SALUS SECURITY

APR 2024

CODE SECURITY ASSESSMENT

BITSMILEY PROTOCOL

Overview

Project Summary

- Name: Bitsmiley Protocol - Incremental Audit
- Platform: Merlin Chain
- Language: Solidity
- Repository: https://github.com/bitSmiley-protocol/evm-contracts
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Bitsmiley Protocol - Incremental Audit
Version	v2
Type	Solidity
Dates	Apr 22 2024
Logs	Apr 01 2024; Apr 22 2024

Vulnerability Summary

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

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	Unable to trigger liquidation	High	Business Logic	Resolved
2	Users cannot hold multiple collateral positions and cannot change the type of collateral	Low	Business Logic	Acknowledged
3	Lack of upper limit for fees in setFeeRate()	Informational	Centralization	Acknowledged

2.2 Notable Findings

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

1. Unable to trigger liquidation

Severity: High	Category: Business Logic

Target:
- contracts/BitSmiley.sol

Description

contracts/BitSmiley.sol:L270-L312

```
function liquidate(
    address _vault
) external nonReentrant whenNotPaused {
    VaultInfo memory vaultInfo = _getVault(_vault);

    (uint256 totalFee, uint256 deltaFee) =
stabilityFee.accrueFee(vaultInfo.collateralId, _vault);

    (
        bool isSafe,
        IVaultManager.Vault memory vault
    ) = vaultManager.vaultPositionWithFee(vaultInfo.collateralId, _vault, deltaFee);

    // if vault does not exist or is safe, cannot be liquidated.
    if (isSafe) {
        revert CannotBeLiquidated();
    }

    _payFee(vaultInfo.collateralId, _vault, msg.sender, totalFee);

    // vault.debtBitUSD contains the debit usd minted + fee till last update
    // so the total debt now is `vault.debtBitUSD + deltaFee`.
    uint256 toBurn = vault.debtBitUSD + deltaFee - totalFee;
    bitUSD.burn(msg.sender, toBurn);
    vaultMinted[_vault] -= int256(toBurn);
    ...
}
```

In the highlighted part of the code above, since vault.debtBitUSD contains the debit usd minted + fee until last update, the fee needs to be deducted from vault.debtBitUSD when calculating the real debt.

contracts/VaultManager.sol:L153-L165

```
function vaultPositionWithFee(bytes32 _collateralId, address _vault, uint256 _fee)
external view returns(
    bool isSafe,
    Vault memory vault
) {
    vault = vaults[_collateralId][_vault];
    vault.debtBitUSD += _fee;
```

```
    uint256 collateralEvaluation = vault.lockedCollateral
        * IOracles(oracle).getPrice(_collateralId)
        * collateralTypes[_collateralId].safetyFactor;

    isSafe = collateralEvaluation >= R * vault.debtBitUSD;
}
```

However, deltaFee has already been added to debtBitUSD in the vaultPositionWithFee() function, resulting in deltaFee being duplicated and added to toBurn. Ultimately, this leads to an underflow in the vaultMinted calculation, which leads to the transaction revert.

Recommendation

It is recommended to remove the duplicate calculation logic.

Status

The team has resolved this issue in commit [46b251ba](46b251ba).

2. Users cannot hold multiple collateral positions and cannot change the type of collateral

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

Description

contracts/BitSmiley.sol:L270-L312

```
function _openVault(bytes32 _collateralId) internal returns (address newVault) {
    if (owners[msg.sender] != address(0)) {
        revert AlreadyOpenedVault();
    }

    newVault = address(new Vault(address(vaultManager)));
    owners[msg.sender] = newVault;
    vaults[newVault] = VaultInfo(msg.sender, _collateralId);

    emit VaultOpened(
        msg.sender,
        _collateralId,
        newVault
    );
}
```

The BitSmiley contract allows users to use different collateral to obtain the stablecoin bitUSD. However, the highlighted code above indicates that each user can only create a vault with one type of collateral and cannot change the collateral used after creation.

Attach Scenario
This may affect the user experience in the following scenario:
a) When the user calls openVault(), a vault is created using WBTC.
b) Retrieving a collateral when the user calls repay().
c) When the user wants to create a vault using WETH, calling openVault() may REVERT due to a code check in the highlighted section.
This means that once the user creates a vault, he cannot change his collateral anymore.

Recommendation

Consider adding a delet vault function that can be called only when the user's debt and collateral are 0.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

3. Lack of upper limit for fees in setFeeRate()

Severity: Informational	Category: Centralization
Target: - contracts/StabilityFee.sol	

Description

The setFee() function is used to modify the value of the protocol fee.

contracts\StabilityFee.sol:L53-L62

```
function setFeeRate(bytes32 _collateralId, uint256 _rate) public onlyOwner {
    if (_rate >= FEE_BASE) {
        revert InvalidFee();
    }

    (, uint256 rate) = feeRates.tryGet(_collateralId);
    emit FeeRateUpdated(msg.sender, _collateralId, rate, _rate);

    feeRates.set(_collateralId, _rate);
}
```

However, the function has no upper limit on fees. If the fee setter's private key is compromised, the attacker can set cross-chain fee to type(uint256).max, preventing users from performing repay/liquidate operations.

Recommendation

It is recommended to include a reasonable upper limit for cross-chain fee in setFeeRate().

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

We audited the commit 17d05cb that introduced new features to the https://github.com/bitSmiley-protocol/evm-contracts repository.

File	SHA-1 hash
AbstractBitSmiley.sol	76e04814ecbc725a0d2865b868b3de71440f540c
BitSmiley.sol	11c598ed56b14a3835ea5efacea6e63e287f60c4
StabilityFee.sol	ee6f9d13f938d956cf22c28cd40981ea9a1c19fc
OnlyCaller.sol	7564b045023c0987312c3cf1724117e5fb10ab00
VaultManager.sol	7895fd8c216d96e59aa163075ae21aac2bed6158