SALUS SECURITY

AUG 2023

CODE SECURITY ASSESSMENT

REIGN OF TERROR

Overview

Project Summary

- Name: Reign of Terror - HackerAssault
- Version: commit 22a0227
- Platform: BNB Smart Chain
- Language: Solidity
- Repository:
    - https://github.com/reddoordigital/deploy-audit-rot-contracts-evm-hackerass
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Reign of Terror - HackerAssault
Version	v3
Type	Solidity
Dates	August 25 2023
Logs	June 19 2023; August 18 2023 ; August 25 2023

Vulnerability Summary

Total High-Severity issues	2
Total Medium-Severity issues	4
Total Low-Severity issues	5
Total informational issues	8
Total	19

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Weak sources of randomness	High	Weak Randomness	Resolved
2	Too many players can lead to denial of service	High	Business Logic	Resolved
3	Can attack during escape time and get extra rewards	Medium	Business Logic	Resolved
4	Creation of boss is not guaranteed to be incremental by 1	Medium	Business Logic	Resolved
5	Should use the _addToBalance function when adding rewards to players	Medium	Business Logic	Resolved
6	Centralization risk	Medium	Centralization	Acknowledged
7	lastDefeatTimestamp is used for the first boss but will not be set	Low	Business Logic	Resolved
8	Improper _isBossActive() check in the _canAttack function	Low	Data Validation	Resolved
9	Inappropriate role privilege	Low	Configuration	Acknowledged
10	No need to authenticate the caller in getBossEscapeTime()	Low	Data Validation	Resolved
11	Implementation contract could be initialized by everyone	Low	Business Logic	Resolved
12	Use of floating pragma	Informational	Configuration	Acknowledged
13	Use safeTransfer instead of transfer	Informational	Business Logic	Resolved
14	Initializer functions with public visibility	Informational	Access Control	Resolved
15	Missing zero address checks	Informational	Data Validation	Resolved
16	Inappropriate variable name	Informational	Code Quality	Resolved
17	Redundant code and comments	Informational	Redundancy	Resolved
18	SafeMath is not required since Solidity 0.8.0	Informational	Redundancy	Resolved

19	Spelling mistakes	Informational	Code Quality	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Weak sources of randomness

Severity: High	Category: Weak Randomness
Target: - contracts/hackerass/Attack.sol	

Description

Using blockhash and block.timestamp as a source of randomness is commonly advised against, as the outcome can be manipulated by calling contracts. In this case, players can use contracts to maximize the damage, i.e. call the startAttack function when the return value of the _random function or _prng function meet their needs.

contracts/hackerass/Attack.sol:L214-L228

```
function _random(uint256 max) internal view returns (uint256) {
        bytes32 blockHash = blockhash(block.number - 1);
        bytes32 hash = keccak256(
            // solhint-disable-next-line not-rely-on-time
            abi.encodePacked(blockHash, block.timestamp, msg.sender)
        );
        return uint256(hash) % max;
}

function _prng(uint256 max) internal view returns (uint256) {
        bytes32 hash = keccak256(
            abi.encodePacked(uint256(1), block.number, msg.sender)
        );
        return uint256(hash) % max;
}
```

Proof of Concept

```
function exploit() external {
        uint rand = uint(keccak256(abi.encodePacked(blockhash(block.number - 1),
block.timestamp, address(this)))) % 100;
        require(rand > 93);
        hackerAssault.deposit(100 ether);
        hackerAssault.startAttack(Attack.AttackOption.FrameworkExploit,
Attack.AttackStrategy.Trojan);
}
```

A player can repeatedly call the exploit function until rand is greater than 93, which is the threshold that can have 1000 as the multiplier, and only then continue to call the startAttack function. In this way, the damage will surely be the maximum.

Recommendation

Consider using a decentralized oracle for the generation of random numbers, such as Chainlink VRF.

Status

This issue has been resolved by the team.

2. Too many players can lead to denial of service

Severity: High	Category: Business Logic
Target: - contracts/hackerass/RewardPool.sol	

Description

The functions _distributeBossDefeatedRewards() and _distributeEscapeRewardPool() iterates on the playerAddresses array to distribute rewards when the boss is defeated or escaped. This array can grow huge and iterating over this may lead to a large amount of gas which may be more than the block gas limit resulting in denial of service.

