Three Sigma

Code Audit

CodeUP DeFi game based on an Ethereum story

Disclaimer

Code Audit

CodeUP DeFi game based on an Ethereum story

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Code Audit

CodeUP DeFi game based on an Ethereum story

Table of Contents

Summary

Code Audit

CodeUP DeFi game based on an Ethereum story

Summary

Three Sigma audited CodeUP in a 1 person week engagement. The audit was conducted from 23-10-2024 to 27-10-2024.

Protocol Description

CODE UP is a DeFi game where players build towers with different floors, hiring builders with various skills to develop Ethereum 2.0. Builders reside on these floors and generate gameETH. By accumulating 40 builders, players receive a special ERC20 token called Codeup (CUP) as a reward. The game combines tower-building strategy with DeFi elements within the Ethereum ecosystem.

Scope

Code Audit

CodeUP DeFi game based on an Ethereum story

Scope

Filepath	nSLOC
src/Codeup.sol	357
src/CodeupERC20.sol	14
Total	**371**

Assumptions

External libraries such as Solmate are considered safe.

Methodology

Code Audit

CodeUP DeFi game based on an Ethereum story

Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Severity / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Code Audit

CodeUP DeFi game based on an Ethereum story

Project Dashboard

Application Summary

Name	CodeUP
Commit	86f8350cde90cec9f858e5380ebead2dfb43473a
Language	Solidity
Platform	Arbitrum, Ethereum, Optimism, Base, Polygon

Engagement Summary

Timeline	23-10-2024 to 27-10-2024
№ of Auditors	2
Review Time	1 person week

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	3	3	0
High	1	1	0
Medium	1	1	0
Low	2	2	0

None	5	4	1

Category Breakdown

Suggestion	3
Documentation	0
Bug	5
Optimization	2
Good Code Practices	2

Code Maturity Evaluation

Code Audit

CodeUP DeFi game based on an Ethereum story

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system's resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Satisfactory. The codebase has a strong access control mechanism.
Arithmetic	Moderate. The codebase uses Solidity version >0.8.0, but some rounding issues were found.
Centralization	Satisfactory. The protocol has no centralization points.
Code Stability	Satisfactory. The codebase was stable during the course of the audit.
Upgradeability	Weak. The price feed is not upgradeable.
Function Composition	Satisfactory. Functionality is well split into different helpers.
Front-Running	Satisfactory. No front-running issues were found.
Monitoring	Satisfactory. Events were correctly emitted.
Specification	Satisfactory. The code matched the design specifications.
Testing and Verification	Satisfactory. Tests were present for most functionality.

Findings

Code Audit

CodeUP DeFi game based on an Ethereum story

Findings

3S-CodeUP-C01

`Codeup::claimCodeupERC20()` may be forever DoSed by creating the **`Uniswap`** pool before it is called

Id	3S-CodeUP-C01
Classification	Critical
Severity	Critical
Likelihood	High
Category	Bug
Status	Addressed in #4864ad9.

Description

`Uniswap` pools may be created without the underlying tokens existence, which means that someone may do this before **`Codeup::claimCodeupERC20()`** is ever called, making it revert when **`UniswapV2Factory::createPair()`** is called as the pool has already been created.

Recommendation

`UniswapV2Router::addLiquidity()` creates the pool if it does not yet exist, so there is not need to directly create it. The only concern is setting the **`uniswapV2Pool`** variable, which may be performed for example by doing:

```
...
if (uniswapV2Pool == address(0)) {
    ... // added liquidity
    uniswapV2Pool = IUniswapV2Factory(uniswapV2Factory).getPair(
        wethMemory,
        codeupERC20Memory
    );
    emit PoolCreated(uniswapV2Pool);
...
```

3S-CodeUP-C02

`tower.gameETHForWithdraw` should be reduced pro-rata when there is not enough **`ETH`**

Id	3S-CodeUP-C02
Classification	Critical
Severity	Critical
Likelihood	High
Category	Bug
Status	Addressed in #31359b8.

Description

`Codeup::withdraw()` and **`Codeup::reinvest()`** cap the withdrawn amount to the available **`ETH`** balance, but always set the **`tower.gameETHForWithdraw`** to **`0`**, which means that a user can have a significant amont of **`gameETH`** for withdraw, but due to the contract having low **`ETH`** balance, most of these funds could be lost.

This is relevant because blockchains often have concurrency issues, which means, for example, 2 users trying to withdraw more than the contract balance at the same time could lead to one of them getting much less **`ETH`** than expected.

Recommendation

When there is not enough **`ETH`**,**`tower.gameETHForWithdraw`** should be reduced by the **`amount`** before subtracting the commission divided by the price, rounded up.

3S-CodeUP-C03

Total **ETH**, **WETH** and **gameETH** are not tracked which will lead to insolvency

Id	3S-CodeUP-C03
Classification	Critical
Severity	Critical
Likelihood	High
Category	Bug
Status	Addressed in #6020c6f.

Description

Users may upgrade their towers so they receive **gameETH** over time which can later be withdrawn for **ETH**. However, there is no actual tracking in either of the funds, which means that eventually once users stop entering the protocol, there will not be enough ETH to pay everyone for their yield. A poc was created showing that after just **16** hours a user receives more **ETH** as yield than their initial deposit when buying **gameETH**.

Thus, initial users that upgrade their towers will just get all yield very quickly from new users, eventually making the protocol insolvent as everyone has yield to claim but nobody wants to deposit.

