SALUS SECURITY

MAY 2024

CODE SECURITY ASSESSMENT

BITU PROTOCOL

Overview

Project Summary

- Name: BitU Protocol
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/BitU-Protocol/Bitu-Contract
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	BitU Protocol
Version	v4
Type	Solidity
Dates	May 27 2024
Logs	Apr 29 2024; Apr 30 2024; May 08 2024; May 27 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	4
Total Low-Severity issues	3
Total informational issues	2
Total	9

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 with FULL_RESTRICTED role could bypass withdraw restriction	Medium	Business Logic	Resolved
2	The residual custodian state may cause deposit to dos	Medium	Denial of Service	Resolved
3	Use of deprecated Chainlink function latestAnswer	Medium	Business Logic	Resolved
4	Centralization risk	Medium	Centralization	Mitigated
5	The first deposit can be dos	Low	Denial of Service	Acknowledged
6	Fixed amount of sBTU always remains locked into the contract	Low	Business Logic	Acknowledged
7	Missing events for functions that change critical state	Low	Logging	Acknowledged
8	Wrong comment or readme	Informational	Configuration	Resolved
9	Redundant code	Informational	Redundancy	Acknowledged

2.2 Notable Findings

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

1. Users with FULL_RESTRICTED role could bypass withdraw restriction

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

Description

In the BitU protocol, certain addresses may be restricted from depositing, withdrawing, and transferring funds for specific reasons.
contracts/StakedBITU.sol:L225-L238

```
function _withdraw(address caller, address receiver, address _owner, uint256 assets,
uint256 shares)
    internal
    override
    nonReentrant
    notZero(assets)
    notZero(shares)
{
    if (hasRole(FULL_RESTRICTED_STAKER_ROLE, caller) ||
hasRole(FULL_RESTRICTED_STAKER_ROLE, receiver)) {
      revert OperationNotAllowed();
    }

    super._withdraw(caller, receiver, _owner, assets, shares);
    _checkMinShares();
}
```

However, restricted users may attempt to circumvent the withdrawal requirements by using the approval functionality.

Attach Scenario
In the following scenario, a restricted user has successfully withdrawn and transferred assets to another address:
1) A restricted user grants approval for an sBTU token to another legitimate address under their control.
2) Using this approved address, the restricted user attempts to call the withdraw function, aiming to withdraw sBTU assets from the restricted address.

Recommendation

Consider adding a check for the sBTU owner in the withdraw() function.

Status

The team has resolved this issue in commit 14035f7b.

2. The residual custodian state may cause deposit to dos

Severity: Medium	Category: Denial of Service
Target: - contracts/BitUMinting.sol	

Description

contracts/BitUMinting.sol:L306-L319

```
function _transferCollateral(
    uint256 amount,
    address asset,
    address benefactor
) internal {
    // cannot mint using unsupported asset or native token even if it is supported for
redemptions
    if (!_supportedAssets.contains(asset)) revert UnsupportedAsset();
    if (!custodianRatiosSetted) revert InvalidCustodianAddress();
    uint256 totalTransferred = 0;
    address[] memory custodian = _supportedCustodian.values();
    if (asset != NATIVE_TOKEN){
        IERC20 token = IERC20(asset);
        for (uint256 i = 0; i < custodian.length; i++) {
            uint256 amountToTransfer = (amount * custodianRatios[custodian[i]]) /
10_000;
            token.safeTransferFrom(benefactor, custodian[i], amountToTransfer);
            totalTransferred += amountToTransfer;
        }
        uint256 remainingBalance = amount - totalTransferred;
        if (remainingBalance > 0) {
            token.safeTransferFrom(benefactor, custodian[custodian.length - 1],
remainingBalance);
        }
    }else{...}
}
```

During the deposit process, BitU assigns the user's collateral to different custodians. This process requires that all custodianRatios add up to equal 10_000, otherwise the highlighted part of the code will cause dos due to overflow.

contracts/BitUMinting.sol:L306-L319

```
function setCollateralDistributionRatios(
    address[] memory _custodians,
    uint256[] memory _distributionRatios
)external onlyRole(DEFAULT_ADMIN_ROLE) {
    if(_custodians.length != _distributionRatios.length) revert InvalidEqualLength();
    uint256 totalRatios;
    for (uint256 i = 0; i < _custodians.length; ++i){
        if(!isSupportedCustody(_custodians[i])) revert InvalidCustodianAddress();
        custodianRatios[_custodians[i]] = _distributionRatios[i];
        totalRatios += _distributionRatios[i];
```

```
    }
    if(totalRatios != 10_000) revert InvalidRatio();
    custodianRatiosSetted = true;
}
```

In the setCollateralDistributionRatios() function, there is a risk of not ensuring that the total sum equals 10,000 when setting custodianRatios. Specifically, this function can only guarantee that the total sum of the most recently passed _distributionRatios[] array is 10,000, but it overlooks any old data that might remain in custodianRatios.

