SALUS SECURITY

MAR 2024

CODE SECURITY ASSESSMENT

DVOL FINANCE

Overview

Project Summary

- Name: dVOL Finance - devault contracts incremental audit
- Platform: BNB Smart Chain; Arbitrum
- Address Set:
  - BNB Smart Chain:
    - Proxy: 0xC6553F147D418dFe3745EBa56514de13feF67eA2
    - Implementation: 0x9B66Cfc6a61F48F1f6060d427225F7aD915cA304
  - Arbitrum:
    - Proxy: 0xce2C993406E86e0efd7D74A83a9fEfdB35bBE05c
    - Implementation: 0x29385d8905ae3b521f84BD509Ba497D14AEBd132
- Language: Solidity
- Repository: https://github.com/dvol-finance/devault-contracts
- Audit Scope: See Appendix - 1

Project Dashboard

Application Summary

Name	dVOL Finance - devault contracts incremental audit
Version	v4
Type	Solidity
Date	Apr 02 2024
Logs	Mar 19 2024; Mar 27 2024; Mar 29 2024; Apr 02 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	1
Total Low-Severity issues	2
Total informational issues	1
Total	4

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	The target vault's auto reinvest flag is incorrectly set in reinvest()	Medium	Business Logic	Resolved
2	Incorrectly judged default auto reinvest flag	Low	Business Logic	Resolved
3	Loss of precision could lead to unexpected loss of lpToken by the user	Low	Numerics	Acknowledged
4	Can use immutable to save gas	Informational	Gas Optimization	Acknowledged

2.2 Notable Findings

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

1. The target vault's auto reinvest flag is incorrectly set in reinvest()	
Severity: Medium	Category: Business Logic
Target: - contracts/Vault.sol	

Description

contracts/Vault.sol:L389-L393

```
function reinvest(uint256 vaultId, uint256 investVaultId, address to) external {
    IVault(to).setAutoReinvest(investVaultId,
idVaultStateMap[vaultId].userReinvestStatusMap[msg.sender] ==
USER_REINVEST_STATUS_AUTO_REINVEST);
    reinvestInternal(vaultId, investVaultId, msg.sender, to);
}
```

In the reinvest() function, it would call the target vault's setAutoReinvest() function to synchronize the user's auto reinvest flag setting.

contracts/Vault.sol:L486-L499

```
function setAutoReinvest(uint256 vaultId, bool isAutoReinvest) requireExists(vaultId)
public {
    setAutoReinvestInternal(vaultId, msg.sender, isAutoReinvest);
}

function setAutoReinvestInternal(uint256 vaultId, address user, bool isAutoReinvest)
requireExists(vaultId) internal {
    VaultState storage vaultState = idVaultStateMap[vaultId];
    if (isAutoReinvest) {
        vaultState.userReinvestStatusMap[user] = USER_REINVEST_STATUS_AUTO_REINVEST;
    } else {
        vaultState.userReinvestStatusMap[user] = USER_REINVEST_STATUS_NOT_AUTO_REINVEST;
    }
}
```

In the setAutoReinvest() function, the entity being modified is msg.sender. This means that within the logic flow above, the target vault incorrectly modifies the auto reinvest flag of the original vault(msg.sender in this call).
And the user's flag is not synchronized to the target vault.

Recommendation

It is recommended to correctly implement the logic that makes the auto flag status of the reinvest target vault consistent.

Status

The team has resolved this issue in commit d44d814.