For example, the _distributeBossDefeatedRewards() function on average uses 2 SLOAD opcodes and 4 SSTORE opcodes. The SSTORE opcode uses 20000 gas and the SLOAD opcode uses 2100 gas. So, the total cost per player is 2 * 2100 + 4 * 20000 = ~84200. With block gas limit of 30 million, the number of players that can lead to denial of service is 30 * 1_000_000 / 84200 = ~356. With the adoption of game, this may lead to issues.

Recommendation

It is recommended to avoid all the actions executed in a single transaction. Consider splitting the reward distribution into several calls in case of facing gas limit or using a pull model where player claims their rewards instead of a push model where the protocol distributes the rewards to all users.

Status

This issue has been resolved by the team.

3. Can attack during escape time and get extra rewards

Severity: Medium	Category: Business Logic
Target: - contracts/hackerass/Boss.sol	

Description

It is possible to attack during escape time and get extra rewards.

The core of vulnerability is lying in _applyBossDamage function, especially the position of calling the _checkBossEscape() function. It is called after making an attack.

When block.timestamp is not smaller than currentBoss.escapeTime and if the PLAYER_MNGR_ROLE hasn't called checkBossEscape(), players can perform an attack to make more damage or even defeat the boss.

Recommendation

Consider calling _checkBossEscape before performing the attack.

Status

This issue has been resolved by the team.

4. Creation of boss is not guaranteed to be incremental by 1

Severity: Medium	Category: Business Logic
Target: - contracts/hackerass/Boss.sol	

Description

PLAYER_MNGR_ROLE can spawn new bosses with their desired level using the createNewBoss function. So, the level of the new boss spawned is not guaranteed to be one more than the level of old boss. However, the bossHealth increases linearly in the _spawnBoss function. This means that if the first boss is not at level 1, the bossHealth of the subsequent bosses will be zero.

contracts/hackerass/Boss.sol:L129-L135

```
if (level == 1) {
     bossHealth = bossLevel1HP;
     escapeTime = timestamp + bossEscapeTime;
} else {
     bossHealth = currentMaxHealth + currentMaxHealth.mul(12).div(100);
     escapeTime = timestamp + bossDefeatCooldown + bossEscapeTime;
}
```

This can also lead to non-consecutive claiming of lagging rewards by the player when boss is defeated in the _distributeBossDefeatedRewards function. For example, if the lagging rewards array consists of lagging rewards for a boss with level [2,3,4] and PLAYER_MNGR_ROLE spawns a boss with level 6, then the player will claim lagging rewards corresponding to the boss with level 3 instead of level 2.

Recommendation

Consider restricting PLAYER_MNGR_ROLE to spawn a boss with level of one more than the previous boss.

Status

This issue has been resolved by the team. The bossHealth for a specific level will be calculated via the calculateMaxHealth function.

5. Should use the _addToBalance function when adding rewards to players

Severity: Medium	Category: Business Logic
Target: - contracts/hackerass/RewardPool.sol	

Description

When adding rewards to players, the _addToBalance function should be used to update the value of totalUsersFunds synchronously.

contracts/hackerass/RewardPool.sol:L214-L219

```
if (damage > 0) {
    // Calculate the reward from the reward pool based on damage contribution
    reward = poolPrize.mul(damage).div(totalBossDamage);
    // Transfer the reward to the player's accomulated rewards
    players[player].balance += reward;
}
```

Recommendation

Consider changing `players[player].balance += reward` to `_addToBalance(player, reward)`.

Status

This issue has been resolved by the team.

6. Centralization risk

Severity: Medium	Category: Centralization
Target: - contracts/hackerass/HackerAssault.sol	

Description

There are some privileged roles in the HackerAssault contract, and they are all assigned to the contract deployer. The UPGRADER_ROLE can upgrade the contract logic. The LOGIC_MNGR_ROLE and PLAYER_MNGR_ROLE can control the game and update settings.

If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the other users.

Recommendation

Consider transferring the privileged roles to multi-sig accounts.

