BLOCKHAT
SECURITY

SHEER

Smart Contract Security Audit

Prepared by BlockHat

October 28th, 2023 - November 3rd, 2023

BlockHat.io

contact@blockhat.io

Document Properties

Client	Florim Fluri
Version	0.1
Classification	Public

Scope

File	Hash
Sheer/presale-eth.sol	49e57cc0f09d4d1ca9e5674d7398aabf
Sheer/SHEER.sol	5813a9fc3be23f76a9c725323be870ed

Contacts

COMPANY	CONTACT
BlockHat	contact@blockhat.io

Contents

1 Introduction

Florim Fluri engaged BlockHat to conduct a security assessment on the SHEER beginning on October 28th, 2023 and ending November 3rd, 2023. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About SHEER

Sheer is a utility token of a worksheer platform.

Issuer	Florim Fluri
Website	`https://worksheer.com`
Type	Solidity Smart Contract
Audit Method	Whitebox

1.2 Approach & Methodology

BlockHat used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

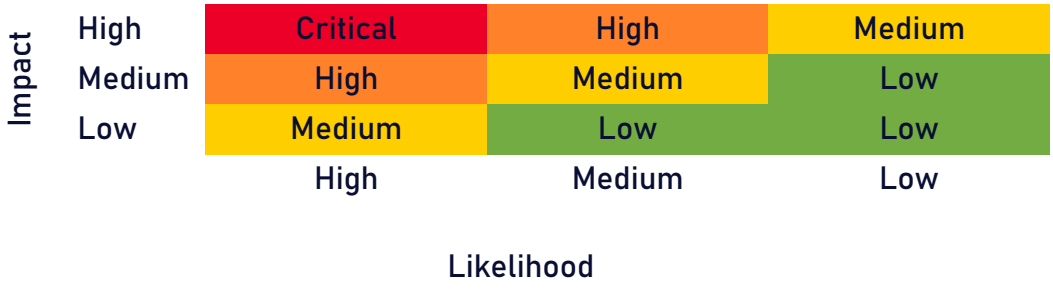
1.2.1 Risk Methodology

Vulnerabilities or bugs identified by BlockHat are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.



2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the SHEER implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include 2 critical-severity, 6 high-severity, 3 medium-severity, 1 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
Manipulable Sale Period by Owner	CRITICAL	Fixed
Inaccurate Token Availability Check	CRITICAL	Not Fixed
Missing Address Zero Checks	HIGH	Fixed
Risk of Hardcoding Token Addresses	HIGH	Fixed
Mutable Token Addresses	HIGH	Fixed
Missing Value Check for usdtAmount	HIGH	Fixed
Missing Value Check for usdcAmount	HIGH	Fixed
High fee limit for both buy and sell fees	HIGH	Fixed
Unchecked Token Transfer	MEDIUM	Not Fixed
Missing Allowance Checks	MEDIUM	Fixed
Use of transfer instead of safeTransfer	MEDIUM	Fixed
Centralization and Owner Privileges	LOW	Acknowledged

3 Finding Details

A presale-eth.sol

A.1 Manipulable Sale Period by Owner [CRITICAL]

Description:

The current mechanism allows the owner to arbitrarily change the start and end times of the presale. This introduces unpredictability and may lead to distrust among participants. Investors and participants rely on predefined and stable sale periods. Allowing the owner to change these parameters in the middle of the sale or just before it begins can harm trust and even lead to potential misuse or front-running.

Code:

Listing 1: presale-eth.sol

```
242     function setStartTime(uint256 _startTime) external onlyOwner {
243         startTime = _startTime;
244     }

246     function setEndTime(uint256 _endTime) external onlyOwner {
247         endTime = _endTime;
248     }
```

Risk Level:

Likelihood – 5

Impact - 5

Recommendation:

Implement a mechanism where the presale starts with a dedicated function like startPresale(), which sets the startTime to the current block timestamp and calculates the endTime

based on a predefined period, Once the presale has started, prohibit any changes to its duration or early termination by the owner. By utilizing this approach, the presale becomes more predictable and transparent, with participants having confidence that the sale duration cannot be manipulated mid-way.

Status - Fixed

A.2 Inaccurate Token Availability Check [CRITICAL]

Description:

The contract checks if there are enough tokens in its balance before allowing a purchase. However, this method doesn't consider tokens that have already been purchased but not yet claimed. Without accounting for these unclaimed tokens, the contract might sell more tokens than it has available, leading to potential issues when users try to claim their tokens.

Code:

Listing 2: presale-eth.sol

```
191         require(amountInTokens <= token.balanceOf(address(this)), "Not
                ↪ enough tokens available");
```

Risk Level:

Likelihood – 4
Impact - 5

Recommendation:

Introduce a new state variable, for example, tokensSold, that accumulates the total tokens sold. Then, modify the require statement to check that tokensSold + amountInTokens is less than or equal to the token balance of the contract. This ensures that the contract only sells tokens that are actually available.

