SALUS SECURITY

JUN 2024

CODE SECURITY ASSESSMENT

AVALON

Overview

Project Summary

- Name: Avalon - AVAF
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/avalonfinancexyz/avaf
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Avalon - AVAF
Version	v2
Type	Solidity
Dates	Jun 11 2024
Logs	May 08 2024; Jun 11 2024

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	0
Total Low-Severity issues	3
Total informational issues	2
Total	6

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	Users can use transferFrom() to bypass the logic in transfer()	High	Business Logic	Resolved
2	Possible reentrancy in withdraw()	Low	Reentrancy	Resolved
3	Third-party dependencies	Low	Dependency	Acknowledged
4	Missing events for functions that change critical state	Low	Logging	Acknowledged
5	The owner cannot remove support for a token in case of an emergency	Low	Business Logic	Acknowledged
6	Missing zero address checks	Informational	Data Validation	Acknowledged

2.2 Notable Findings

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

1. Users can use transferFrom() to bypass the logic in transfer()

Severity: High	Category: Business Logic

Target:
- contracts/reward/GovRevenueStaking.sol
- contracts/reward/StakingRewardPool.sol

Description

In the AVAF protocol, there are numerous specific handling logics for token transfers, but there's a common oversight concerning transfers made using the transferFrom() function of the ERC20 token standard.

contracts/reward/GovRevenueStaking.sol:L163-L165

```
function transfer(address, uint256) public virtual override returns (bool) {
    revert("not allowed.");
}
```

contracts/reward/StakingRewardPool.sol:L134-L142

```
function transfer(
    address to,
    uint256 amount
) public virtual override returns (bool) {
    super.transfer(to, amount);
    boostConfigure.updateUser(msg.sender);
    boostConfigure.updateUser(to);
    return true;
}
```

When users perform token transfers using transferFrom(), they bypass the logic in the transfer() function, which could potentially lead to unexpected results.

Recommendation

Consider overriding _transfer() to implement specific logic.

Status

The team has resolved this issue in commit 310936f.

2. Possible reentrancy in withdraw()

Severity: Low	Category: Reentrancy

Target:
- contracts/reward/GovRevenueStaking.sol
- contracts/reward/StakingRewardPool.sol

Description

There are several places in the AVAF protocol that do not follow the Check-Effect-Interaction rule, which may lead to a reentrancy attack.

contracts/reward/GovRevenueStaking.sol:L107-L117

```
function withdraw(
    uint pid,
    uint _amount
) external updateReward(msg.sender) {
    ...
    stakeInfo[pid].stakingToken.safeTransfer(msg.sender, _amount);
    boostConfigure.updateUser(msg.sender);
}
```

contracts/reward/StakingRewardPool.sol:L83-L88

```
function withdraw(uint _amount) external updateReward(msg.sender) {
    ...
    stakingToken.safeTransfer(msg.sender, _amount);
    boostConfigure.updateUser(msg.sender);
}
```

Since the tokens are transferred before the boost configuration is updated, it is possible to perform a reentrancy attack if the token has some kind of call-back functionality, e.g. pBTC.

As the boostConfigure contract is a third-party contract, we can't be sure of the exact reentry path. But we think it is best practice to always follow the Check-Effect-Interaction rule in complicated call stacks.

Recommendation

It is recommended to follow the "Check-Effect-Interaction" rule in the code.

Status

The team has resolved this issue in commit 8f60e1a4.

3. Third-party dependencies

Severity: Low	Category: Dependency
Target: - contracts/vester/VesterManager.sol - contracts/reward/GovRevenueStaking.sol	

Description

The AVAF protocol relies on the checker contract to enable satisfaction checking, and the boostConfigure contract to enable the calculation of rewards. The current audit treats third-party entities as black boxes and assumes they are working correctly. However, in reality, third parties could be compromised, resulting in the loss of user assets.

Recommendation

We understand that the business logic requires interaction with third parties. We encourage the team to regularly monitor the statuses of third parties to reduce the impacts when they are not functioning properly.

Status

This issue has been acknowledged by the team.

4. Missing events for functions that change critical state	
Severity: Low	Category: Logging
Target: - contracts/tokens/baseToken.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 baseToken.sol, events are lacking in the privileged setter functions (e.g. setHandler() and setMinter()).

Recommendation

It is recommended to emit events for critical state changes.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

5. The owner cannot remove support for a token in case of an emergency

Severity: Informational	Category: Business Logic
Target: - contracts/reward/GovRevenueStaking.sol	

Description

The owner does not have the ability to remove a token from the list of available tokens.

However, it should not be overlooked that the supported tokens can be hacked, and in order to prevent such an attack from affecting the AVAF protocol, the contract should have an appropriate emergency exit feature.

Recommendation

Consider adding an emergency exit function for removing a token from the support list (stakeInfo[]).

Status

This issue has been acknowledged by the team.

6. Missing zero address checks

Severity: Informational	Category: Data Validation

Target:
- contracts/reward/GovRevenueStaking.sol
- contracts/reward/StakingRewardsManager.sol

Description

It is considered a security best practice to verify addresses against the zero address during initialization or setting. However, the following code does not verify addresses:

contracts/reward/GovRevenueStaking.sol:L47; L68-L72

```
constructor(
    string memory name,
    string memory symbol,
    address _rewardToken
) ERC20(name, symbol) {
    rewardsToken = IERC20(_rewardToken);
}

function addToken(IERC20 stakingToken) external onlyOwner {
    _checkToken(stakingToken);
    StakingTokenInfo storage newStakeToken = stakeInfo.push();
    newStakeToken.stakingToken = stakingToken;
}
```

contracts/reward/StakingRewardsManager.sol:L21-L36

```
function addPool(
    string memory name,
    string memory symbol,
    IERC20 stakingToken,
    IERC20 rewardsToken
) external onlyOwner {
    ...
    newPool.stakingToken = stakingToken;
    newPool.rewardsToken = rewardsToken;
}
```

contracts/vester/LinearVester.sol:L19-L22

```
constructor(address _manager, uint256 _vestingDuration) {
    vesterManager = _manager;
    vestingDuration = _vestingDuration;
}
```

Recommendation

Consider adding zero address checks for address variables.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit b2df3bb:

File	SHA-1 hash
Checker.sol	572787a995c6b7c369f77cbb767df477b25c6b17
GovRevenueStaking.sol	0c685bcbe600287f239417b91f5a1b10b828430f
StakingRewardPool.so	43e619a2ef3164754a72fc101922c42e9bdced91
StakingRewardsManager.sol	d67e553797f49282c7014b93a9a10196f9c04e1a
AVAF.sol	b045f9d7c925a3976b0b216a3ab43b64b176148f
baseToken.sol	2f470e4ad86c3b89f63ead2f2f75a98abede6937
esAVAF.sol	5daa6a51c214f5c178464892e38cb4b5b32dc819
stAVAF.sol	a5c8c39ee5fb4611d1202f861a17a229cb8baa22
LinearVester.sol	77b64394c38cf31ebca720644f7b381a118e7b5f
Vester.sol	6381373350212d4ed6566b2ecf6419774d990dd7
VesterManager.sol	a16dcd6f96375f0ab21ec195c760d308395eab43