SALUS SECURITY

MAR 2024

CODE SECURITY ASSESSMENT

BITSMILEY PROTOCOL

Overview

Project Summary

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

Project Dashboard

Application Summary

Name	Bitsmiley Protocol
Version	v4
Type	Solidity
Date	Mar 12 2024
Logs	Feb 29 2024; Mar 02 2024; Mar 11 2024; Mar 12 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	4
Total Low-Severity issues	5
Total informational issues	3
Total	12

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 are unable to repay their loans in the paused status, resulting in being unexpectedly liquidated	Medium	Business Logic	Pending
2	BitSmileyNoFee do not implement the pause feature correctly	Medium	Business Logic	Pending
3	Missing collateral rate leads to miscalculation	Medium	Numerics	Resolved
4	Inconsistency between eligible's accurate external and internal recording	Medium	Numerics	Resolved
5	VaultManager lacks decimal normalization for collateral	Low	Numerics	Pending
6	The totalParticipants variable incorrectly tracks NFT count instead of total users	Low	Business Logic	Pending
7	Attacker can gain mint and burn access to BitUSD by front-running init()	Low	Front Running	Resolved
8	The calculateVaultOverview() function will revert if the position is not healthy	Low	Business Logic	Resolved
9	Implementation can be initialized	Low	Configuration	Resolved
10	Lack of indexed parameters in events	Informational	Logging	Pending
11	The openVaultAndMintFromBTC() is missing the nonReentrant modifier	Informational	Business Logic	Resolved
12	Redundant code	Informational	Redundancy	Resolved

2.2 Notable Findings

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

1. Users are unable to repay their loans in the paused status, resulting in being unexpectedly liquidated

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

Description

BitSmileyNoFee.sol inherits the PausableUpgradeable contract and uses the whenNotPaused modifier in the following function:

- openVault()
- openVaultAndMintFromBTC()
- mintFromBTC()
- repayToBTC()
- liquidateVaultBTC()

While the contract is in a paused state, the user will not be able to redeem the collateral by calling the repayToBTC() function.

In this case, the users will not be able to take action to protect their collateral if the market fluctuates and causes the price of the collateral to fall. Once the contract goes back into the unpaused state, the liquidator can immediately liquidate the user's collateral.

This is not fair to the user, thus the repayToBTC() function should not be paused no matter what state the contract is in.

Recommendation

Consider removing the whenNotPaused modifier from the repayToBTC() function.

2. BitSmileyNoFee does not implement the pause feature correctly

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

Description

BitSmileyNoFee.sol inherits OpenZeppelin's PausableUpgradeable module to implement the key modifier whenNotPaused().

But there is no way to pause/unpause a contract because the _pause() function is internal by default, and needs to be written as a custom public function and then call the _pause() internal function.

Recommendation

Consider writing a custom pause()/unpause() function that calls _pause()/_unpause internally.

3. Missing collateral rate leads to miscalculation

Severity: Medium	Category: Numerics
Target: - contracts/VaultManager.sol	

Description

The calculateVaultOverview() function is used to get the state of the user's vault, but the function has some calculation errors caused by the missing collateralType.rate.

a) contracts/VaultManager.sol:L208-L209

```
// The amount of BitUSD available to mint
overview.availableToWithdraw = _div(overview.availableToMint, priceWithSafety);
```

The *overview.availableToMint* is missing the collateralization factor, which causes this calculation to always be collateralType.rate times smaller than the expected result.

b) contracts/VaultManager.sol:L211-L212

```
// The liquidation price for the vault
overview.liquidationPrice = _div(overview.debtBitUSD, _mul(overview.lockedCollateral,
collateralType.safetyFactor));
```

The *overview.debtBitUSD* is missing the collateralization factor, which causes this calculation to always be collateralType.rate times smaller than the expected result.

Recommendation

It is recommended to multiply collateralType.rate in the numerator in the above calculation.