Status - Not Fixed

The addition of the tokensSold = tokensSold.add(amountInTokens); line in the buyTokensWithUSDT function is a fix that ensures the contract doesn't sell more tokens than available. However, it appears that the corresponding buyTokensWithUSDC function has not been updated with a similar line of code. This omission could lead to the same issue of overselling tokens, as the tokensSold variable won't accurately reflect the total number of tokens committed after a USDC transaction. To maintain consistency and ensure accurate tracking of token sales, the same line should be integrated into the buyTokensWithUSDC function. This will prevent any potential discrepancies between the tokens sold and the contract's balance, thereby safeguarding the integrity of the token sale process.

A.3 Missing Address Zero Checks [HIGH]

Description:

The constructor initializes the token, usdtToken, and usdcToken with addresses without checking if they are the zero address. Using a zero address can render the contract non-functional.

Code:

Listing 3: presale-eth.sol

```
166     constructor(address _tokenAddress, address _usdtTokenAddress,
           ↪ address _usdcTokenAddress, uint256 _usdPrice, uint256
           ↪ _startTime, uint256 _endTime) {
167         token = IERC20(_tokenAddress);
168         usdtToken = IERC20(_usdtTokenAddress);
169         usdcToken = IERC20(_usdcTokenAddress);
170         usdRate = _usdPrice;
171         startTime = _startTime;
172         endTime = _endTime;
173         _transferOwnership(msg.sender);
174     }
```

Risk Level:

Likelihood – 3
Impact - 4

Recommendation:

Always check if provided addresses are not zero and correct before initializing contract state variables.

Status - Fixed

A.4 Risk of Hardcoding Token Addresses [HIGH]

Description:

Smart contracts are immutable once deployed. Mistakes made during deployment cannot be corrected without redeploying the entire contract. In the provided presale contract, the addresses for token, usdtToken, and usdcToken are provided as parameters during deployment. This introduces a risk: an inadvertent error during deployment (such as copying and pasting the wrong address or making a typo) can assign the wrong address to these crucial variables, leading to a malfunctioning or non-functional contract.

When dealing with well-known tokens like USDT and USDC, their contract addresses on specific blockchains are fixed and widely recognized. For instance, on the Ethereum mainnet, the address for USDT has been the same since its deployment. Thus, the benefit of dynamically setting such addresses is outweighed by the potential risk of human error.

Code:

Listing 4: presale-eth.sol

```
166     constructor(address _tokenAddress, address _usdtTokenAddress,
            ↪ address _usdcTokenAddress, uint256 _usdPrice, uint256
            ↪ _startTime, uint256 _endTime) {
167         token = IERC20(_tokenAddress);
168         usdtToken = IERC20(_usdtTokenAddress);
```

```
169         usdcToken = IERC20(_usdcTokenAddress);
170         usdRate = _usdPrice;
171         startTime = _startTime;
172         endTime = _endTime;
173         _transferOwnership(msg.sender);
174     }
```

Risk Level:

Likelihood – 3
Impact - 4

Recommendation:

For widely recognized tokens with fixed addresses, it is recommended to hardcode these addresses directly into the contract.

Status - Fixed

A.5 Mutable Token Addresses [HIGH]

Description:

The contract allows changing the USDC, USDT and presale token addresses after deployment. This could lead to potential misuse.

Code:

Listing 5: presale-eth.sol

```
250     function setTokenContract(address newTokenAddress) external
            ↪ onlyOwner {
251         require(newTokenAddress != address(0), "Token contract address
                ↪ cannot be zero");
252         token = IERC20(newTokenAddress);
253     }
```

```
255     function setUsdtTokenContract(address newUsdtTokenAddress) external
            ↪ onlyOwner {
256         require(newUsdtTokenAddress != address(0), "USDT Token contract
                ↪ address cannot be zero");
257         usdtToken = IERC20(newUsdtTokenAddress);
258     }

260     function setUsdcTokenContract(address newUsdcTokenAddress) external
            ↪ onlyOwner {
261         require(newUsdcTokenAddress != address(0), "USDC Token contract
                ↪ address cannot be zero");
262         usdcToken = IERC20(newUsdcTokenAddress);
263     }
```

Risk Level:

Likelihood – 4
Impact - 3

Recommendation:

Fix the token addresses and don't allow changes after contract deployment.

Status - Fixed

A.6 Missing Value Check for usdtAmount [HIGH]

Description:

The function buyTokensWithUSDT does not check whether the passed usdtAmount is greater than 0. This can allow users to call the function with 0 USDT, which would unnecessarily consume gas without any real transaction.

