function approve(address spender, uint256 amount) external override
    returns (bool){
279         uint256 scaledAmount;
280         if (amount > (type(uint256).max / liquidityIndex / 10 ** 9)) {
281             scaledAmount = type(uint256).max;
282         } else {
```

Description

The `approve` function uses a hardcoded value instead of a constant in a calculation. This causes worse readability and maintainability.

Recommendation

Replace the hardcoded value with a constant or add a proper code comment for the value.

Fix 1.1

The code block is now properly documented.

Go back to Findings Summary

I4: Unused function parameter

Impact:	Info	Likelihood:	N/A
Target:	PortfolioManager	Type:	Dead code

Description

The `deposit` function in the `PortfolioManager` contract has an unused parameter `_amount`. As a result, any address that is passed does not affect the function.

Recommendation

Remove the unused parameter or implement it.

Fix 1.1

The unused parameter is removed.

Go back to Findings Summary

I5: The `payout` function could be external

Impact:	Info	Likelihood:	N/A
Target:	Exchange.sol	Type:	Gas optimization

Description

The `payout` is public and could be external since it is going to be called only by an external service.

Recommendation

Change the function visibility to `external`.

Fix 1.1

The function is now declared as `external`.

Go back to Findings Summary

I6: Contract id based validation

Impact:	Info	Likelihood:	N/A
Target:	`**/*`	Type:	Data validation

Description

The project uses zero-address checks for addresses data validation, however, validation can be more stringent if contract ids are used.

Recommendation

To each component that is passed to another add a constant variable that contains the contract id and use it for validation.

For example, the Exchange contract will contain a variable named `CONTRACT_ID`:

```
bytes32 public constant CONTRACT_ID = keccak256("OVN Exchange");
```

and the `setExchanger` function in PortfolioManager will contain a check for the value of this variable:

```
require(
    PortfolioManager(_exchanger).CONTRACT_ID() == keccak256("OVN
Exchange"),
    "Invalid exchanger address"
);
```

This will help to reduce the risk of passing incorrect values.

Client's response

Acknowledged by the client.

> We use own CLI and deploy scripts for checking this validations. Also, the addition of new checks entails an increase in the code base. This is what we are trying to avoid.
>
> But the verification method is interesting, we will take it into service in our other projects.
>
> — Overnight Finance

Go back to Findings Summary

I7: Use pre-incrementation in for cycles

Impact:	Info	Likelihood:	N/A
Target:	`**/*`	Type:	Gas optimization

Description

The project contains for cycles with post-incrementation. The pre-incrementation is more gas efficient.

Recommendation

Use pre-incrementation in for cycle headers instead of post-incrementation.

Go back to Findings Summary

I8: Upgrader role is used inconsistently

Impact:	Info	Likelihood:	N/A
Target:	`**/*`	Type:	Access controls

Description

The project is mostly using Upgrader role for upgrades, but there are places where Upgrader role is not used:

- Exchanger
- Strategy

Recommendation

Ensure that this is a wanted behavior and not an issue. Otherwise, use Upgrader role consistently.

Fix 1.1

The Upgrader role was removed and the default admin role was used instead.

Go back to Findings Summary

I9: The `initSlippages` and `setSlippages` functions could be merged

Impact:	Info	Likelihood:	N/A
Target:	`**/*`	Type:	Code duplicity

Listing 4. Excerpt from Strategy.initSlippages & Strategy.setSlippages

```
71     function initSlippages(
72         uint256 _swapSlippageBP,
73         uint256 _navSlippageBP
74     ) public onlyAdmin {
75         swapSlippageBP = _swapSlippageBP;
76         navSlippageBP = _navSlippageBP;
77         emit SlippagesUpdated(_swapSlippageBP, _navSlippageBP);
78     }
79
80     function setSlippages(
81         uint256 _swapSlippageBP,
82         uint256 _navSlippageBP
83     ) public onlyPortfolioAgent {
84         swapSlippageBP = _swapSlippageBP;
85         navSlippageBP = _navSlippageBP;
86         emit SlippagesUpdated(_swapSlippageBP, _navSlippageBP);
87     }
```

Description

The `initSlippages` and `setSlippages` functions have the same content but different access controls (see Listing 4).

Recommendation

Consider merging these two functions, since the `initSlippages` can be called repeatedly by Strategy Admin and does not work as the typical ”init” function.

Fix 1.1

The `initSlippages` function is removed.

Go back to Findings Summary

6. Report revision 1.1

The following issues were fixed:

- M2: Divison by zero if parameters are not set,
- W2: Wide Solidity pragma usage,
- I1: Inconsistent usage of `msg.sender` over `_msgSender`,
- I2: The lockfile can be overwritten,
- I3: Usage of hardcoded value instead of constant,
- I4: Unused function parameter,
- I5: The `payout` function could be external,
- I8: Upgrader role is used inconsistently,
- I9: The `initSlippages` and `setSlippages` functions could be merged,

and the rest of the issues were acknowledged. For more information see each finding in Summary of Findings.

6.1. System Overview

The Upgrader role was replaced with the default admin role.

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, Overnight Finance: Core, 3.3.2023.

Appendix B: Glossary of terms

The following terms might be used throughout the document:

Superclass/Ancestor of C

A contract that C inherits/derives from.

Subclass/Child of C

A contract that inherits/derives from C.

Syntactic contract

A Solidity contract. May have an inheritance chain, and may be deployed.

Deployed contract

An EVM account with non-zero code. If its source was written in Solidity, it was created through at least one syntactic contract. If that contract had superclasses (parents), it would be composed of multiple syntactic contracts.

Init/initialization function

A non-constructor function that serves as an initializer. Often used in upgradeable contracts.

External entrypoint

A `public` or `external` function.

Public/Publicly-accessible function/entrypoint

An `external` or `public` function that can be successfully executed by any network account.

Mutating function

A non-`view` and non-`pure` function.

ackee

Thank You

Ackee Blockchain a.s.


Prague, Czech Republic


hello@ackeeblockchain.com


https://discord.gg/z4KDUbuPxq