Status

The team has resolved this issue in commit 7b1863a, a3e8d2b.

4. Inconsistency between eligible's accurate external and internal recording

Severity: Medium	Category: Numerics

Target:
- contracts/BitSmileyNoFee.sol
- contracts/VaultManager.sol

Description

contracts/VaultManager.sol:L228-L299

```
function calculate(
    ...
    int256 _bitUSD
) external onlyCaller whenNotPaused {
    CollateralType memory collateralType = collateralTypes[_collateralId];
    if (collateralType.rate == 0) {
        revert CollateralNotInitialized();
    }
    ...
    int dtab = _mul(collateralType.rate, _bitUSD);
    ...
    eligible[_debtOwner] = _add(eligible[_debtOwner], dtab);
}
```

In VaultManager.calculate(), *eligible* contains the collateral factor collateralType.rate.

contracts/BitSmileyNoFee.sol:L134-L166

```
function _mintFromBTC(
    address _vault,
    int256 _bitUSD,
    int256 _collateral
) internal whenNotPaused senderOwner(_vault) {
    ...
    // deduct the eligible bitUSD available from the vault, lock it in
    // the current address.
    vaultManager.transferEligible(_vault, address(this), _bitUSD);

    // mint the bitUSD
    bitUSD.mint(msg.sender, uint256(_bitUSD));
}
```

But in BitSmileyNoFee._mintFromBTC(), eligible does not contain the collateralization factor collateralType.rate.
This inconsistency leads to confusion in BitSmiley's record-keeping, as eligible[user] still holds a large amount of eligible after the user has withdrawn their BitUSD.

Recommendation

Consider keeping the *eligible* precision factor consistent.

Status

The team has resolved this issue in commit 9703337, 76081a1.

5. VaultManager lacks decimal normalization for collateral

Severity: Low	Category: Numerics
Target: - contracts/VaultManager.sol	

Description

contracts/VaultManager.sol:L228-L299

```
function calculate(
    bytes32 _collateralId,
    address _vaultAddr,
    int256 _collateral,
    int256 _bitUSD
) external onlyCaller {
    ...
    Vault memory vault = vaults[_collateralId][_vaultAddr];
    vault.lockedCollateral = _add(vault.lockedCollateral, _collateral);
    ...

    // vault is either less risky than before, or it is safe
    uint256 priceWithSafety = IOracles(oracle)
        .getPrice(_collateralId)
        * collateralType.safetyFactor;

    // The total debt for minted _bitUSD plus the extra fee
    uint256 vaultDebt = collateralType.rate * vault.debtBitUSD;

    if (
        !either(
            both(_bitUSD <= 0, _collateral >= 0),
            vaultDebt <= vault.lockedCollateral * priceWithSafety
        )
     ) {
        revert VaultPositionNotSafe();
    }
}
```

VaultManager.sol supports providing multiple tokens for collateralization (collateralId), but different collateral may have different decimals.

Missing normalization of token decimals will result in miscalculation of collateral value.

Recommendation

It is recommended to normalize the decimal for different collateral tokens.

We understand that the issue will not exist in BitSmileyNoFee (only WBTC is supported). However, if the team wants to use VaultManager as a base contract to support other tokens, the issue needs to be fixed.

6. The totalParticipants variable incorrectly tracks NFT count instead of total users

Severity: Low	Category: Business Logic
Target: - contracts/StakingERC721EndWithdraw.sol	

Description

contracts/StakingERC721EndWithdraw.sol:L73-L84

```
function _stake(uint256 _tokenId, address _from) internal {
    totalParticipants += 1;

    tokenStake[_tokenId] = TokenStake({
        owner: _from,
        stakedTime: block.number,
        withdrawTime: 0
    });
    userStakes[_from].push(_tokenId);

    emit Staked(_from, _tokenId);
}
```

In the code base, the totalParticipants variable tracks the total number of participants in the pledge. In fact, the value of this variable is increased by 1 whenever a user pledges an NFT.
The result is that the totalParticipants variable tracks the total number of NFTs pledged, not the total number of people participating in the pledge.