Code:

Listing 6: presale-eth.sol

```
186     function buyTokensWithUSDT(uint256 usdtAmount) external
            ↪ onlyDuringSale {

188         uint256 amountInUsdt = usdtAmount;
189         uint256 amountInTokens = amountInUsdt.mul(usdRate).mul(1e18).div
                ↪ (1e6);

191         require(amountInTokens <= token.balanceOf(address(this)), "Not
                ↪ enough tokens available");
192         require(usdtToken.transferFrom(msg.sender, owner(), amountInUsdt)
                ↪ , "USDT transfer failed");

194         purchasedAmounts[msg.sender] = purchasedAmounts[msg.sender].add(
                ↪ amountInTokens);
195         usdRaised = usdRaised.add(amountInUsdt);
196         emit TokensPurchased(msg.sender, amountInTokens);
197     }
```

Risk Level:

Likelihood – 3
Impact - 4

Recommendation:

Add a require statement at the beginning of the function to check that usdtAmount is greater than 0

Status - Fixed

A.7 Missing Value Check for usdcAmount [HIGH]

Description:

The function buyTokensWithUSDC does not check whether the passed usdcAmount is greater than 0. This oversight can allow users to call the function with 0 USDC, leading to unnecessary gas consumption without any meaningful transaction. Additionally, having such a check can prevent potential bugs or unintended behaviors.

Code:

Listing 7: presale-eth.sol

```
199     function buyTokensWithUSDC(uint256 usdcAmount) external
            ↪ onlyDuringSale {

201         uint256 amountInUsdc = usdcAmount;
202         uint256 amountInTokens = amountInUsdc.mul(usdRate).mul(1e18).div
                ↪ (1e6);

204         require(amountInTokens <= token.balanceOf(address(this)), "Not
                ↪ enough tokens available");
205         require(usdcToken.transferFrom(msg.sender, owner(), amountInUsdc)
                ↪ , "USDC transfer failed");

207         purchasedAmounts[msg.sender] = purchasedAmounts[msg.sender].add(
                ↪ amountInTokens);
208         usdRaised = usdRaised.add(amountInUsdc);
209         emit TokensPurchased(msg.sender, amountInTokens);
210     }
```

Risk Level:

Likelihood – 3
Impact - 4

Recommendation:

Add a require statement at the beginning of the function to check that usdcAmount is greater than 0

Status - Fixed

A.8 Unchecked Token Transfer [MEDIUM]

Description:

In the claimTokens function, the transfer method is used without checking its return value or without using a safer version like safeTransfer. The plain transfer method of the ERC20 standard can fail silently without reverting the transaction, leading to potential loss of funds or unintended behaviors.

Code:

Listing 8: presale-eth.sol

```
212     function claimTokens() external onlyAfterSale {
213         require(!hasClaimed[msg.sender], "Tokens have already been
                ↪ claimed");

215         uint256 claimableAmount = purchasedAmounts[msg.sender];
216         require(claimableAmount > 0, "No tokens to claim");

218         token.transfer(msg.sender, claimableAmount);
219         hasClaimed[msg.sender] = true;

221         emit TokensClaimed(msg.sender, claimableAmount);
```

```
222     }
```

Risk Level:

Likelihood – 3
Impact - 4

Recommendation:

Replace the transfer call with a require statement that checks the return value of the transfer, or preferably, use a safeTransfer function from a library like OpenZeppelin's SafeERC20. Here's how you can modify it with a require check

Status - Not Fixed

SafeERC20.safeTransfer from the OpenZeppelin library, which already includes checks and error handling to ensure the safety of token transfers. Unlike the basic transfer method in the ERC20 standard that may fail silently, safeTransfer will revert the transaction if the transfer fails. Therefore, the additional require statement to check the return value of safeTransfer is unnecessary, as safeTransfer will throw an error and revert the whole transaction if the transfer is unsuccessful. This is a redundancy that can be removed to simplify the code without compromising on security or functionality.

A.9 Missing Allowance Checks [MEDIUM]

Description:

Before transferring tokens using transferFrom, it's important to check if the contract has the required allowances. Without this check, the transfer can fail.

Code:

Listing 9: presale-eth.sol

```
192 require(usdtToken.transferFrom(msg.sender, owner(), amountInUsdt), "USDT
        ↪ transfer failed");
```

Listing 10: presale-eth.sol

```
205         require(usdcToken.transferFrom(msg.sender, owner(), amountInUsdc)
                ↪ , "USDC transfer failed");
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

Always check for sufficient allowances before transferring tokens.

Status - Fixed

B Sheer.sol

B.1 High fee limit for both buy and sell fees [HIGH]

Description:

The updateBuyFees and updateSellFees functions both have a condition where the total combined fees (for buy and sell) must be kept at 50% or less. A 50% fee is a substantial amount and might be seen as excessive or unfair by users or investors.

Code:

Listing 11: Sheer.sol

```
663     function updateBuyFees(uint256 _marketingFee, uint256 _liquidityFee)
            ↪  external onlyOwner {
664         liquidityFeeOnBuy = _marketingFee;
665         marketingFeeOnBuy = _liquidityFee;
666         _totalFeesOnBuy = liquidityFeeOnBuy + marketingFeeOnBuy;
667         require(_totalFeesOnBuy <= 50, "Must keep fees at 50% or less");
```