This residual data can lead to unexpected denial-of-service attacks.

Recommendation

Consider removing all the status of the custodianRatios in advance in the setCollateralDistributionRatios() function.

Status

The team has resolved this issue in commit 391b17f3.

3. Use of deprecated Chainlink function latestAnswer

Severity: Medium	Category: Business Logic

Target:
- contracts/BitUMinting.sol

Description

contracts/BitUMinting.sol:L498

```
function getAssetPrice(address asset) internal view returns(int256) {
    if(!isSupportedAsset(asset)) revert InvalidAssetAddress();
    return AggregatorV2V3Interface(oracles[asset]).latestAnswer();
}
```

According to Chainlink's documentation, the latestAnswer function is deprecated. This function does not error if no answer has been reached but returns 0, causing an incorrect price fed to the BitUMinting.

Recommendation

Consider using the latestRoundData function to get the price instead. Add checks on the return data with proper revert messages if the price is stale or the round is uncomplete, for example:

```
(
    uint80 roundID,
    int256 price,
    ,
    uint256 timeStamp,
    uint80 answeredInRound
) = chainlink.latestRoundData();
require(timeStamp != 0, "ChainlinkOracle::getLatestAnswer: round is not complete");
require(answeredInRound >= roundID, "ChainlinkOracle::getLatestAnswer: stale data");
require(price != 0, "Chainlink Malfunction");
```

Status

The team has resolved this issue in commit e7182789.

4. Centralization risk

Severity: Medium	Category: Centralization
Target: - contracts/StakedBITU.sol	

Description

In the StakedBITU contract, there is a privileged DEFAULT_ADMIN_ROLE role. The admin role has the ability to:

- modify all permissions
- transfer the tokens in the contract except asset
- transfer the balance of the user marked as FULL_RESTRICTED_STAKER_ROLE to the specified address

If the admin's private key is compromised, the attacker can exploit the admin's role to transfer the assets of anyone in the protocol. In this case, the user's assets in the protocol will be seriously threatened.

Recommendation

We recommend transferring the owner role to a multisig account with a timelock feature for enhanced security. This ensures that no single person has full control over the account and that any changes must be authorized by multiple parties.

Status

The team has moved the admin privilege role to multisig address [0x07C3C3Ff0742A985cf15cA5A6b027679c2B57373](0x07C3C3Ff0742A985cf15cA5A6b027679c2B57373).

5. The first deposit can be dos

Severity: Low	Category: Denial of Service
Target: - contracts/StakedBITU.sol	

Description

contracts/StakedBITU.sol:L191-L194

```
function _checkMinShares() internal view {
    uint256 _totalSupply = totalSupply();
    console.log("totalSupply", totalSupply());
    if (_totalSupply > 0 && _totalSupply < MIN_SHARES) revert MinSharesViolation();
}
```

In StakedBITU.sol, both deposit and withdraw limit the minimum share.

ERC4626.sol:L199-L201

```
function _convertToShares(uint256 assets, Math.Rounding rounding) internal view virtual
returns (uint256) {
    return assets.mulDiv(totalSupply() + 10 ** _decimalsOffset(), totalAssets() + 1,
rounding);
}
```

A malicious user can transfer assets to a contract using the asset.transfer() function. This behavior may result in over-amplification of the denominator of the _convertToShares() function when totalSupply is zero.

This amplification may result in the first depositing user receiving a very small share when making a deposit and failing the _checkMinShares() validation check.

Recommendation

We understand that _checkMinShares() was originally designed to avoid possible inflation attacks. But obviously, this is not the best solution, and we recommend to mint 1 ether for address(0) in the constructor to avoid inflation attacks.

Status

This issue has been acknowledged by the team.