Status

This issue has been acknowledged by the team.

7. lastDefeatTimestamp is used for the first boss but will not be set

Severity: Low	Category: Business Logic
Target: - contracts/hackerass/Boss.sol - contracts/hackerass/Player.sol	

Description

lastDefeatTimestamp is used in the _canAttack and _startAttack functions to ensure that attack is made after the cooldown period when the last boss is defeated. But for the first boss, lastDefeatTimestamp won't be set and condition `block.timestamp >= lastDefeatTimestamp + bossDefeatCooldown` will always hold true.

Recommendation

Consider setting a default value to lastDefeatTimestamp so that the first boss is attacked after the desired time interval.

Status

This issue has been resolved by the team.

8. Improper _isBossActive() check in the _canAttack function

Severity: Low	Category: Data Validation

Target:
- contracts/hackerass/Player.sol

Description

Players can quickly determine whether they can initiate an attack through the _canAttack function. The _canAttack function includes an _isBossActive() check, which causes the function to return false if the boss is not active.

contracts/hackerass/Player.sol:L106-L115

```
function _canAttack() internal view returns (bool) {
        return
        _isPlayerRegistered(msg.sender) &&
        block.timestamp >= // solhint-disable-line not-rely-on-time
        players[msg.sender].lastAttack + attackCooldownTime &&
        !_hasPlayerReachedDamageLimit() &&
        _isBossActive() &&
        // solhint-disable-next-line not-rely-on-time
        block.timestamp >= lastDefeatTimestamp + bossDefeatCooldown;
}
```

However, if the boss is created using _createNextBoss(), the state of the boss will remain inactive until the first attack.

contracts/hackerass/Player.sol:

```
// first atacker activates the boss
if (currentBoss.state == BossState.Inactive) {
    currentBoss.state = BossState.Active;
}
```

Recommendation

Consider changing `_isBossActive()` to `!_isBossEscaped()`.

Status

This issue has been resolved by the team.

9. Inappropriate role privilege

Severity: Low	Category: Configuration

Target:
- contracts/hackerass/HackerAssault.sol

Description

There are six roles in the HackerAssault contract. However, BOSS_MNGR_ROLE, REFERRAL_MNGR_ROLE and REWARD_POOL_MNGR_ROLE are not used throughout the codebase. Meanwhile, the PLAYER_MNGR_ROLE has the privilege to call functions such as setBossRewardDistribution(), createNewBoss(), etc.

contracts/hackerass/HackerAssault.sol:L128-L135

```
bytes32 public constant UPGRADER_ROLE = keccak256("UPGRADER_ROLE");
bytes32 public constant LOGIC_MNGR_ROLE = keccak256("LOGIC_MNGR_ROLE");
bytes32 public constant PLAYER_MNGR_ROLE = keccak256("PLAYER_MNGR_ROLE");
bytes32 public constant BOSS_MNGR_ROLE = keccak256("BOSS_MNGR_ROLE");
bytes32 public constant REFERRAL_MNGR_ROLE =
       keccak256("REFERRAL_MNGR_ROLE");
bytes32 public constant REWARD_POOL_MNGR_ROLE =
       keccak256("REWARD_POOL_MNGR_ROLE");
```

Recommendation

Please reconsider the scope of privileges for each role.

Status

This issue has been acknowledged by the team.

10. No need to authenticate the caller in getBossEscapeTime()

Severity: Low	Category: Data Validation

Target:
- contracts/hackerass/HackerAssault.sol

Description

There is no need to authenticate the caller in the getBossEscapeTime function.

contracts/hackerass/HackerAssault.sol:L349-L356

```
function getBossEscapeTime()
        external
        view
        onlyRole(PLAYER_MNGR_ROLE)
        returns (uint256)
{
        return _getBossEscapeTime();
}
```

Recommendation

Consider removing `onlyRole(PLAYER_MNGR_ROLE)`.

Status

This issue has been resolved by the team.

11. Implementation contract could be initialized by everyone

Severity: Low	Category: Business Logic
Target: - contracts/hackerass/HackerAssault.sol	

Description

The implementation contract should not be left uninitialized. An uninitialized implementation contract can be taken over by an attacker, which may impact the proxy.