Recommendation

It is recommended to add an if judgment condition to check if the `*from*` address has already been involved in a pledge.

7. Attacker can gain mint and burn access to BitUSD by front-running init()

Severity: Low	Category: Front Running
Target: - contracts/BitUSDL2.sol	

Description

contracts/BitUSDL2.sol:L25-L33

```
function init(address _caller) external {
    if (initialized) {
        revert AlreadyInitialized();
    }

    caller = _caller;

    initialized = true;
}
```

In BitUSDL2.sol, there is an external init() function, which means that once the contract is deployed, anyone can become a caller by calling init() and thus gain access to mint and burn the BitUSD contract.

Recommendation

It is recommended to deploy the BitUSDL2 contract in BitSmileyNoFee.initialize().
For example, the BitUSDL2 constructor could be modified as follows:

```
constructor(string memory name_, string memory symbol_) ERC20(name_, symbol_) {
    caller = msg.sender;
}
```

And BitSmileyNoFee.initialize() can be modified as follows

```
function initialize(address _vaultManager, address _bitUSD) external initializer {
    __Ownable_init();
    __Pausable_init();

    vaultManager = IVaultManager(_vaultManager);
    bitUSD = new BitUSDL2("BitUSDL2", "BU");
}
```

Status

The team has resolved this issue in commit f5e6d79a.

8. The calculateVaultOverview() function will revert if the position is not healthy

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

Description

contracts/VaultManager.sol:L182-L216

```
function calculateVaultOverview(
    bytes32 _collateralId,
    address _vaultAddr,
    int256 _collateral,
    int256 _bitUSD
) external view returns (
    VaultOverview memory overview
) {
    ...
    Vault memory vault = vaults[_collateralId][_vaultAddr];

    overview.debtBitUSD = _add(vault.debtBitUSD, _bitUSD);
    overview.lockedCollateral = _add(vault.lockedCollateral, _collateral);

    uint256 vaultCapBitUSD = _div(_mul(overview.lockedCollateral, priceWithSafety),
collateralType.rate);

    // The amount of BitUSD available to mint
    overview.availableToMint = _sub(vaultCapBitUSD, overview.debtBitUSD);
    ...
}
```

The calculateVaultOverview() function is used to query the state of the user's vault. But the current code snippet will return an underflow error if vaultCapBitUSD(collateral denominated in native decimals) will be less than debtBitUSD(bitUSD denominated in native decimals).

This means that during market fluctuations, when the position will be unhealthy, users won't be able to call calculateVaultOverview() to get crucial data, which affects the user interaction experience.

Recommendation

Consider adding an If statement that stores 0 in case collateral < debt. For example, the highlighted code could be modified as follows:

```
if (vaultCapBitUSD > overview.debtBitUSD) {
    overview.availableToMint = _sub(vaultCapBitUSD, overview.debtBitUSD);
}
```

Status

The team has resolved this issue in commit 7b1863a.

9. Implementation can be initialized

Severity: Low	Category: Configuration
Target: - contracts/BitSmileyNoFee.sol - contracts/VaultManager.sol	

Description

The BitSmileyNoFee and VaultManager contracts use the Initializable module from OpenZeppelin.
According to the OpenZeppelin's documentation, it's best to call the `_disableInitializers` function in the constructor to prevent the implementation contract from being initialized by malicious users.

Recommendation

Consider using the _disableInitializers() function in the constructor to prevent malicious initialization of the Implement contract.

Status

The team has resolved this issue in commit f5e6d79a.

2.3 Informational Findings

10. Lack of indexed parameters in events

Severity: Informational	Category: Logging
Target: - contracts/StakingERC721EndWithdraw.sol	

Description

Event emission and properly indexed parameters assist off-chain observers watch, search, and filter on-chain activity.

contracts/StakingERC721EndWithdraw.sol

```
event Staked(address who, uint256 tokenId);
event Withdrawn(address who, uint256 tokenId);
```

In StakingERC721EndWithdraw.sol, the Staked and Withdrawn events do not apply the indexed keyword to the token and from parameters.

