SALUS SECURITY

MAR 2025

CODE SECURITY ASSESSMENT

UPTOP

Overview

Project Summary

- Name: UpTop
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
  - https://github.com/dsoftgames/UpTopContract
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	UpTop
Version	v2
Type	Solidity
Dates	Mar 28 2025
Logs	Mar 26 2025; Mar 28 2025

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	2
Total Low-Severity issues	5
Total informational issues	3
Total	11

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	Dos attack	High	Denial of Service	Resolved
2	Predictable hash values lead to controllable blind box rewards	Medium	Business Logic	Resolved
3	Incorrect total quantities were used to calculate rewards	Medium	Business Logic	Resolved
4	Precision loss	Low	Numerics	Resolved
5	The excess stockFee was not refunded	Low	Business Logic	Resolved
6	Mismatch with comment	Low	Code Quality	Resolved
7	Centralization risk	Low	Centralization	Acknowledged
8	Missing events for functions that change critical state	Low	Logging	Resolved
9	Redundant Code	Informational	Redundancy	Partially resolved
10	Missing two-step transfer ownership pattern	Informational	Business Logic	Acknowledged
11	Use of floating pragma	Informational	Configuration	Acknowledged

2.2 Notable Findings

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

1. Dos attack

Severity: High	Category: Denial of Service
Target: - contracts/Foo.sol	

Description

The `buildFloor()` function can add an `inviter` to the latest floor. When the floor's holdings reach the maximum, it will automatically distribute rewards to the `inviter`. If the reward is in native tokens (i.e., within the `Foo` contract), it will be directly transferred to the inviter's address via the `transfer` function. However, the contract does not verify whether the `inviter` is an EOA rather than a contract. If the address is a contract that rejects `ETH` transfers, the floor will never reach maximum holdings, preventing the creation of new floors and causing the system to stall.

contracts/Foo.sol:L220 - L228

```
for (uint256 i = 0; i < floors[index - 1].invitees.length; i++) {
    Invitee memory invitee = floors[index - 1].invitees[i];
    if (invitee.addr != address(0)) {
        // 邀请人奖励
        uint256 inviterReward = invitee.balance;
        remaining -= inviterReward;
        payable(invitee.addr).transfer(inviterReward);
    }
}
```

An attacker may have two possible motives:

1. Launch a DoS attack to freeze the system and then demand payment from the project team to restore normal operations.
2. Exploit the DoS attack to halt floor updates, ensuring they control the last two floors—one reaching maximum holdings (as the valid last floor) and another used to stall the system—so that the attacker's designated partner receives rewards, ultimately draining the entire finalBonusPool.

contracts/Foo.sol:L343 - L353

```
uint256 finalBonusRemaining = finalBonusPool;
for (uint256 i = 0; i < finalFloor.partners.length; i++) {
    uint256 bonus = finalBonusPool * finalFloor.partners[i].stock / finalFloor.stock;
    // 如果是最后一个合伙人，分完剩余的奖金
    if (i == finalFloor.partners.length - 1) {
        bonus = finalBonusRemaining;
    }
    blindBoxBalance[finalFloor.partners[i].addr] += bonus;
    finalBonusRemaining -= bonus;
    emit FloorReward(finalFloorIndex, finalFloor.partners[i].addr, bonus, 1,
finalFloorIndex);
}
```

The `settle()` function has a similar vulnerability: whether the final floor can be successfully refunded depends on the partner's cooperation. If a malicious partner (an attacker) deliberately rejects the refund, the project will be unable to complete the settlement process.

contracts/Foo.sol:L332 - L342

```
if (finalFloor.stock < maxStock) {
    // 退回费用
    for (uint256 i = 0; i < finalFloor.partners.length; i++) {
        payable(finalFloor.partners[i].addr).transfer(finalFloor.partners[i].fee);
    }
    if (floors.length == 1) {// 只有一层, 直接返回
        return;
    }
    finalFloor = floors[floors.length - 2]; // 倒数第二层
    finalFloorIndex = floors.length - 1; // 倒数第二层索引
}
```

Recommendation

Consider using `call` instead of `transfer` when sending native tokens, and properly handle the potential risks introduced by external calls.

Status

The team has resolved this issue in commit d62b6d9.

2. Predictable hash values lead to controllable blind box rewards	
Severity: Medium	Category: Business Logic
Target: - contracts/Foo.sol - contracts/Erc20Foo.sol	

Description

The hash calculation used in the contract can be predicted, resulting in the number of blind box reward floors corresponding to this block being calculated before calling the function.