```
668     }

670     function updateSellFees(uint256 _marketingFee, uint256 _liquidityFee
            ↪ ) external onlyOwner {
671         liquidityFeeOnSell = _marketingFee;
672         marketingFeeOnSell = _liquidityFee;
673         _totalFeesOnSell = liquidityFeeOnSell + marketingFeeOnSell;
674         require(_totalFeesOnSell <= 50, "Must keep fees at 50% or less");
675     }
```

Risk Level:

Likelihood – 5
Impact - 5

Recommendation:

It's advisable to reconsider the fee structure and potentially reduce the upper limit. Ensure that the fee structure is transparent to users and justified for the utility it provides.

Status - Fixed

B.2 Use of transfer instead of safeTransfer [MEDIUM]

Description:

The transfer method can silently fail without reverting the transaction, leading to potential loss of funds or unintended behavior.

Code:

Listing 12: Sheer.sol

```
631     function claimStuckTokens(address token) external onlyOwner {
632         require(token != address(this), "Owner cannot claim contract's
                ↪ balance of its own tokens");
```

```
633         if (token == address(0x0)) {
634             payable(msg.sender).sendValue(address(this).balance);
635             return;
636         }
637         IERC20 ERC20token = IERC20(token);
638         uint256 balance = ERC20token.balanceOf(address(this));
639         ERC20token.transfer(msg.sender, balance);
640     }
```

Risk Level:

Likelihood – 3
Impact - 4

Recommendation:

Replace transfer with safeTransfer from a reputable library like OpenZeppelin to ensure transaction reverts on failure.

Status - Fixed

B.3 Centralization and Owner Privileges [LOW]

Description:

The smart contract contains several functions that can only be called by the owner, providing a high degree of centralization. For instance, the owner can:

- Enable or disable token trading
- Change fees
- Exclude accounts from fees
- Claim stuck tokens
- Update max transaction amount and wallet amount
- And more...

Risk Level:

Likelihood – 2
Impact - 2

Recommendation:

Consider introducing decentralized governance or more transparent measures to reduce the trust required by holders in the owner.

Status - Acknowledged

4 Best Practices

BP.1 Lack of Transparency in Token Decimals

Description:

For transparency, it's recommended to use the token's own decimals function instead of hardcoding values.

Recommendation:

Use the decimals function from the token contract to determine the token's precision.

BP.2 Token Transfer Instead of TransferFrom

Description:

It's safer to use transferFrom instead of transfer to move tokens from the contract to users.

Recommendation:

Use transferFrom to transfer tokens to users.

BP.3 Update State Before External Calls

Description:

For safety, state variables should be updated before making external calls.

Recommendation:

Update state variables before making any external calls.

BP.4 Missing Event for setUsdRate

Description:

The setUsdRate function updates the usdRate state variable but does not emit an event to log the change. Emitting events for state changes is a best practice in Ethereum smart contracts as it provides transparency and allows easy tracking of contract operations.

Recommendation:

Introduce a new event, for example, UsdRateUpdated, and emit this event after updating the usdRate variable

5 Static Analysis (Slither)

Description:

Block Hat expanded the coverage of the specific contract areas using automated testing methodologies. Slither, a Solidity static analysis framework, was one of the tools used. Slither was run on all-scoped contracts in both text and binary formats. This tool can be used to test mathematical relationships between Solidity instances statically and variables that allow for the detection of errors or inconsistent usage of the contracts' APIs throughout the entire codebase.

Results:

```
INFO:Detectors:
Reentrancy in SHEER._transfer(address,address,uint256) (SHEER.sol
    ↪ #695-777):
        External calls:
        - swapAndLiquify(liquidityTokens) (SHEER.sol#750)
                - uniswapV2Router.
                    ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                    ↪ half,0,path,address(this),block.timestamp) (SHEER.
                    ↪ sol#816-821)
                - uniswapV2Router.addLiquidityETH{value: newBalance}(
                    ↪ address(this),otherHalf,0,0,address(0xdead),block.
                    ↪ timestamp) (SHEER.sol#825-832)
        - swapAndSendMarketing(marketingTokens) (SHEER.sol#755)
                - (success) = recipient.call{value: amount}() (SHEER.sol
                    ↪ #234)
                - uniswapV2Router.
                    ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                    ↪ tokenAmount,0,path,address(this),block.timestamp) (
                    ↪ SHEER.sol#844-849)
                - address(marketingWallet).sendValue(newBalance) (SHEER.
                    ↪ sol#853)
```

