ChainLight

Kurrency

WEMADE - Kurrency Security Audit

: Kurrency PLAY

Oct 13, 2023

Revision 1.0

ChainLight@Theori

Table of Contents

Executive Summary

Starting on September 27, 2023, ChainLight of Theori assessed the Kurrency PLAY for four days. Kurrency PLAY is a new product based on Kurrency's code that we audited multiple times across revisions. Most code changes were deletions since it does not have liquidation or collateral asset investing. Codes differences from the original Kurrency were reviewed more thoroughly, although we also checked other parts. We focused on identifying issues that put collaterals at risk, allow unauthorized modification of configuration parameters, or diverge from the intended behavior.

As a result, we found seven (including a critical issue that could lead to theft of funds, one medium issue that could lead to partial theft of funds when uncommon configuration is used, and five informational issues concerning potential threats and defense-in-depth) issues. Any regulation issues are out of the scope of this security audit; specifically, this assessment does not consider any legislation or regulation on stablecoins and financial products in general from any legislative or regulatory body based in Singapore or South Korea.

The report does not take into account any Maximum Extractable Value (MEV) issues, which have minimal impact on the WEMIX 3.0 blockchain that uses a highly centralized consensus algorithm (Stake-based Proof of Authority) that inherently allows operators and stakeholders to disincentivize attack attempts by slashing the offender from the consensus process.

For more information, please check the finding details below.

Audit Overview

Scope

Name	WEMADE - Kurrency Security Audit
Target / Version	• Git Repository (kurrency-contract-play-audit): commit `1583d4aea7492ba7de08616119293bc06229cc15` ~ `ece8367dfeacf9cbf57dd2ec502add8cb44c66b9`
Application Type	Smart contracts
Lang. / Platforms	Smart contracts [Solidity]

Code Revision

N/A

Severity Categories

Severity	Description
Critical	The attack cost is low (not requiring much time or effort to succeed in the actual attack), and the vulnerability causes a high-impact issue. (e.g., Effect on service availability, Attacker taking financial gain)
High	An attacker can succeed in an attack which clearly causes problems in the service's operation. Even when the attack cost is high, the severity of the issue is considered "high" if the impact of the attack is remarkably high.
Medium	An attacker may perform an unintended action in the service, and the action may impact service operation. However, there are some restrictions for the actual attack to succeed.
Low	An attacker can perform an unintended action in the service, but the action does not cause significant impact or the success rate of the attack is remarkably low.
Informational	Any informational findings that do not directly impact the user or the protocol.

Status Categories

Status	Description
Confirm	ChainLight reported the issue to the vendor, and they confirm that they received.
Reported	ChainLight reported the issue to the vendor.
Fixed	The vendor resolved the issue.
Acknowledged	The vendor acknowledged the potential risk, but they will resolve it later.
WIP	The vendor is working on the patch.
Won't Fix	The vendor acknowledged the potential risk, but they decided to accept the risk.

Finding Breakdown by Severity

Category	Count	Findings
Critical	1	• KURRENCYPLAY-005
High	0	• N/A
Medium	1	• KURRENCYPLAY-004
Low	0	• N/A
Informational	5	• KURRENCYPLAY-001 • KURRENCYPLAY-002 • KURRENCYPLAY-003 • KURRENCYPLAY-006 • KURRENCYPLAY-007

Findings

Summary

#	ID	Title	Severity	Status
1	KURRENCYPLAY-001	Implied price (from `exchangeRatio`) lower than the market price may lead to usability issues	Informational	Won't Fix
2	KURRENCYPLAY-002	`rpow()` function is missing in `MathUpgradeable` library	Informational	Fixed
3	KURRENCYPLAY-003	`SafeERC20` library must check that the `token` address is a contract	Informational	Fixed
4	KURRENCYPLAY-004	Dust threshold should be reduced for tokens with decimals lower than 18	Medium	Fixed
5	KURRENCYPLAY-005	`PlayManager` should use token's decimals instead of `18` for some calculations involving token amount	Critical	Fixed
6	KURRENCYPLAY-006	`MP` enabled `NCPStaking` pool must not be used in the investor	Informational	Fixed
7	KURRENCYPLAY-007	Additional sanity checks	Informational	Fixed

#1 `KURRENCYPLAY-001` Implied price (from `exchangeRatio`) lower than the market price may lead to usability issues

ID	Summary	Severity
`KURRENCYPLAY-001`	If the price of game tokens in the Play contract remains lower than the market price due to price fluctuations of game tokens or a delay in updating the `exchangeRatio`, arbitrage may happen, which leads to depletion of game tokens held by the Play contract.	Informational

Description

Due to price fluctuations of game tokens or a delay in updating the `exchangeRatio`, the price of game tokens in the Play contract may differ from the market price. If the price of game tokens in the Play contract is low compared to other markets, game tokens held by the Play contract may be depleted by arbitrage, and the market price may decrease. If the price gap widens in the opposite case, users who have deposited collateral and borrowed less than the maximum LTV may be unhappy. This situation prevents smooth contract usage for users who want to earn tokens for gameplay, so it should be mitigated.