If the user has a large share in one of the previous floors, the number of blind box floors can be predicted by calculating the hash corresponding to the current block, and if the calculated hash corresponds to a blind box floor that contains his or her own floor, he or she will make up the remaining share of the last floor, thus allowing the user's own floor to obtain the blind box reward.

contracts/Foo.sol:L259 - L263 and contracts/Erc20Foo.sol:L276 - L280

```
uint256 luck = uint256(keccak256(abi.encodePacked(
    blockhash(block.number - 1),
    blockhash(block.number - 5),
    block.coinbase
))) % floorCount;
```

Recommendation

It is recommended to use a third-party oracle such as Chainlink to obtain random numbers.

Status

The team has resolved this issue in commit 3d40fdf.

3. Incorrect total quantities were used to calculate rewards	
Severity: Medium	Category: Business Logic
Target: - contracts/Foo.sol - contracts/Erc20Foo.sol	

Description

When distributing rewards, `fee` and `amount` are used to calculate separately.

contracts/Foo.sol:L205 - L216 and contracts/Erc20Foo.sol:L285 - L296

```
// finalBonusPool 分走amount的40%
uint256 _finalBonusPool = amount * finalBonusRate / 100;
finalBonusPool += _finalBonusPool;
// 盲盒奖励 level1BonusPool 分走amount的40%
uint256 level1BonusPool = fee * blindBoxRate / 100;
blindBoxReward(level1BonusPool, index);
// 分红奖励 level2BonusPool 分走amount的19%
uint256 level2BonusPool = fee * dividendRate / 100;
// dividendReward(level2BonusPool, index);
// 创建者奖励 creatorReward 分走amount的1%
// uint256 creatorReward = fee * creatorRate / 100;
payable(creator).transfer(fee * creatorRate / 100);
```

However, `fee` and `amount` are not exactly the same, this is due to the loss of precision in the stockFee calculation entered by the user.

contracts/Foo.sol:L173,174 and contracts/Erc20Foo.sol:L224,225:

```
uint256 fee = buildFee(index);
uint256 stockFee = (fee * stock) / 100;
```

The loss of precision here will cause all stock fees to be accumulated, that is, `floor.amount `is not equal to `fee`.so the value used above, `amount` is always less than `fee`.

Therefore, the calculation results of the above three rewards will be larger than the results of the actual assets in the contract, which may cause the contract to be unable to distribute rewards normally, thus causing a run on the bank.

Recommendation

It is recommended to replace the `fee` in the above reward calculation with `amount`.

Status

The team has resolved this issue in commit 98d6911.

4. Precision loss	
Severity: Low	Category: Numerics
Target: - contracts/Erc20Foo.sol - contracts/Foo.sol	

Description

The code does not consistently follow the "multiply-before-divide" best practice in certain calculations, which may lead to precision loss.

contracts/Erc20Foo.sol:L160

```
return totalAmount * dividendRate / 100 * userStocks / (maxStock * finishedFloors); // 分红奖励
```

contracts/Erc20Foo.sol:L214

```
Invitee memory invitee = Invitee(inviter, stockFee * 1 / 100 * 30 / 100);
```

contracts/Foo.sol:L153

```
return totalAmount * dividendRate / 100 * userStocks / (maxStock * finishedFloors); // 分红奖励
```

contracts/Foo.sol:L196

```
Invitee memory invitee = Invitee(inviter, stockFee * 1 / 100 * 30 / 100);
```

Recommendation

It is recommended to adhere to the "multiply-before-divide" best practice to prevent precision loss in calculations.

Status

The team has resolved this issue in commits ab151ad, 6d7cf8d and 1be5679.

5. The excess stockFee was not refunded	
Severity: Low	Category: Business Logic
Target: - contracts/Foo.sol	

Description

In the `` `foo` `` contract, the user needs to pay `` `stockFee` `` in order to participate in the project. Each tier has a maximum shareholding limit of 100 shares and the price per share increases with the number of tiers.

The `` `Erc20Foo` `` contract will strictly transfer `` `stockFee` `` equivalent token, but the `` `foo` `` contract only requires `` `msg.value>=stockFee` ``. Suppose 100 shares require only 1 ether, but if the user buys 100 shares and pays 1.1 ether, the excess 0.1 ether will not be returned, causing a loss to the user.

Recommendation

It is recommended that excess funds paid by users be returned to them.

Status

The team has resolved this issue in commit 98d6911.

