Three Sigma

Code Audit

LAYER3

Layer3 Button On-chain Game

Disclaimer

Code Audit

Layer3 Button On-chain Game

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

Layer3 Button On-chain Game

Table of Contents

Summary

Code Audit

Layer3 Button On-chain Game

Summary

Three Sigma audited Layer3 in a 0.4 person week engagement. The audit was conducted on 29/04/2025.

Protocol Description

The Button contract is an interactive on-chain game where players compete by pressing the button for a chance to win the prize pool. Each press resets the countdown timer and shortens its duration, increasing urgency with every interaction until a defined minimum is reached. When the timer runs out, the last player to have pressed wins the entire pool. The contract supports flexible configuration for fees, timing, and rewards—including fixed or growing prize pools—along with referral incentives, pause controls, and emergency withdrawals. It uses OpenZeppelin libraries for security, access control, and token handling.

Scope

Code Audit

Layer3 Button On-chain Game

Scope

Filepath	nSLOC
src/button/Button.sol	225
Total	**225**

Methodology

Code Audit

Layer3 Button On-chain Game

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 classify findings based on Immunefi's Vulnerability Severity Classification System (v2.3) as a guideline. The final classification considers both the potential impact of an issue, as defined in the referenced system, and its likelihood of being exploited. The following table summarizes the general expected classification according to impact and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

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

Project Dashboard

Code Audit

Layer3 Button On-chain Game

Project Dashboard

Application Summary

Name	Layer3
Repository	https://github.com/layer3xyz/play
Commit	ea3299469c108c5621d5e107e08e9962f87ca2c3
Language	Solidity
Platform	Base

Engagement Summary

Timeline	29/04/2025
№ of Auditors	2
Review Time	0.4 person weeks

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	0	0	0
High	1	1	0
Medium	1	0	1
Low	3	3	0
None	2	2	0

Category Breakdown

Suggestion	3
Documentation	0
Bug	4
Optimization	0
Good Code Practices	0

Risk Section

Code Audit

Layer3 Button On-chain Game

Risk Section

- **Administrative Key Management Risks**: The system makes use of administrative keys to manage critical operations. Compromise or misuse of these keys could result in unauthorized actions and financial losses.

Findings

Code Audit

Layer3 Button On-chain Game

Findings

3S-Layer3-H01

gameEndTime is incorrectly updated during unpause

Id	3S-Layer3-H01
Classification	High
Impact	High
Likelihood	Medium
Category	Bug
Status	Addressed in #5585c17.

Description

The pause/unpause mechanism fails to adjust the game end time to account for elapsed time during pauses. When unpausing, the contract calculates **`gameEndTime`** using **`pausedAt + countdown`** instead of considering the actual pause duration. If a game is paused long enough, it will end immediately upon unpausing, declaring the last player before pause as the winner without giving other players the chance to participate.

Consider the following scenario:

1. Game starts at timestamp 1000 with **`gameStartTime = 1000`** and **`countdown = 300 seconds`**

2. Initial **`gameEndTime = 1300`**

3. At timestamp 1005, Player A presses the button

- **`countdown`** decreases to 270 seconds (after 30-second decrement)
- **`gameEndTime`** is updated to **`1005 + 270 = 1275`**
- **`lastPlayer = Player A`**

4. At timestamp 1010, the game is paused for maintenance

- **pausedAt = 1005** (timestamp of last press)

5. The game remains paused for 500 seconds until timestamp 1510

6. At timestamp 1510, the game is unpaused

- **gameEndTime** is incorrectly set to **pausedAt + countdown = 1005 + 270 = 1275**

7. Since timestamp 1510 > gameEndTime 1275, the game immediately ends

8. Player A is declared the winner without other players having a chance to compete

---

Recommendation

It is suggested to implement the following changes:

```
    function pause() external override onlyRole(PAUSER_ROLE) {
        uint40 lastPeriod = lastPress != 0 ? lastPress : gameStartTime;
-       pausedAt = lastPeriod;
+       duration = gameEndTime - block.timestamp;
        _pause();
    }
    /// @inheritdoc IButton
    function unpause() external override onlyRole(PAUSER_ROLE) {
-       gameEndTime = pausedAt + countdown;
-       pausedAt = 0;
+       gameEndTime = block.timestamp + duration;
+       duration  = 0;
        _unpause();
    }
```

3S-Layer3-M01

Game design is vulnerable to block stuffing attacks

Id	3S-Layer3-M01
Classification	Medium
Impact	High
Likelihood	Low
Category	Bug
Status	Acknowledged

Description

A malicious actor (miner or player) can execute a **block stuffing attack** to gain an unfair advantage in the game by preventing other users from interacting with the contract.