Recommendation

Consider indexing applicable event parameters to support the searching and filtering abilities of offchain services.

11. The openVaultAndMintFromBTC() is missing the nonReentrant modifier

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

Description

contracts/BitSmileyNoFee.sol:L67-L81

```
function openVaultAndMintFromBTC(int256 _bitUSD, int256 _collateral) external payable {
    address newVault = _openVault();
    _mintFromBTC(newVault, _bitUSD, _collateral);
}

/// @notice Mint bitUSD to a vault using the collateral
function mintFromBTC(
    address _vault,
    int256 _bitUSD,
    int256 _collateral
) external payable nonReentrant whenNotPaused senderOwner(_vault) {
    _mintFromBTC(_vault, _bitUSD, _collateral);
}
```

Both openVaultAndMintFromBTC() and mintFromBTC() function to mint BitUSD, but the openVaultAndMintFromBTC() function is missing the nonReentrant modifier.

The inconsistency in the code allows the user to bypass the nonReentrant modifier check by calling the openVaultAndMintFromBTC() function.

Recommendation

Consider adding the nonReentrant modifier to the openVaultAndMintFromBTC() function to keep the code consistent.

12. Redundant code

Severity: Informational	Category: Redundancy

Target:
- contracts/BitSmileyNoFee.sol
- contracts/VaultManager.sol

Description

1. contracts/BitSmileyNoFee.sol:L75-L81

```
/// @notice Mint bitUSD to a vault using the collateral
function mintFromBTC(
    address _vault,
    int256 _bitUSD,
    int256 _collateral
) external payable nonReentrant whenNotPaused senderOwner(_vault) {
    _mintFromBTC(_vault, _bitUSD, _collateral);
}
```

The whenNotPaused and senderOwner modifiers are already existing in the underlying function _mintFromBTC().

2. contracts/BitSmileyNoFee.sol:L25

```
uint256 public nextVaultId;
```

The nextVaultId variable is not used in the BitSmiley Protocol.

3. contracts/VaultManager.sol:L23

```
uint256 public constant G = 10 ** 9;
```

The G variable is not used in the BitSmiley Protocol.

Recommendation

Consider removing the redundant code.

Status

The team has resolved this issue in commit f5e6d79a.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 7f57c5d:

File	SHA-1 hash
BitSmileyNoFee.sol	5232ec392c6549ba0fb5818a28ce93923cae4148
BitUSDL2.sol	1e12c1fb8c935eb851b2801d91f59defa80221da
StabilityFee.sol	297a96108736bd0dea4f0ebf89c22ad8333cc87d
TransferUtil.sol	c0e7f1ad9da743393befcc26949c4fe4d52f3264
Vault.sol	2b3616b9e533247469271b9de3ad884a805c4703
VaultManager.sol	d941544f494b08aee57d72580d710e582d5651b1

And the subsequent audit covered the following files in commit a60e80a:

File	SHA-1 hash
BitSmileyNoFee.sol	6a7cbe0e70974a3e2700b07cb2e915b4fad8bbd2
BitUSDL2.sol	854ffadbc43217723b3382a171f54832f50dcbfa
StabilityFee.sol	297a96108736bd0dea4f0ebf89c22ad8333cc87d
TransferUtil.sol	1258d2d5e189266a4ff4f1dc6258a3e62d096f92
Vault.sol	30fdf919d01e74185196862869685f4ef9166091
VaultManager.sol	65403e69c54fcac43319360b3cfa9aa93ddbbd5e

The subsequent audit covered the following files in commit af1a04e:

File	SHA-1 hash
StakingERC721EndWithdraw.sol	093e80b85abdcda96dc10e71bd53bc86a55cb454