Impact

Informational

Recommendation

When setting the `exchangeRatio` in the keeper, it should not be set to a value that is a discount to the market price. If there is logic to supply additional tokens automatically, a failsafe should be added to check that the `exchangeRatio` is appropriate before doing so. Additionally, the contract can include a check to deny borrowing if the keeper has not been active for a certain time or a check to compare the price in Oracle with the current `exchangeRatio` based price.

Patch

Won't Fix

Allowing arbitrage is intended by design. But the team will double-check if it would lead to excessive insurance of game tokens at the early stage.

#2 `KURRENCYPLAY-002` `rpow()` function is missing in `MathUpgradeable` library

ID	Summary	Severity
`KURRENCYPLAY-002`	`MathUpgradeable.sol` library does not implement `rpow()` which exists in `Math.sol`.	Informational

Description

`MathUpgradeable.sol` library does not implement `rpow()` which exists in `Math.sol`, so the code that calls `rpow()` will not be compiled when `MathUpgradeable.sol` library is used instead of `Math.sol` in the future.

Impact

Informational

Recommendation

Implement `Math.rpow()` in `MathUpgradeable.sol` or delete the library if it is unnecessary.

Patch

Fixed

It is fixed as recommended.

#3 `KURRENCYPLAY-003` `SafeERC20` library must check that the `token` address is a contract

ID	Summary	Severity
`KURRENCYPLAY-003`	Functions in the `SafeERC20` library that take a `token` as an argument must check that the `token` address is a contract.	Informational

Description

Functions in the `SafeERC20` library that take a `token` as an argument do not check that the `token` address is a contract. The code below calls the `token` and checks the result to determine whether to revert.

```
(bool success, bytes memory data) = address(token).call(abi.encodeWithSelector(function selector, to, value));
require(success && (data.length == 0 || abi.decode(data, (bool))), "SafeERC20: TRANSFER_FAILED");
```

If the `token` address is not a contract, it always does not revert regardless of whether the function call succeeds or fails. Therefore, the protocol may behave differently than intended if the contract uses the `SafeERC20` library without validating the `token` address. We haven't found any paths that use invalid `token` addresses, but fixing them is recommended because it differs from the expected behavior of the `SafeERC20` library.

Impact

Informational

Recommendation

In the `SafeERC20` library, functions that take a `token` as an argument must check that the `token` address is a contract (`code` length is non-zero).

Patch

Fixed

It is fixed as recommended.

#4 `KURRENCYPLAY-004` Dust threshold should be reduced for tokens with decimals lower than 18

ID	Summary	Severity
`KURRENCYPLAY-004`	`PlayManager._updateDebtInfo()` rounds down the debt to 0 if there is a little debt (dust) left after repayment, but this threshold is based on $1 tokens with 18 decimals. So, dust's value can be large enough to drain the funds if the token's `decimals` is lower and the price is the same or higher.	Medium

Description

When the token price is the same as $1, the dust value of the 18 decimals token is about $0.000000000000001, but in the case of 6 decimals, it is about $0.001. Since a transaction fee of approximately $0.1 is incurred while performing borrow and repay, it is difficult to exploit this even if the `decimals` is low. However, it may become exploitable if the token's price is high or the attack cost is reduced in the future due to various reasons, such as lower transaction fees or lower price of the native token.

Impact

Medium

Since most ERC20 tokens use 18 decimals and the price is not that high, the likelihood of a profitable attack is low, and even in a situation where an attack is possible, a substantial number of repetitions is required to steal a significant amount of funds.

Recommendation

When the decimals of the `gameToken` is less than 18, the dust threshold can be lowered to 10 to reduce the possibility of attack significantly, and tokens with low decimals and high prices should not be used because they can still be a problem.

Patch

Fixed

It is fixed as recommended.

#5 KURRENCYPLAY-005 PlayManager should use token's decimals instead of 18 for some calculations involving token amount

ID	Summary	Severity
KURRENCYPLAY-005	Suppose the decimals of the collateral token or game token are not 18. In that case, incorrect results may occur in the `isOverMaxBorrowableAmount()`, `getCollateralValueOf()`, and `getBorrowableAmount()` functions of the `PlayManager` contract, which may lead to the theft of funds.	Critical

Description

`getCollateralValueOf()` returns the collateral value deposited in the vault converted to USD value in `1e18` units. For this purpose, `return amount * tokenPrice / 1e18;` is used, where `tokenPrice` means the USD value of 1 token (`1e18` decimals). Therefore, the above formula can be viewed as calculating the token quantity (`amount / 1e18`) * USD price of one token (`tokenPrice`). However, `amount / 1e18` does not consider the case where the decimals of the collateral token are not 18. (It must have been divided by the decimals of the collateral token, not `1e18`.)