Attack Scenario:

1. The game starts at **`t = 0`** with a pot of **$1000**.

2. Countdown = 300 seconds , decrement = 30 seconds and minimum countdown = 60 seconds.

3. The attacker alternates button presses between two controlled addresses (e.g., Player A and Player B), reducing the countdown to 60 seconds (countdownMinimum).

4. Once the countdown is at 60 seconds, the attacker initiates a **block stuffing attack** to fill all blocks with high-gas transactions for the next 60 seconds.

5. As a result, no other players can interact with the contract, and the attacker becomes the winner by default.

Block stuffing effectively censors legitimate interactions and ensures only the attacker's transactions are included.

Cost Consideration:

On Optimism, the cost to stuff a full block for one minute is currently around **$169** (based on a gas price of 5.46 Gwei and block gas limit of 35,000,000). Thus, the **attack becomes profitable when the potential reward (pot size) exceeds the attack cost**.

References:

- The Anatomy of a Block Stuffing Attack

Recommendation

As game can only be ended by a trusted user, this won't be a problem for now. But might be a problem when prize claiming process will also become permission-less.

3S-Layer3-L01

No check to prevent sponsorships after game is over

Id	3S-Layer3-L01
Classification	Low
Impact	Low
Likelihood	Medium
Category	Bug
Status	Addressed in #4c39cdf.

Description

The **`sponsor()`** function currently does not check whether the game has ended before accepting sponsorships. As a result, users can continue to send funds to the game contract even after the game is over, which is not intended.

Recommendation

Add a check at the beginning of the **`sponsor()`** function to prevent sponsorships after the game has ended:

```
    function sponsorPot(uint256 amount) public override {
+       require(gameEndTime == 0 || !isGameOver(), GameIsOver());
        IERC20(PAYMENT_TOKEN).safeTransferFrom(msg.sender, address(this),
amount);
        totalPot += amount;
    }
```

3S-Layer3-L02

Zero transfers might revert for some ERC20 tokens

Id	3S-Layer3-L02
Classification	Low
Impact	Medium
Likelihood	Low
Category	Bug
Status	Addressed in #7c9d632.

Description

The contract performs a token transfer even when the amount is zero. Some ERC20 tokens do not allow zero-value transfers and may revert, potentially breaking the flow.

Specifically, when **`referralFee`** is set to 0, **`_referralFee`** becomes zero, resulting in a zero-amount transfer to the **`referrer`**, which could cause issues with non-compliant tokens.

Recommendation

Add a check to skip the referral transfer if **`_referralFee`** is zero to if compatibility with all tokens is necessary.

3S-Layer3-L03

Low fee precision prevents fractional percentage configuration

Id	3S-Layer3-L03
Classification	Low
Impact	Low
Likelihood	Medium
Category	Suggestion
Status	Addressed in #bef86b5.

Description

The **`protocolFee`** and **`referralFee`** variables use low-precision integer percentages, which restricts fee configuration to whole numbers like 1% or 2%, and prevents using more accurate values like 1.5% or 2.5%.

Recommendation

Use a higher-precision system (e.g., 1e4 = 10000 = 100.00%) to support fractional percentages and allow more flexible fee settings.

3S-Layer3-N01

`collectDust` function allows sweeping the payment token

Id	3S-Layer3-N01
Classification	None
Category	Suggestion
Status	Addressed in #bcd3ff6.

Description

The **`collectDust`** function does not restrict the **`token`** parameter, allowing the contract owner to sweep the **`PAYMENT_TOKEN`** as dust. This could lead to accidental removal of funds meant for the game pot.

Recommendation

Add a check to ensure that **`token != PAYMENT_TOKEN`** in the **`collectDust`** function to prevent sweeping the main payment token.

3S-Layer3-N02

Add validations for fee parameters

Id	3S-Layer3-N02
Classification	None
Category	Suggestion
Status	Addressed in #a70739c.

Description

The contract does not perform any validation on the **`protocolFee`** and **`referralFee`** values. If the fees are set to values outside the expected range (e.g., greater than 100%), it could cause incorrect calculations. There is no check in place to ensure the fee values are within acceptable limits.

Recommendation

Implement validations to ensure that **`protocolFee`** and **`referralFee`** are within a valid range (e.g., between 0 and 100%).