Despite the fact that developers leave comments about it, it is necessary to disable initializers for every implementation.

contracts/hackerass/HackerAssault.sol:L149-L152

```
// /// @custom:oz-upgrades-unsafe-allow constructor
// constructor() {
//     _disableInitializers();
// }
```

Recommendation

It is recommended to uncomment the constructor.

Status

This issue has been resolved by the team.

2.3 Informational Findings

12. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - All	

Description

```
pragma solidity ^0.8.0;
```

The Reign of Terror codebase uses a floating compiler version ^0.8.0.

Using a floating pragma ^0.8.0 statement is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

Status

This issue has been acknowledged by the team.

13. Use safeTransfer instead of transfer

Severity: Informational	Category: Business Logic
Target: - contracts/hackerass/Player.sol	

Description

It is recommended to use safeTransfer when transferring tokens, as it will check the return value.

Recommendation

Consider using the SafeERC20Upgradeable library.

Status

This issue has been resolved by the team.

14. Initializer functions with public visibility

Severity: Informational	Category: Access Control

Target:
- contracts/hackerass/Attack.sol
- contracts/hackerass/RewardPool.sol
- contracts/hackerass/Boss.sol
- contracts/hackerass/Player.sol

Description

All initializer functions have public visibility instead of internal.

contracts/hackerass/RewardPool.sol:L52-L54

```
function __RewardPool_init(
    IERC20Upgradeable _reignToken
) public virtual onlyInitializing
```

contracts/hackerass/Boss.sol:L75-L77

```
function __Boss_init(
    IERC20Upgradeable _reignToken
) public virtual onlyInitializing
```

contracts/hackerass/Attack.sol:L55-L57

```
function __Attack_init(
    IERC20Upgradeable _reignToken
) public virtual onlyInitializi
```

contracts/hackerass/Player.sol:L41-L44

```
function __Player_init(
    IERC20Upgradeable reignToken,
    address treasuryAddress
) public virtual onlyInitializing
```

Recommendation

Consider using internal instead of public.

Status

This issue has been resolved by the team.

15. Missing zero address checks

Severity: Informational	Category: Data Validation
Target: - contracts/hackerass/HackerAssault.sol	

Description

It is considered a security best practice to verify addresses against the zero address during initialization or setting.

The zero address check is missing for _reignToken and treasuryAddress variables in the initialize function and treasury variable in the setTreasuryAddress function.

Recommendation

Consider adding zero address checks for the above mentioned variables.

Status

This issue has been resolved by the team.

16. Inappropriate variable name

Severity: Informational	Category: Code Quality
Target: - contracts/hackerass/Player.sol	

Description

The _setTreasuryAddress function uses the name `amount` for an address type, which is inappropriate.

contracts/hackerass/Player.sol:L75-L77

```
function _setTreasuryAddress(address amount) internal {
        treasury = amount;
}
```

Recommendation

Consider changing `amount` to `_treasury`.

Status

This issue has been resolved by the team.

17. Redundant code and comments

Severity: Informational	Category: Redundancy

Target:
- All

Description

Since the owner of the contracts is not used, the inheritance of OwnableUpgradeable can be removed.

contracts/hackerass/Player.sol:L50-L61

```
function _deposit(uint256 amount) internal {
        if (!_isPlayerRegistered(msg.sender)) {
            _registerPlayer(msg.sender);
        }
        require(_isPlayerRegistered(msg.sender), MESSAGE_PLAYER_NOT_REGISTERED);

        ...
}
```

If a player is not registered, he will then be registered through the _registerPlayer function. Thus, the line highlighted above is redundant.

contracts/hackerass/Attack.sol:L150-L173

```
function _attack(
        address player,
        AttackOption attackOption,
        AttackStrategy strategy
) internal returns (uint256) {
        require(
            attackOption >= AttackOption.FrameworkExploit &&
               attackOption < AttackOption.Last,
            MESSAGE_INVALID_ATTACK_OPTION
        );
        ...
}
```

The _attack function is only used in the _startAttack function. The lines highlighted above have been checked in the _startAttack function.