```
        External calls sending eth:
        - swapAndLiquify(liquidityTokens) (SHEER.sol#750)
              - uniswapV2Router.addLiquidityETH{value: newBalance}(
                 ↪ address(this),otherHalf,0,0,address(0xdead),block.
                 ↪ timestamp) (SHEER.sol#825-832)
        - swapAndSendMarketing(marketingTokens) (SHEER.sol#755)
              - (success) = recipient.call{value: amount}() (SHEER.sol
                 ↪ #234)
        State variables written after the call(s):
        - super._transfer(from,address(this),fees) (SHEER.sol#773)
              - _balances[sender] = senderBalance - amount (SHEER.sol
                 ↪ #478)
              - _balances[recipient] += amount (SHEER.sol#480)
        ERC20._balances (SHEER.sol#384) can be used in cross function
           ↪ reentrancies:
        - ERC20._mint(address,uint256) (SHEER.sol#487-497)
        - ERC20._transfer(address,address,uint256) (SHEER.sol#465-485)
        - ERC20.balanceOf(address) (SHEER.sol#414-416)
        - super._transfer(from,to,amount) (SHEER.sol#776)
              - _balances[sender] = senderBalance - amount (SHEER.sol
                 ↪ #478)
              - _balances[recipient] += amount (SHEER.sol#480)
        ERC20._balances (SHEER.sol#384) can be used in cross function
           ↪ reentrancies:
        - ERC20._mint(address,uint256) (SHEER.sol#487-497)
        - ERC20._transfer(address,address,uint256) (SHEER.sol#465-485)
        - ERC20.balanceOf(address) (SHEER.sol#414-416)
        - swapping = false (SHEER.sol#758)
        SHEER.swapping (SHEER.sol#569) can be used in cross function
           ↪ reentrancies:
        - SHEER._transfer(address,address,uint256) (SHEER.sol#695-777)
    Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
       ↪ #reentrancy-vulnerabilities
    INFO:Detectors:
```

```
SHEER.claimStuckTokens(address) (SHEER.sol#631-640) ignores return value
    ↪  by ERC20token.transfer(msg.sender,balance) (SHEER.sol#639)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #unchecked-transfer
INFO:Detectors:
SHEER._transfer(address,address,uint256)._totalFees (SHEER.sol#761) is a
    ↪  local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #uninitialized-local-variables
INFO:Detectors:
SHEER.swapAndLiquify(uint256) (SHEER.sol#806-835) ignores return value
    ↪ by uniswapV2Router.addLiquidityETH{value: newBalance}(address(
    ↪ this),otherHalf,0,0,address(0xdead),block.timestamp) (SHEER.sol
    ↪ #825-832)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #unused-return
INFO:Detectors:
SHEER.updateBuyFees(uint256,uint256) (SHEER.sol#663-668) should emit an
    ↪ event for:
        - liquidityFeeOnBuy = _marketingFee (SHEER.sol#664)
        - marketingFeeOnBuy = _liquidityFee (SHEER.sol#665)
        - _totalFeesOnBuy = liquidityFeeOnBuy + marketingFeeOnBuy (SHEER.
            ↪ sol#666)
SHEER.updateSellFees(uint256,uint256) (SHEER.sol#670-675) should emit an
    ↪  event for:
        - liquidityFeeOnSell = _marketingFee (SHEER.sol#671)
        - marketingFeeOnSell = _liquidityFee (SHEER.sol#672)
        - _totalFeesOnSell = liquidityFeeOnSell + marketingFeeOnSell (
            ↪ SHEER.sol#673)
SHEER.updateMaxTxnAmount(uint256) (SHEER.sol#792-795) should emit an
    ↪ event for:
        - maxTransactionAmount = newAmount (SHEER.sol#794)
SHEER.updateMaxWalletAmount(uint256) (SHEER.sol#797-800) should emit an
    ↪ event for:
```

```
        - maxWallet = newAmount (SHEER.sol#799)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #missing-events-arithmetic
INFO:Detectors:
Reentrancy in SHEER._transfer(address,address,uint256) (SHEER.sol
    ↪ #695-777):
        External calls:
        - swapAndLiquify(liquidityTokens) (SHEER.sol#750)
              - uniswapV2Router.
                  ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                  ↪ half,0,path,address(this),block.timestamp) (SHEER.
                  ↪ sol#816-821)
              - uniswapV2Router.addLiquidityETH{value: newBalance}(
                  ↪ address(this),otherHalf,0,0,address(0xdead),block.
                  ↪ timestamp) (SHEER.sol#825-832)
        - swapAndSendMarketing(marketingTokens) (SHEER.sol#755)
              - (success) = recipient.call{value: amount}() (SHEER.sol
                  ↪ #234)
              - uniswapV2Router.
                  ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                  ↪ tokenAmount,0,path,address(this),block.timestamp) (
                  ↪ SHEER.sol#844-849)
              - address(marketingWallet).sendValue(newBalance) (SHEER.
                  ↪ sol#853)
        External calls sending eth:
        - swapAndLiquify(liquidityTokens) (SHEER.sol#750)
              - uniswapV2Router.addLiquidityETH{value: newBalance}(
                  ↪ address(this),otherHalf,0,0,address(0xdead),block.
                  ↪ timestamp) (SHEER.sol#825-832)
        - swapAndSendMarketing(marketingTokens) (SHEER.sol#755)
              - (success) = recipient.call{value: amount}() (SHEER.sol
                  ↪ #234)
        Event emitted after the call(s):
        - SwapAndSendMarketing(tokenAmount,newBalance) (SHEER.sol#855)
```