2. Incorrectly judged default auto reinvest flag	
Severity: Low	Category: Business Logic
Target: - contracts/Vault.sol	

Description

contracts/Vault.sol:L501-L504

```
uint8 constant private USER_REINVEST_STATUS_AUTO_REINVEST = 1;
uint8 constant private USER_REINVEST_STATUS_NOT_AUTO_REINVEST = 2;

function getAutoReinvest(uint256 vaultId, address user) requireExists(vaultId) view
external returns (bool) {
//        default is auto reinvest, and it's value is 0
    return idVaultStateMap[vaultId].userReinvestStatusMap[user] ==
USER_REINVEST_STATUS_AUTO_REINVEST ||
idVaultStateMap[vaultId].userReinvestStatusMap[user] == 0;
}
```

According to the above comment, the user's default auto reinvest flag is true.

contracts/Vault.sol:L389-L402

```
function reinvest(uint256 vaultId, uint256 investVaultId, address to) external {
//        the auto reinvest flag of target vault should be the same as the previous
vault
    IVault(to).setAutoReinvest(investVaultId,
idVaultStateMap[vaultId].userReinvestStatusMap[msg.sender] ==
USER_REINVEST_STATUS_AUTO_REINVEST);
    reinvestInternal(vaultId, investVaultId, msg.sender, to);
}
function autoReinvest(uint256 vaultId, uint256 investVaultId, address user)
requireExists(vaultId) requireManager external {
    require(idVaultStateMap[vaultId].userReinvestStatusMap[user] ==
USER_REINVEST_STATUS_AUTO_REINVEST, "not auto reinvest");

//        if the previous vault is not auto reinvest, then the target vault should be
auto reinvest too
    setAutoReinvestInternal(investVaultId, user, true);
    reinvestInternal(vaultId, investVaultId, user, address(this));
}
```

However, the default value of 0 for userReinvestStatusMap is treated as false in the reinvest() and autoReinvest() functions.
This may result in the user not being able to use the auto reinvest function properly and will cost extra gas to set userReinvestStatusMap to 2 (true).

Recommendation

It is recommended to make correct judgment on the default value of userReinvestStatusMap.

Status

The team has resolved this issue in commit d44d814.

3. Loss of precision could lead to unexpected loss of lpToken by the user	
Severity: Low	Category: Numerics
Target: - contracts/Vault.sol	

Description

contracts/Vault.sol:L274-L294

```
function claim(uint256 vaultId) requireExists(vaultId) requireEnded(vaultId) external {
    ...
    ILPToken(vaultState.lpTokenContract).burn(msg.sender, lpTokenBalance);
    for (uint8 i = 0; i < vaultInfo.claimTokens.length; i++) {
        address claimToken = vaultInfo.claimTokens[i];
        uint256 transferAmount = lpTokenBalance * vaultInfo.claimTokenAmounts[i] /
vaultState.soldAmount;
        if (transferAmount > 0) {
            ILPToken(vaultState.lpTokenContract).transferTo(payable(msg.sender),
transferAmount, claimToken);
        }
    }
}
```

The claim() function is used to allow users to claim their rewards. Before performing this operation, the lpToken held by the user is burned. However, when calculating the rewards, the precision of calculations may result in the user not being able to access the reward tokens.

For example, when the lpToken balance is 1e8, if soldAmount is 10^8 times greater than claimTokenAmounts, the user will not be able to access the reward.
This situation is more prominent in cases where the claim token's decimal and the lp token's decimal are both smaller, such as WBTC.

This could result in the user suffering an unexpected loss, affecting their experience, or even leading to leaving the protocol.

Recommendation

It is recommended to set the minimum value of deposit for each vault in order to avoid as much as possible unexpected losses for users. Additionally, using a token with a larger decimal value for the lp token and the claim token can also help alleviate precision loss issues.

Status

This issue has been acknowledged by the team. The team has claimed that they will use a sufficiently large number of lp tokens and the claim token.

2.3 Informational Findings

4. Can use immutable to save gas	
Severity: Informational	Category: Gas Optimization
Target: - contracts/Vault.sol	

Description

The following variables could be set immutable.

contracts/Vault.sol:L90

```
address public WETH9;
```

Recommendation

Consider defining variables set in the constructor and not changed after deployment as immutable.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

We audited the commit a83571a that introduced new features to the dvol-finance/devault-contracts repository.

File	SHA-1 hash
Vault.sol	e42983d27f9fba24485519e5d01624c4d940220c