Similar to above, the decimals of the game token must be considered when calculating `maxBorrowAmount` in `isOverMaxBorrowableAmount()`. In the formula for calculating the maximum amount of game tokens that can be borrowed, `uint maxBorrowAmount = collateralValue * exchangeRatio / 1e18;`, `collateralValue` is the USD value of collateral in `1e18` units, and the `exchangeRatio` variable is in `1e18` units, so the decimals of `maxBorrowAmount` is always fixed to 18. Therefore, the formula should be changed to `uint maxBorrowAmount = collateralValue * exchangeRatio * (10 ** gameToken.decimals()) / 1e36;` to take into account the decimals of the game token.

Also in `getBorrowableAmount()`, formula `uint vaultBorrowableAmountMax = getCollateralValueOf(pairId, vaultBalance) * exchangeRatio / 1e18;` should be changed to `uint vaultBorrowableAmountMax = getCollateralValueOf(pairId, vaultBalance) * exchangeRatio * (10 ** gameToken.decimals()) / 1e36;` to take into account the decimals of the game token.

Impact

Critical

Suppose the decimals of the collateral token or game token are not 18. In that case, an excessive amount of game tokens can be loaned compared to the collateral, or a tiny amount can be loaned compared to the collateral.

Recommendation

All changes below must be applied.

1. Change `1e18` to decimals of `collateralToken` in `getCollateralValueOf()`.
2. Change the formula in `isOverMaxBorrowableAmount()` to `uint maxBorrowAmount = collateralValue * exchangeRatio * (10 ** gameToken.decimals()) / 1e36;`.
3. Change the formula in `getBorrowableAmount()` to `uint vaultBorrowableAmountMax = getCollateralValueOf(pairId, vaultBalance) * exchangeRatio * (10 ** gameToken.decimals()) / 1e36;`.

Patch

Fixed

It is fixed as recommended.

#6 KURRENCYPLAY-006 MP enabled NCPStaking pool must not be used in the investor

ID	Summary	Severity
KURRENCYPLAY-006	There is a problem with the MP reward implementation in the third-party contract NCPStaking, which can cause losses to the protocol when using pools with that feature enabled.	Informational

Description

Due to a problem with the MP reward implementation in the NCPStaking contract, the InvestorWonder contract may receive less reward if they interact with the NCPStaking contract less frequently than other users when investing assets in a pool with the MP feature enabled. Kurrency Play deposits WEMIX collateral into the WONDER staking pool. WONDER staking pool is a pool registered to the NCPStaking contract, and according to WEMIX.FI docs it does not support the MP feature. If a WONDER staking pool with MP enabled or a DIOS staking pool that originally supports MP is accidentally used for collateral asset investment, the protocol may suffer losses.

Impact

Informational

It is not a problem as of now since the protocol only uses a WONDER staking pool that does not support MP.

Recommendation

Until the bug in the NCPStaking contract is fixed, the team should be careful not to use pools that support the MP feature.

Patch

Fixed

It is fixed as recommended.

#7 KURRENCYPLAY-007 Additional sanity checks

ID	Summary	Severity
KURRENCYPLAY-007	It is necessary to add a validation process to prevent incorrect settings caused by operational mistakes, mitigate potential issues, and implement defense-in-depth.	Informational

Description

- If `PlayFeeBox.withdrawFee()` is called before `feeReceiver` is set, the accumulated fee may be burned.
- Add the missing `onlyFeeReceiverSet()` modifier to `PlayFeeBox.withdrawFee()`.

- `PlayManager.setMultipleVaultWhitelist()` does not check if the address to be whitelisted has a debt, so if a user with a debt is whitelisted, they will not be able to repay their existing debt, and the collateral will be locked.
- In `PlayManager.setMultipleVaultWhitelist()`, check if the address to be whitelisted has a debt.

- `PlayConfig.setWonderStaking()` and `PlayConfig.setWWEMIX()` do not check if the address passed in as an argument is a contract.
- Check that the address passed in as an argument is a contract.

- The address validation of `_gameToken` and `_investor` in `PlayManager.addPairInfo()` should validate that they are contracts instead of comparing them to zero.

- In `PlayCore._deposit()`, if a token other than the native token is collateralized, checking that the `msg.value` is 0 can prevent accidentally sending the native token.

- Add the `nonReentrant` modifier to `playRewardDistributor.distributeRewardAll()`.

Impact

Informational

Recommendation

Apply the fixes suggested in the Description.

Patch

Fixed

It is fixed as recommended.

Revision History

Version	Date	Description
1.0	Oct 13, 2023	Initial version

Theori, Inc. (“We”) is acting solely for the client and is not responsible to any other party. Deliverables are valid for and should be used solely in connection with the purpose for which they were prepared as set out in our engagement agreement. You should not refer to or use our name or advice for any other purpose. The information (where appropriate) has not been verified. No representation or warranty is given as to accuracy, completeness or correctness of information in the Deliverables, any document, or any other information made available. Deliverables are for the internal use of the client and may not be used or relied upon by any person or entity other than the client. Deliverables are confidential and are not to be provided, without our authorization (preferably written), to entities or representatives of entities (including employees) that are not the client, including affiliates or representatives of affiliates of the client.