```
                - swapAndSendMarketing(marketingTokens) (SHEER.sol#755)
        - Transfer(sender,recipient,amount) (SHEER.sol#482)
                - super._transfer(from,address(this),fees) (SHEER.sol#773)
        - Transfer(sender,recipient,amount) (SHEER.sol#482)
                - super._transfer(from,to,amount) (SHEER.sol#776)
Reentrancy in SHEER.swapAndLiquify(uint256) (SHEER.sol#806-835):
        External calls:
        - uniswapV2Router.
            ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(half,0,
            ↪ path,address(this),block.timestamp) (SHEER.sol#816-821)
        - uniswapV2Router.addLiquidityETH{value: newBalance}(address(this
            ↪ ),otherHalf,0,0,address(0xdead),block.timestamp) (SHEER.
            ↪ sol#825-832)
        External calls sending eth:
        - uniswapV2Router.addLiquidityETH{value: newBalance}(address(this
            ↪ ),otherHalf,0,0,address(0xdead),block.timestamp) (SHEER.
            ↪ sol#825-832)
        Event emitted after the call(s):
        - SwapAndLiquify(half,newBalance,otherHalf) (SHEER.sol#834)
Reentrancy in SHEER.swapAndSendMarketing(uint256) (SHEER.sol#837-856):
        External calls:
        - uniswapV2Router.
            ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
            ↪ tokenAmount,0,path,address(this),block.timestamp) (SHEER.
            ↪ sol#844-849)
        - address(marketingWallet).sendValue(newBalance) (SHEER.sol#853)
        Event emitted after the call(s):
        - SwapAndSendMarketing(tokenAmount,newBalance) (SHEER.sol#855)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-3
INFO:Detectors:
Address._revert(bytes,string) (SHEER.sol#325-337) uses assembly
        - INLINE ASM (SHEER.sol#330-333)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #assembly-usage
INFO:Detectors:
SHEER._transfer(address,address,uint256) (SHEER.sol#695-777) has a high
    ↪ cyclomatic complexity (13).
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #cyclomatic-complexity
INFO:Detectors:
Address._revert(bytes,string) (SHEER.sol#325-337) is never used and
    ↪ should be removed
Address.functionCall(address,bytes) (SHEER.sol#238-240) is never used
    ↪ and should be removed
Address.functionCall(address,bytes,string) (SHEER.sol#242-248) is never
    ↪ used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (SHEER.sol#250-256)
    ↪  is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (SHEER.sol
    ↪ #258-267) is never used and should be removed
Address.functionDelegateCall(address,bytes) (SHEER.sol#282-284) is never
    ↪  used and should be removed
Address.functionDelegateCall(address,bytes,string) (SHEER.sol#286-293)
    ↪ is never used and should be removed
Address.functionStaticCall(address,bytes) (SHEER.sol#269-271) is never
    ↪ used and should be removed
Address.functionStaticCall(address,bytes,string) (SHEER.sol#273-280) is
    ↪ never used and should be removed
Address.isContract(address) (SHEER.sol#227-229) is never used and should
    ↪  be removed
Address.verifyCallResult(bool,bytes,string) (SHEER.sol#313-323) is never
    ↪  used and should be removed
Address.verifyCallResultFromTarget(address,bool,bytes,string) (SHEER.sol
    ↪ #295-311) is never used and should be removed
Context._msgData() (SHEER.sol#345-348) is never used and should be
    ↪ removed
```

```
ERC20._burn(address,uint256) (SHEER.sol#499-514) is never used and
    ↪ should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dead-code
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (SHEER.sol#231-236)
    ↪ :
        - (success) = recipient.call{value: amount}() (SHEER.sol#234)
Low level call in Address.functionCallWithValue(address,bytes,uint256,
    ↪ string) (SHEER.sol#258-267):
        - (success,returndata) = target.call{value: value}(data) (SHEER.
            ↪ sol#265)
Low level call in Address.functionStaticCall(address,bytes,string) (
    ↪ SHEER.sol#273-280):
        - (success,returndata) = target.staticcall(data) (SHEER.sol#278)
Low level call in Address.functionDelegateCall(address,bytes,string) (
    ↪ SHEER.sol#286-293):
        - (success,returndata) = target.delegatecall(data) (SHEER.sol
            ↪ #291)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #low-level-calls
INFO:Detectors:
Function IUniswapV2Pair.DOMAIN_SEPARATOR() (SHEER.sol#33) is not in
    ↪ mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (SHEER.sol#34) is not in
    ↪ mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (SHEER.sol#51) is not in
    ↪ mixedCase
Function IUniswapV2Router01.WETH() (SHEER.sol#71) is not in mixedCase
Parameter SHEER.changeMarketingWallet(address)._marketingWallet (SHEER.
    ↪ sol#655) is not in mixedCase
Parameter SHEER.updateBuyFees(uint256,uint256)._marketingFee (SHEER.sol
    ↪ #663) is not in mixedCase
```