contracts/hackerass/Attack.sol:L67-L74

```
function _setAttackCosts(uint32[3] memory costs) internal {
    require(costs.length == 3, MESSAGE_WRONG_ARRAY_SIZE);
    ...
}
```

contracts/hackerass/Attack.sol:L95-L101

```
function _setAttackProbabilities(
    AttackStrategy strategy,
    uint8[4] memory probabilities
) internal {
    require(probabilities.length == 4, MESSAGE_WRONG_ARRAY_SIZE);
    ...
}
```

contracts/hackerass/Attack.sol:L113-L119

```
function _setDamageMultipliers(
    AttackStrategy strategy,
    uint32[3] memory multipliers
) internal {
    require(multipliers.length == 3, MESSAGE_WRONG_ARRAY_SIZE);
    ...
}
```

There is no need to check the length of fixed-size arrays.

contracts/hackerass/RewardPool.sol:L9
contracts/hackerass/Boss.sol:L5
contracts/hackerass/HackerAssault.sol:L10
contracts/hackerass/Attack.sol:L8
contracts/hackerass/Player.sol:L6

```
// import "hardhat/console.sol";
```

contracts/hackerass/RewardPool.sol:L183,L186,L190

```
// console.log("LAGGING!!!", laggingReward);
...
// console.log("Accum Reward2: ", reward);
...
// console.log("Balance: ", players[player].balance);
```

contracts/hackerass/Attack.sol:L60,L201

```
// console.log("attack owner: ", owner());
...
//console.log("damageType:", damageType);
```

The above comment-out code should be removed before the deployment of the contracts.

Recommendation

Consider removing the unnecessary code.

Status

This issue has been resolved by the team.

18. SafeMath is not required since Solidity 0.8.0

Severity: Informational	Category: Redundancy

Target:
- contracts/hackerass/Attack.sol
- contracts/hackerass/RewardPool.sol
- contracts/hackerass/Boss.sol
- contracts/hackerass/Player.sol

Description

The contracts use SafeMathUpgradeable library for mathematical operations. SafeMath is generally used to avoid overflow and underflow. But there are inbuilt overflow and underflow operations from solidity version 0.8.0. So, the use of SafeMathUpgradeable library is redundant.

Recommendation

It is recommended to remove the SafeMathUpgradeable library.

Status

This issue has been resolved by the team.

19. Spelling mistakes

Severity: Informational	Category: Code Quality

Target:
- All

Description

There are a lot of typos in the code comments and function names that need to be corrected.

contracts/hackerass/HackerAssault.sol:L343

```
function setBossEspapeTime
```

Es**p**ape should be Es**c**ape.

contracts/hackerass/Attack.sol:L18-L23

```
enum AttackOption {
        FrameworkExploit,
        NetworkScaner,
        PasswordCracker,
        Last
}
```

Scaner should be Sca**n**ner.

contracts/hackerass/Player.sol:L146

```
// first atacker activates the boss
```

atacker should be a**t**tacker.

contracts/hackerass/RewardPool.sol:L217

```
// Transfer the reward to the player's accomulated rewards
```

acc**o**mulated should be acc**u**mulated.

contracts/hackerass/Boss.sol:L192-L196

```
// if boss escapes player gets rewded with the last damage
// so it pays for the escape but gets something
// (maybe we can give him 10% more or something)
// The owner of the contract could also call this function from outside to
// triger the scape
```

rewded should be rewarded, triger should be tri**g**ger and scape should be escape.

Recommendation

Consider fixing the typos.

Status

This issue has been resolved by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 22a0227:

File	SHA-1 hash
contracts/hackerass/Attack.sol	de4e335b03791b2e2bf76e58101366191a5c39e9
contracts/hackerass/Boss.sol	a0fd3bd8308270f40bb431803a689cb7f38188f4
contracts/hackerass/HackerAssault.sol	68ae74ca9b52ba36720ed4b7c573c36be4cc0aa6
contracts/hackerass/Player.sol	6939fcd3d5ea1d70e451370769165f6a602c9e89
contracts/hackerass/RewardPool.sol	ea1e1953f59d2ae65abe63ce8cb17b8a4b5cfd49