LATTESWAP SECURITY ASSESSMENT REPORT

NOV. 18 ~ DEC. 01, 2021

DISCLAIMER

• This document is based on a security assessment conducted by a blockchain security company SOOHO. This document describes the detected security vulnerabilities and also discusses the code quality and code license violations.

• This security assessment does not guarantee nor describe the usefulness of the code, the stability of the code, the suitability of the business model, the legal regulation of the business, the suitability of the contract, and the bug-free status. Audit document is used for discussion purposes only.

• SOOHO does not disclose any business information obtained during the review or save it through a separate media.

• SOOHO presents its best endeavors in smart contract security assessment.

SOOHO

SOOHO with the motto of "Audit Everything, Automatically" researches and provides technology for reliable blockchain ecosystem. SOOHO verifies vulnerabilities through entire development life-cycle with Aegis, a vulnerability analyzer created by SOOHO, and open source analyzers. SOOHO is composed of experts including Ph.D researchers in the field of automated security tools and white-hackers verifying contract codes and detected vulnerabilities in depth. Professional experts in SOOHO secure partners' contracts from known to zero-day vulnerabilities.

INTRODUCTION

SOOHO conducted a security assessment of LatteSwap's smart contract from Nov. 18 until Dec. 01, 2021. The following tasks were performed during the audit period:

• Performing and analyzing the results of Odin, a static analyzer of SOOHO.

• Writing Exploit codes on suspected vulnerability in the contract.

• Recommendations on codes based on best practices and the Secure Coding Guide.

Our security experts participated in a vulnerability analysis of the contract. The experts are professional hackers with Ph.D. academic backgrounds and experiences of receiving awards from national/international hacking competitions such as Defcon, Nuit du Hack, White Hat, SamsungCTF, and etc.

The detected vulnerabilities are as follows: Note 2. It is recommended to promote the stability of service through continuous code audit and analyze potential vulnerabilities.

ANALYSIS TARGET

The following projects were analyzed during period.

Project	`flat-contract/v8`
Commit #	da97c4bf
# of Files	40
# of Lines	4,167

KEY AUDIT POINTS & PROCESS

LatteSwap's flat-contract is a Defi service that contains collateral lending market and yield farming strategies. Each component works with underlying assets called FLAT tokens. Accordingly, we mainly reviewed issues that may occur in the tokens, interest accrues logics, liquidation kills, borrowing, and repay.

However, we did not take any internal hackings by administrators and the price stabilization into account. Analyzes are about the functioning of the subject contract, given the safety of the system.

1 **Automated Vulnerability Analysis**

2 **Manual Code Analysis**

The followings are considered:

- Preferential analysis of codes with greater risks.
- Supervision of Access Control management.
- Analyze whether the code is written under the client’s intention.

3 **Review of Exploitability and PoC Code**

The followings are considered:

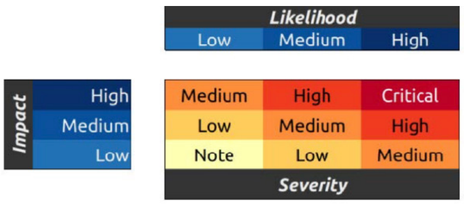
- Dynamic analysis through code execution.
- Examine possible financial gain by misusing detected vulnerabilities. (e.g., infinite withdrawal)
- Examine the possibility of adverse effect to the token service by misusing detected vulnerabilities.

RISK RATING OF VULNERABILITY

Detected vulnerabilities are listed on the basis of the risk rating of vulnerability.

Critical High Medium Low Note

The risk rating of vulnerability is set based on OWASP’s Impact & Likelihood Risk Rating Methodology as seen on the right. Some issues were rated vulnerable aside from the corresponding model and the reasons are explained in the following results.