```
Parameter SHEER.updateBuyFees(uint256,uint256)._liquidityFee (SHEER.sol
    ↪ #663) is not in mixedCase
Parameter SHEER.updateSellFees(uint256,uint256)._marketingFee (SHEER.sol
    ↪ #670) is not in mixedCase
Parameter SHEER.updateSellFees(uint256,uint256)._liquidityFee (SHEER.sol
    ↪ #670) is not in mixedCase
Parameter SHEER.setSwapEnabled(bool)._enabled (SHEER.sol#781) is not in
    ↪ mixedCase
Variable SHEER._isExcludedMaxTransactionAmount (SHEER.sol#549) is not in
    ↪  mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (SHEER.sol#346)" inContext (SHEER.sol
    ↪ #340-349)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #redundant-statements
INFO:Detectors:
Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256
    ↪ ,uint256,uint256,address,uint256).amountADesired (SHEER.sol#76)
    ↪ is too similar to IUniswapV2Router01.addLiquidity(address,address
    ↪ ,uint256,uint256,uint256,uint256,address,uint256).amountBDesired
    ↪ (SHEER.sol#77)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #variable-names-too-similar
INFO:Detectors:
SHEER.constructor() (SHEER.sol#583-625) uses literals with too many
    ↪ digits:
        - _mint(owner(),200000000000e18) (SHEER.sol#617)
SHEER.setSwapTokensAtAmount(uint256) (SHEER.sol#786-790) uses literals
    ↪ with too many digits:
        - require(bool,string)(newAmount > totalSupply() / 1000000,
            ↪ SwapTokensAtAmount must be greater than 0.0001% of total
            ↪ supply) (SHEER.sol#787)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #too-many-digits
INFO:Detectors:
SHEER.uniswapV2Pair (SHEER.sol#545) should be immutable
SHEER.uniswapV2Router (SHEER.sol#544) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #state-variables-that-could-be-declared-immutable
INFO:Detectors:
Presale.claimTokens() (presale-eth.sol#212-222) ignores return value by
    ↪ token.transfer(msg.sender,claimableAmount) (presale-eth.sol#218)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #unchecked-transfer
INFO:Detectors:
Reentrancy in Presale.claimTokens() (presale-eth.sol#212-222):
        External calls:
        - token.transfer(msg.sender,claimableAmount) (presale-eth.sol
            ↪ #218)
        State variables written after the call(s):
        - hasClaimed[msg.sender] = true (presale-eth.sol#219)
        Presale.hasClaimed (presale-eth.sol#160) can be used in cross
            ↪ function reentrancies:
        - Presale.canClaimTokens(address) (presale-eth.sol#233-235)
        - Presale.claimTokens() (presale-eth.sol#212-222)
        - Presale.getClaimableTokens(address) (presale-eth.sol#225-231)
        - Presale.hasClaimed (presale-eth.sol#160)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-1
INFO:Detectors:
Presale.setUsdRate(uint256) (presale-eth.sol#237-240) should emit an
    ↪ event for:
        - usdRate = newUsdRate (presale-eth.sol#239)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #missing-events-arithmetic
INFO:Detectors:
```

```
Reentrancy in Presale.buyTokensWithUSDC(uint256) (presale-eth.sol
    ↪ #199-210):
        External calls:
        - require(bool,string)(usdcToken.transferFrom(msg.sender,owner(),
            ↪ amountInUsdc),USDC transfer failed) (presale-eth.sol#205)
        State variables written after the call(s):
        - purchasedAmounts[msg.sender] = purchasedAmounts[msg.sender].add
            ↪ (amountInTokens) (presale-eth.sol#207)
        - usdRaised = usdRaised.add(amountInUsdc) (presale-eth.sol#208)
Reentrancy in Presale.buyTokensWithUSDT(uint256) (presale-eth.sol
    ↪ #186-197):
        External calls:
        - require(bool,string)(usdtToken.transferFrom(msg.sender,owner(),
            ↪ amountInUsdt),USDT transfer failed) (presale-eth.sol#192)
        State variables written after the call(s):
        - purchasedAmounts[msg.sender] = purchasedAmounts[msg.sender].add
            ↪ (amountInTokens) (presale-eth.sol#194)
        - usdRaised = usdRaised.add(amountInUsdt) (presale-eth.sol#195)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in Presale.buyTokensWithUSDC(uint256) (presale-eth.sol
    ↪ #199-210):
        External calls:
        - require(bool,string)(usdcToken.transferFrom(msg.sender,owner(),
            ↪ amountInUsdc),USDC transfer failed) (presale-eth.sol#205)
        Event emitted after the call(s):
        - TokensPurchased(msg.sender,amountInTokens) (presale-eth.sol
            ↪ #209)
Reentrancy in Presale.buyTokensWithUSDT(uint256) (presale-eth.sol
    ↪ #186-197):
        External calls:
        - require(bool,string)(usdtToken.transferFrom(msg.sender,owner(),
            ↪ amountInUsdt),USDT transfer failed) (presale-eth.sol#192)
```