Another related concern is that users have no option to withdraw their **gameETH**, even if they have for example upgraded their towers to the max level. It does not make sense to ever deposit more than the max cost to upgrade all builders.

Lastly, the remaining **weth** balance will be impossible to withdraw because the last user that has upgraded their towers to the max builders will withdraw these funds and pay a commision to the pool in **amountForPool**, which will never be recovered.

Recommendation

Users not being able to always get their yield due to the contract not having enough eth is a design choice, as users need to be quick to claim their yield. However, it never makes sense to add more game eth than the amount left needed to fully upgrade their tower, so a check could be added. Additionally, a way to add as liquidity the last weth from commisions should be added.

3S-CodeUP-H01

Significant rounding errors due to **gameETH** not having precision

Id	3S-CodeUP-H01
Classification	High
Severity	High
Likelihood	Medium
Category	Bug
Status	Addressed in #a5bd528.

Description

In **Codeup::addGameETH()** and **Codeup::reinvest()**, **gameETH** is obtained as **gameETH = amount / gameETHPrice**, which means there may be rounding errors. **gameETH** has no decimals as can be confirmed by the return values of **Codeup::_getUpgradePrice()** and **Codeup::_getYield()**, which have no extra precision.

Thus, as **gameETHPrice** may be a big value - currently **1e12** in the tests, the rounding error may be up to **1e12 - 1**, which is approx **0.0023 USD**. If the price is increased, the rounding error grow to bigger values causing significant loss of funds for users.

Recommendation

Add some precision to **gameETH** by scaling the values in the functions **Codeup::_getUpgradePrice()** and **Codeup::_getYield()** and decreasing the **gameETHPrice** so **gameETH** inherits the decimals of **ETH**.

3S-CodeUP-M01

`Codeup::claimCodeupERC20()` may revert whenever the `weth` balance is very low

Id	3S-CodeUP-M01
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Bug
Status	Addressed in #389b655.

Description

`Codeup::claimCodeupERC20()` adds liquidity to the `Uniswap` pool whenever the `weth` balance is bigger than `1`. However, an amount bigger than `1` may still lead to reverts if it is low enough. If it is exactly 1, it will shift right to get the amount of `weth` to swap for `CodeupERC20`, trying to swap an amount of `0` and reverting. If it is bigger than 2, but still low, it may swap this for an even smaller amount of `CodeupERC20`, reverting when adding liquidity due to not providing enough liquidity to mint a single share.

A poc is available to confirm the finding.

Recommendation

Instead of setting `1`, a slightly bigger dust amount could be use to ensure it does not revert.

3S-CodeUP-L01

`Codeup::claimCodeupERC20()` is vulnerable to sandwich attacks

Id	3S-CodeUP-L01
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Addressed in #4168f79.

Description

`Codeup::claimCodeupERC20()` does not set minimum values for adding liquidity or swapping (sets 0) and places a deadline of `block.timestamp`, which means mev bots may sandwich these calls for profit for the protocol's loss.

Recommendation

Send the minimum amounts as an argument as well as the deadline.

3S-CodeUP-L02

Missing key zero value check

Id	3S-CodeUP-L02
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Addressed in #f32b6f5.

Description

`gameETHForWithdrawRate` is set in **`Codeup::constructor()`** to **`_gameETHPrice / 1000`** but a zero check is missing.

Recommendation

Add the **`_checkValue()`** function to **`gameETHForWithdrawRate = _checkValue(_gameETHPrice / 1000)`**.

3S-CodeUP-N01

Unused **`TransferFailed`** error in **`CodeupERC20`**

Id	3S-CodeUP-N01
Classification	None
Category	Optimization
Status	Addressed in #094b944.

Description

The **`TransferFailed`** error in **`CodeupERC20`** is not used and can be removed.

Recommendation

Remove this error.

3S-CodeUP-N02

`Ownable` is unused in **`CodeupERC20`** and could be removed

Id	3S-CodeUP-N02
Classification	None
Category	Optimization
Status	Acknowledged

Description

`CodeupERC20` inherits **`Ownable`** but does not use any of its functionalities.

Recommendation

Remove **`Ownable`** as it is not used.

3S-CodeUP-N03

All functions could have a **`started`** modifier as they should not be called before the start time

Id	3S-CodeUP-N03
Classification	None
Category	Suggestion
Status	Addressed in #b50f821.

Description

Only **`Codeup::addGameETH()`** checks if the protocol has started in **`require(block.timestamp > startUNIX, NotStarted());`**, but the same checks could be applied to the remaining functions. Although this presents no vulnerability as these functions will not do anything before adding **`gameETH`**, it is a good practice to prevent unwanted behaviours.

Recommendation

Create a modifier having **`require(block.timestamp > startUNIX, NotStarted());`** and place it in all functions.

3S-CodeUP-N04

Hardcoded values present in the codebase

Id	3S-CodeUP-N04
Classification	None
Category	Good Code Practices
Status	Addressed in #9734cd9.

Description

Hardcoded values should be avoided for better readability.

Recommendation

The following instances were found: `(tokenAmount * 10) / 100` (1, 2), Codeup::_syncTower(), Codeup::_getUpgradePrice(), Codeup::_getYield()

3S-CodeUP-N05

Typos in the codebase

Id	3S-CodeUP-N05
Classification	None
Category	Good Code Practices
Status	Addressed in #8f66046.

Description

There are typos in the codebase that could be fixed.

Recommendation

The following instances were found: availble, spend, luqidity, spended, **`emitted`** (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11), hrs (should be **`mins`**, 1, 2), totalInvestedBedore.