Impact \ Likelihood	Low	Medium	High
High	Medium	High	Critical
Medium	Low	Medium	High
Low	Note	Low	Medium

Severity

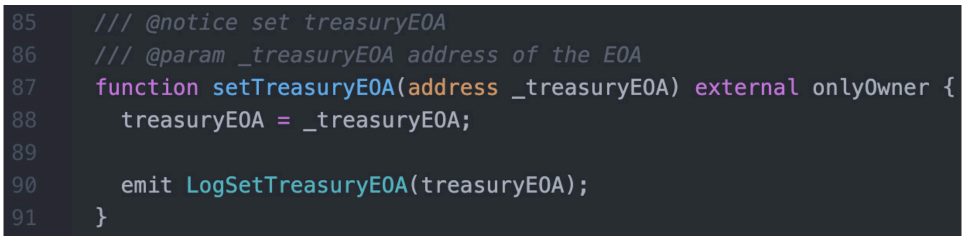
ANALYSIS RESULTS

Analysis results are categorized into Critical, High, Medium, Low, and Note. SOOHO recommends upgrades on every detected issue.

TREASURY EOA CAN BE NULL Note

Additional resources and comments

File Name : `TreasuryHolder.sol`
File Location : `flat-contract/contracts/v8`
`└── TreasuryHolder.sol`
MD5 : 8725a7ed8bce56db56e21f6210c718f3



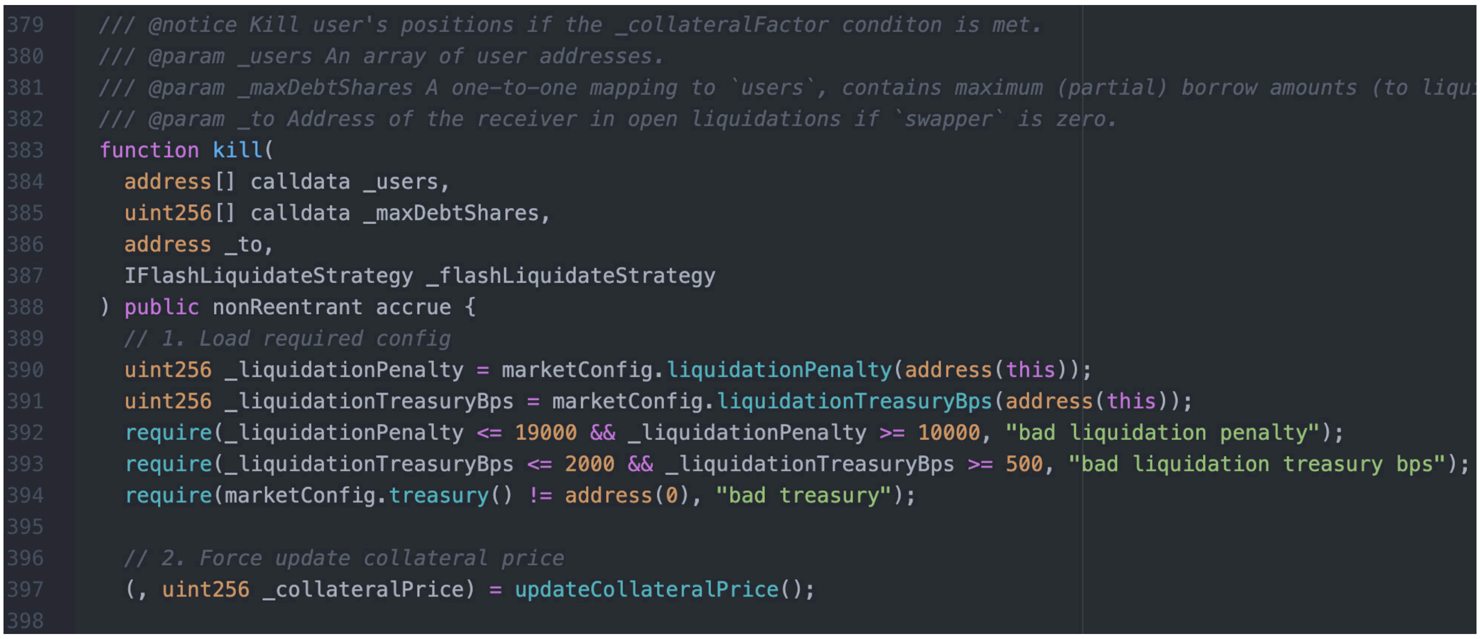
```
/// @notice set treasuryEOA
/// @param _treasuryEOA address of the EOA
function setTreasuryEOA(address _treasuryEOA) external onlyOwner {
  treasuryEOA = _treasuryEOA;

  emit LogSetTreasuryEOA(treasuryEOA);
}
```

Details `treasuryEOA` can be null due to a lack of validation of parameters. We recommend adding `require` statements before assigning.