6. Mismatch with comment	
Severity: Low	Category: Code Quality
Target: - contracts/Foo.sol - contracts/Erc20Foo.sol	

Description

In the `FooFactory` and `Erc20FooFactory` contracts, the newly created contracts are granted the permission to mint NFTs during `createFoo`. However, the `Foo` and `Erc20Foo` contracts do not have corresponding interfaces to execute this operation.

contracts/Foo.sol:L57 - L62 and contracts/Foo.sol:L56 - L61

```
if (_nft != address(0)) {
    // 确保 Foo 合约有铸造 NFT 的权限
    if(nft.hasRole(nft.DEFAULT_ADMIN_ROLE(), address(this))){
        nft.grantRole(nft.MINTER_ROLE(), foo);
    }
}
```

The code sets `dividendRate` to 18, but the comment incorrectly states it as 19.

contracts/Foo.sol:L211and contracts/Foo.sol:L229

```
// 分红奖励 level2BonusPool 分走amount的19%
```

Recommendation

Consider modifying the comments or adding an NFT minting function to the `Foo` and `Erc20Foo` contracts.

Status

The team has resolved this issue in commits d5b0d4c and ff5ae28.

7. Centralization risk	
Severity: Low	Category: Centralization
Target: - contracts/Erc20Foo.sol - contracts/Foo.sol - contracts/Erc20FooFactory.sol - contracts/FooFactory.sol - contracts/UpTopNFT.sol	

Description

The contract code contains multiple privileged functions managed by privileged addresses. For example, in the factory contract, the owner can set `createFee`, `timeOut`, and in foo contract, they can perform `setFloorFee` and other operations.

If the owner's private key is compromised, an attacker could modify critical parameters, jeopardizing user interests and causing financial harm to the project itself.

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

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been acknowledged by the team.

8. Missing events for functions that change critical state	
Severity: Low	Category: Logging
Target: - contracts/FooFactory.sol - contracts/Erc20FooFactory.sol	

Description

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes that allow users to evaluate them. Missing events do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in protocol users.

In the `Erc20FooFactory` and `FooFactory` contracts, events are lacking in the privileged setter functions `setCreateFee()`, `setTimeout()`, `setTimeoutOrd()`, `setNFT()`.

Recommendation

It is recommended to emit events for critical state changes.

Status

The team has resolved this issue in commit 29f9a3d.

2.3 Informational Findings

9. Redundant Code	
Severity: Informational	Category: Redundancy
Target: - contracts/Foo.sol - contracts/Erc20Foo.sol - contracts/FooFactory.sol	

Description

Unused code should be removed before deploying the contract to mainnet. We have identified the following code are not being utilized:

contracts/Foo.sol:L80 - L85
contracts/Erc20Foo.sol:L83 - L88

```
event FinalBonusPoolUpdate(uint256 floor, uint256 fee);

modifier onlyCreator() {
    require(msg.sender == creator, "You are not the creator");
    _;
}
```

contracts/FooFactory.sol:L11

```
event Erc20FooCreated(address indexed foo, address creator, string icon, string title,
string message, uint256 startTime, address erc20);
```

Recommendation

Consider removing the redundant code.

Status

The team has partially resolved this issue in commit d5b0d4c.

10. Missing two-step transfer ownership pattern	
Severity: Informational	Category: Business logic
Target: - contracts/FooFactory.sol - contracts/Erc20FooFactory.sol	

Description

The `Erc20FooFactory` and `FooFactory` contracts inherit from the `OwnableUpgradeable` contract. This contract does not implement a two-step process for transferring ownership. Thus, ownership of the contract can easily be lost when making a mistake in transferring ownership.

Recommendation

Consider using the Ownable2StepUpgradeable contract from OpenZeppelin instead.

Status

This issue has been acknowledged by the team.

11. Use of floating pragma	
Severity: Informational	Category: Configuration
Target: - All	

Description

```
pragma solidity ^0.8.24;
pragma solidity ^0.8.18;
```

All contracts except `USDTCoin` use float compiler version `^0.8.24`, and `USDTCoin` uses float compiler version `^0.8.18`.

Using floating pragma `^0.8.24` and `^0.8.18` statements 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.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 2044641 :

File	SHA-1 hash
contracts/Erc20Foo.sol	6643e8c16b299c62b446c6ad416f91ef81611033
contracts/Erc20FooFactory.sol	32021e5690af81ca3bd5fc39ec17017762e49a81
contracts/Foo.sol	cc7b63e55d9d5896bc626e0561615b6a4edb580c
contracts/FooFactory.sol	107c574b1f97341098713c22069e616ec58195a6
contracts/FooLib.sol	a6ca455318bf204b4b9cc11e4dd6c4bf1dcd578d
contracts/Lock.sol	38d42f01215cc5865ee5af9f14d4f6189a223cff
contracts/USDTCoin.sol	8fd9d316f3ad08d69634804e0effce52bc249501
contracts/UpTopNFT.sol	6cf2dd9c567b0e88f1706d576cf5d8266028ffc4