Latte Survival Game Audit

November 12, 2021

Table of Contents

Summary

This report has been prepared for **Survival Game** smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Overview

Project Summary

Project Name	**Survival Game**
Codebase	**https://github.com/latteswap-official/survival-game-contract**
Commit	**55255f67629b3948be71c351dd7a676e51c7ab78**
Language	**Solidity**
Platform	**BSC**

Audit Summary

Delivery Date	**Nov 12, 2021**
Audit Methodology	**Static Analysis, Manual Review**
Total Isssues	**11**

LS-1: Avoid unnecessary storage writes can save gas

Informational

Issue Description

contracts/SurvivalGame.sol#L342

```
for (uint256 i = 0; i < _remainingPlayerCount; ++i) {
    bytes memory _data = abi.encodePacked(_entropy, address(this), msg.sender, ++nonce);
    // eliminated if hash value mod 100 more than the survive percent
    bool _survived = _survivalBps > (uint256(keccak256(_data)) % 1e4);
    if (_survived) {
        ++_survivorCount;
    }
}
```

The gas cost of storage writes (SSTORE) is significant.

At L342, the storage variable `nonce` is being write once in each loop, and the for loop be repeated for up to 1000 times.

Create a local variable can save a lot of gas.

Recommendation

Change to:

```
for (uint256 i = 0; i < _remainingPlayerCount; ++i) {
    bytes memory _data = abi.encodePacked(_entropy, address(this), msg.sender, nonce+i+1);
    // eliminated if hash value mod 100 more than the survive percent
    bool _survived = _survivalBps > (uint256(keccak256(_data)) % 1e4);
    if (_survived) {
        ++_survivorCount;
    }
}
nonce += _remainingPlayerCount;
```

Status

✓ Fixed in commit: 752deaa94a8109cfcf6f0be4b3fbf5d6a64d72b5.

LS-2: Consider adding unclaimedPrize and totalUnclaimedPrize to give winners more time to claim

Recommendation

Issue Description

In the current design, winners must `claim` before next game starts. Otherwise, the prizes will be returned to the prize pool.

This makes the game must wait a fair amount of time before starting the next game. If the claim period is too short, when the network is congested, many users may lose their prize.

To solve this problem, we suggest:

- Add two variables, `gameInfo.unclaimedPrize` and `totalUnclaimedPrize`;
- Change `prizePoolInLatte()` to:

```
function prizePoolInLatte() public view returns (uint256) {
    uint256 _balance = IERC20Upgradeable(latte).balanceOf(address(this))
    if (totalUnclaimedPrize > _balance) {
        return 0;
    }
    return _balance - totalUnclaimedPrize;
  }
```

- Set `gameInfo.unclaimedPrize` and `totalUnclaimedPrize` in `_complete()`;
- Add parameter `gameId` to `claim()` and update `gameInfo[gameId].unclaimedPrize` and `totalUnclaimedPrize` in `claim()`;

If the claim period is designed to be limited, and the unclaimed prizes are supposed to be returned to the prize pool, a limit can still be added to the `claim()` function and we can add a new function named `returnUnclaimedPrize()` to return the unclaimed prizes.

With these changes, the claim period can unbundle from the interval of games, allowing games to run faster one after another.

Status

ⓘ Acknowledged

LS-3: Redundant code in initialize()

Minor

Issue Description

contracts/SurvivalGame.sol#L149-L151

```
gameId = 0;
nonce = 0;
lastUpdatedBlock = 0;
```

Setting `uint` variables to `0` is redundant as they default to `0`.

Recommendation

Consider removing the above code or change to:

```
gameId = 1;
nonce = 1;
lastUpdatedBlock = block.timestamp;
```

This will lower the gas cost for the first user.

Status

✓ **Fixed** in commit: ac4fd619636b2d7df1f60c68719b2f0b3ddb586a.

LS-4: Misleading variable names

Informational

Issue Description

contracts/SurvivalGame.sol#L159-L167

```
/// @dev only the one having a OPERATOR_ROLE can continue an execution
  modifier onlyOper() {
    require(hasRole(OPERATOR_ROLE, _msgSender()), "SurvivalGame::onlyOper::only OPERATOR role");
    require(
      uint256(block.timestamp) - lastUpdatedBlock >= operatorCooldown,
      "SurvivalGame::onlyOper::OPERATOR should not proceed the game consecutively"
    );
    _;
  }
```

`lastUpdatedBlock` is actual a timestamp.

Consider renaming to `lastUpdatedTimestamp`.

Status

✓ Fixed in commit: ac4fd619636b2d7df1f60c68719b2f0b3ddb586a.

LS-5: Misleading comments

Informational

Issue Description

contracts/SurvivalGame.sol#L40-L41

```
// Minimum required blocks before operator can execute function again
uint256 public operatorCooldown;
```

`operatorCooldown` is in seconds, but the comment says it's in blocks.

contracts/SurvivalGame.sol#L343-L344

```
// eliminated if hash value mod 100 more than the survive percent
bool _survived = _survivalBps > (uint256(keccak256(_data)) % 1e4);
```

Should be `mod 10000`.

Status

✓ Fixed in commit: ac4fd619636b2d7df1f60c68719b2f0b3ddb586a.

LS-6: Ownable is redundant

Minor

Issue Description

contracts/SurvivalGame.sol#L25-L25

The `SurvivalGame.sol` contract is already using `AccessControl`, and `onlyOwner` is not being used.