6. Fixed amount of sBTU always remains locked into the contract

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

Description

contracts/StakedBITU.sol:L225-L238

```
function _withdraw(address caller, address receiver, address _owner, uint256 assets,
uint256 shares)
    internal
    override
    nonReentrant
    notZero(assets)
    notZero(shares)
{
    if (hasRole(FULL_RESTRICTED_STAKER_ROLE, caller) ||
hasRole(FULL_RESTRICTED_STAKER_ROLE, receiver)) {
      revert OperationNotAllowed();
    }

    super._withdraw(caller, receiver, _owner, assets, shares);
    _checkMinShares();
}
```

The above code indicates that there is a check for the minimum shares (MIN_SHARES) in the withdraw() function. This means that if the total shares in the contract fall below MIN_SHARES, it won't be possible to successfully withdraw the corresponding amount of assets from the contract, potentially resulting in users not being able to fully withdraw their assets and suffering losses.

Recommendation

We understand that _checkMinShares() was originally designed to avoid possible inflation attacks. But obviously, this is not the best solution, and we recommend to mint 1ether for address(0) in the constructor to avoid inflation attacks.

Status

This issue has been acknowledged by the team.

7. Missing events for functions that change critical state

Severity: Low	Category: Logging
Target: - filepath	

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.

However, the following function is missing events in the code base:

contracts/BitUMinting.sol:L306-L316

```
function setCollateralDistributionRatios(
    address[] memory _custodians,
    uint256[] memory _distributionRatios
)external onlyRole(DEFAULT_ADMIN_ROLE) {
    ...
}
```

contracts/BitUStaking.sol:L78-L90

```
function unstake(address receiver) external {
    ...
}
```

Recommendation

It is recommended to emit events for critical state changes.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

8. Wrong comment or readme

Severity: Informational	Category: Configuration
Target: - contracts/StakedBITU.sol	

Description

In the code base, there are comments or readme files with expositions that don't match the code.

a) contracts/StakedBITU.sol:L101-L127

```
/**
    * @notice Allows the owner (DEFAULT_ADMIN_ROLE) and blacklist managers to blacklist
addresses.
    */
function addToBlacklist(address target, bool isFullBlacklisting)
    external
    onlyRole(BLACKLIST_MANAGER_ROLE)
    notOwner(target)
{
...
}

/**
    * @notice Allows the owner (DEFAULT_ADMIN_ROLE) and blacklist managers to
un-blacklist addresses.
    */
function removeFromBlacklist(address target, bool isFullBlacklisting)
    external
    onlyRole(BLACKLIST_MANAGER_ROLE)
    notOwner(target)
{
...
}
```

In the code above, DEFAULT_ADMIN_ROLE does not have permission to call the function.

b) contracts/README.md

```
- **SOFT_RESTRICTED_STAKER_ROLE**: For addresses from countries where yield distribution
is prohibited (e.g., the USA), restricting them from depositing BITU for sBITU or
withdrawing sBITU for BITU. However, they can trade sBITU on the open market.
```

The SOFT_RESTRICTED_STAKER_ROLE is still available for use withdraw functionality in the code base.

Recommendation

It is recommended that comments and documentation be consistent with the code.

Status

The team has resolved this issue in commit 674a963.

9. Redundant code

Severity: Informational	Category: Redundancy
Target: - contracts/StakedBITU.sol	

Description

contracts/StakedBITU.sol:L89-L99

```
function transferInRewards(uint256 amount) external nonReentrant onlyRole(REWARDER_ROLE)
notZero(amount) {
    if (getUnvestedAmount() > 0) revert StillVesting();
    uint256 newVestingAmount = amount + getUnvestedAmount();

    vestingAmount = newVestingAmount;
    lastDistributionTimestamp = block.timestamp;
    // transfer assets from rewarder to this contract
    IERC20(asset()).safeTransferFrom(msg.sender, address(this), amount);

    emit RewardsReceived(amount, newVestingAmount);
}
```

Highlighting part of the code is unnecessary because he will always be 0

Recommendation

Consider removing redundant code.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 32a877c:

File	SHA-1 hash
BITU.sol	c64dc56a3b59a1da442cb81d6083d50f26484409
BitUMinting.sol	4287505a067e70197b832f4f3d48a6921a79d1ff
BitUStaking.sol	f8fdb89a83086133a0eaf385a10eaf6804b53781
BITUVault.sol	03f40ae23bfe70caac0f459ba9e4f8fad53c8e44
SingleAdminAccessControl.sol	5b2975b41ca556d22faf52ed7426ad0f73afd190
StakedBITU.sol	0cf93bff25dea43682b3bdb7a93386d321cc38af

And we audited the commit 091fbe3 that introduced new features to this repository.

File	SHA-1 hash
BitUMinting.sol	4b4a1da0e3caf735060d82c5d8e6aa14d41a744a