```
        Event emitted after the call(s):
        - TokensPurchased(msg.sender,amountInTokens) (presale-eth.sol
            ↪ #196)
Reentrancy in Presale.claimTokens() (presale-eth.sol#212-222):
        External calls:
        - token.transfer(msg.sender,claimableAmount) (presale-eth.sol
            ↪ #218)
        Event emitted after the call(s):
        - TokensClaimed(msg.sender,claimableAmount) (presale-eth.sol#221)
Reentrancy in Presale.recoverWrongTokens(address) (presale-eth.sol
    ↪ #271-279):
        External calls:
        - require(bool,string)(wrongToken.transfer(owner(),balance),Token
            ↪  recovery failed) (presale-eth.sol#276)
        Event emitted after the call(s):
        - TokensRecovered(msg.sender,_tokenAddress,balance) (presale-eth.
            ↪ sol#278)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-3
INFO:Detectors:
Context._msgData() (presale-eth.sol#91-93) is never used and should be
    ↪ removed
SafeMath.div(uint256,uint256,string) (presale-eth.sol#71-76) is never
    ↪ used and should be removed
SafeMath.mod(uint256,uint256) (presale-eth.sol#60-62) is never used and
    ↪ should be removed
SafeMath.mod(uint256,uint256,string) (presale-eth.sol#78-83) is never
    ↪ used and should be removed
SafeMath.sub(uint256,uint256) (presale-eth.sol#48-50) is never used and
    ↪ should be removed
SafeMath.sub(uint256,uint256,string) (presale-eth.sol#64-69) is never
    ↪ used and should be removed
SafeMath.tryAdd(uint256,uint256) (presale-eth.sol#6-12) is never used
    ↪ and should be removed
```

```
SafeMath.tryDiv(uint256,uint256) (presale-eth.sol#30-35) is never used
   ↪ and should be removed
SafeMath.tryMod(uint256,uint256) (presale-eth.sol#37-42) is never used
   ↪ and should be removed
SafeMath.tryMul(uint256,uint256) (presale-eth.sol#21-28) is never used
   ↪ and should be removed
SafeMath.trySub(uint256,uint256) (presale-eth.sol#14-19) is never used
   ↪ and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #dead-code
INFO:Detectors:
Low level call in Presale.withdrawFunds() (presale-eth.sol#265-269):
       - (success) = address(msg.sender).call{value: ethBalance}() (
          ↪ presale-eth.sol#267)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #low-level-calls
INFO:Detectors:
Parameter Presale.setStartTime(uint256)._startTime (presale-eth.sol#242)
   ↪  is not in mixedCase
Parameter Presale.setEndTime(uint256)._endTime (presale-eth.sol#246) is
   ↪ not in mixedCase
Parameter Presale.recoverWrongTokens(address)._tokenAddress (presale-eth
   ↪ .sol#271) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #conformance-to-solidity-naming-conventions
INFO:Detectors:
Variable Presale.constructor(address,address,address,uint256,uint256,
   ↪ uint256)._usdcTokenAddress (presale-eth.sol#166) is too similar
   ↪ to Presale.constructor(address,address,address,uint256,uint256,
   ↪ uint256)._usdtTokenAddress (presale-eth.sol#166)
Variable Presale.buyTokensWithUSDC(uint256).amountInUsdc (presale-eth.
   ↪ sol#201) is too similar to Presale.buyTokensWithUSDT(uint256).
   ↪ amountInUsdt (presale-eth.sol#188)
```

```
Variable Presale.setUsdcTokenContract(address).newUsdcTokenAddress (
    ↪ presale-eth.sol#260) is too similar to Presale.
    ↪ setUsdtTokenContract(address).newUsdtTokenAddress (presale-eth.
    ↪ sol#255)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #variable-names-too-similar
INFO:Slither:. analyzed (16 contracts with 85 detectors), 75 result(s)
    ↪ found
```

Conclusion:

Most of the vulnerabilities found by the analysis have already been addressed by the smart contract code review.

6 Conclusion

In this audit, we examined the design and implementation of SHEER contract and discovered several issues of varying severity. Florim Fluri team addressed issues raised in the initial report and implemented the necessary fixes, while classifying the rest as a risk with low-probability of occurrence. Blockhat auditors advised Florim Fluri Team to maintain a high level of vigilance and to keep those findings in mind in order to avoid any future complications.

BLOCKHAT
SECURITY

For a Smart Contract Audit, contact us at contact@blockhat.io