Therefore, `Ownable` can be removed to make the code simpler and save some gas.

Status

ⓘ Acknowledged

LS-7: Inconsistent use of _msgSender()

Informational

Issue Description

Direct use of `msg.sender` vs internal call of `_msgSender()`.

contracts/SurvivalGame.sol#L154-L155

```
_setupRole(DEFAULT_ADMIN_ROLE, _msgSender());
_setupRole(OPERATOR_ROLE, _msgSender());
```

contracts/SurvivalGame.sol#L170-L173

```
modifier onlyEntropyGenerator() {
    require(msg.sender == address(entropyGenerator),
    "SurvivalGame::onlyEntropyGenerator::only entropy generator");
    _;
}
```

Status

✓ Fixed in commit: ac4fd619636b2d7df1f60c68719b2f0b3ddb586a.

LS-8: start() Redundant code

Informational

Issue Description

contracts/SurvivalGame.sol#L266-L271

```
function start() external onlyOper onlyOpened {
    gameInfo[gameId].status = GameStatus.Processing;
    _requestRandomNumber();
    lastUpdatedBlock = block.timestamp;
    emit LogSetGameStatus(gameId, "Processing");
}
```

L269 is redundant as `_proceed()` will update `lastUpdatedBlock` at L434.

Status

✓ **Fixed** in commit: ac4fd619636b2d7df1f60c68719b2f0b3ddb586a.

LS-9: Cache storage variables in the stack can save gas

Informational

Issue Description

For the storage variables that will be accessed multiple times, cache them in the stack can save ~100 gas from each extra read (`SLOAD` after Berlin).

For example:

- `gameId` in `_proceed()`

  contracts/SurvivalGame.sol#L423-L435

```
function _proceed(uint256 _entropy) internal {
    uint8 _nextRoundNumber = gameInfo[gameId].roundNumber.add(1);
    roundInfo[gameId][_nextRoundNumber].entropy = _entropy;
    emit LogSetEntropy(gameId, _nextRoundNumber, _entropy);

    gameInfo[gameId].roundNumber = _nextRoundNumber;
    emit LogSetRoundNumber(gameId, _nextRoundNumber);

    gameInfo[gameId].status = GameStatus.Started;
    emit LogSetGameStatus(gameId, "Started");

    lastUpdatedBlock = block.timestamp;
}
```

- `gameId` in `check()`
- `gameId` in `buy()`

Status

✓ Fixed in commit: b1fe1f8edcba4e17cc0ad8b669a1dd172cc25be6.

LS-10: Consider adding a function to re-request randomness when VRF fulfillment failed

Medium

Issue Description

Per the Chainlink VRF document: https://docs.chain.link/docs/vrf-security-considerations/#fulfillrandomness-must-not-revert

> If your fulfillRandomness implementation reverts, the VRF service will not attempt to call it a second time.

Even though it's unlikely for the `consumeRandomNumber()` function to revert, but when it happens, or if VRF fulfillRandomness failed for other reasons, the whole game will be stuck in `Processing` status.

Therefore, we suggest adding a function to re-request randomness or force fulfillRandomness as last resort.

Status

✓ **Fixed** in commit: b1fe1f8edcba4e17cc0ad8b669a1dd172cc25be6.

LS-11: check() the mechanism to determined _survived can be gamed

High

Issue Description

contracts/SurvivalGame.sol#L324-L338

```
RoundInfo memory _roundInfo = roundInfo[_gameId][_roundNumber];
uint256 _entropy = _roundInfo.entropy;
require(_entropy != 0, "SurvivalGame::_check::no entropy");
uint256 _survivalBps = _roundInfo.survivalBps;
{
  _survivorCount = 0;
  for (uint256 i = 0; i < _remainingPlayerCount; ++i) {
    bytes memory _data = abi.encodePacked(_entropy, address(this), msg.sender, nonce + i + 1);
    // eliminated if hash value mod 10000 more than the survive bps
    bool _survived = _survivalBps > (uint256(keccak256(_data)) % 1e4);
    if (_survived) {
      ++_survivorCount;
    }
  }
  nonce += _remainingPlayerCount;
```

The global storage variable `nonce` will affect the `_survived` status of user's players.

A malicious user or an attacker can dry run the `check()` with `eth_call` to predict the result of `_survivorCount` and only to send the transaction when the `_survivorCount` is the largest.

This gives the attacker an unfair competitive edge in the game. Making it possible for the attacker to seize a large or even the majority portion of the prize pool.

Recommendation

Consider removing `nonce` and use the index instead:

```
RoundInfo memory _roundInfo = roundInfo[_gameId][_roundNumber];
uint256 _entropy = _roundInfo.entropy;
require(_entropy != 0, "SurvivalGame::_check::no entropy");
uint256 _survivalBps = _roundInfo.survivalBps;
{
  _survivorCount = 0;
  for (uint256 i = 0; i < _remainingPlayerCount; ++i) {
    bytes memory _data = abi.encodePacked(_entropy, address(this), msg.sender, i);
    // eliminated if hash value mod 10000 more than the survive bps
    bool _survived = _survivalBps > (uint256(keccak256(_data)) % 1e4);
    if (_survived) {
      ++_survivorCount;
    }
  }
```

Status

✓ Fixed in commit: 55255f67629b3948be71c351dd7a676e51c7ab78.

Appendix

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by WatchPug; however, WatchPug does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Smart Contract technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.