COLLATERAL PRICE SHOULD UPDATED Note

Additional resources and comments

File Name : `FlatMarket.sol`
File Location : `flat-contract/contracts/v8/`
`└── FlatMarket.sol`
MD5 : 9981fbc96bc7b82de9c12a1038db0d98



```
/// @notice Kill user's positions if the _collateralFactor conditon is met.
/// @param _users An array of user addresses.
/// @param _maxDebtShares A one-to-one mapping to `users`, contains maximum (partial) borrow amounts (to liqu
/// @param _to Address of the receiver in open liquidations if `swapper` is zero.
function kill(
  address[] calldata _users,
  uint256[] calldata _maxDebtShares,
  address _to,
  IFlashLiquidateStrategy _flashLiquidateStrategy
) public nonReentrant accrue {
  // 1. Load required config
  uint256 _liquidationPenalty = marketConfig.liquidationPenalty(address(this));
  uint256 _liquidationTreasuryBps = marketConfig.liquidationTreasuryBps(address(this));
  require(_liquidationPenalty <= 19000 && _liquidationPenalty >= 10000, "bad liquidation penalty");
  require(_liquidationTreasuryBps <= 2000 && _liquidationTreasuryBps >= 500, "bad liquidation treasury bps");
  require(marketConfig.treasury() != address(0), "bad treasury");

  // 2. Force update collateral price
  (, uint256 _collateralPrice) = updateCollateralPrice();

```

Details During the execution of the `kill` function, to check the safety of the user positions, it forces updates the collateral price. User safety will be calculated with the cached value if oracle could not return the proper values. We recommend running the `kill` function only if the `updateCollateralPrice` returns `_updated` as true.

ANALYSIS RESULTS

Analysis results are categorized into Critical, High, Medium, Low, and Note. SOOHO recommends upgrades on every detected issue.

WELL MANAGED SUPPLY ✓

File Name : `FLAT.sol`
File Location : `flat-contract/contracts/v8`
`└── FLAT.sol`
MD5 : 62105d74ec31083c23046096fcdbb2aa

Details The implementation of the FLAT has been verified to be correct. Especially, minting logics are guarded by MintRange and MintBps. We have tested the logic based on the detailed comments and we have confirmed that the codes work properly.

Additional resources and comments

MARKET HAVE VERIFIED ✓

File Name : `FlatMarket.sol`
File Location : `flat-contract/contracts/v8`
`└── FlatMarket.sol`
MD5 : 9981fbc96bc7b82de9c12a1038db0d98

Details According to the system, Baristas in the LatteSwap can be borrowed, repaid, add/remove collateral, and earn interest through the FlatMarket. We have reviewed the functionalities in the contract and we have confirmed the logic implemented well.

Additional resources and comments

LIQUIDATIONS ✓

File Name : `LatteSwapLiquidationStrategy.sol`
File Location : `flat-contract/contracts/v8/strategies/liquidation`
`└── LatteSwapLiquidationStrategy.sol`
MD5 : 95004d78c3ccebcc9cb34152f2a26549

Details We have confirmed that the liquidation operations are safe. We assumed that anyone can participate in liquidation process.

Additional resources and comments

ANALYSIS RESULTS

Analysis results are categorized into Critical, High, Medium, Low, and Note. SOOHO recommends upgrades on every detected issue.

YIELD FARMING STRATEGIES ✓

File Name : `LatteSwapYieldStrategy.sol`

File Location : `flat-contract/contracts/v8/strategies/yield`
```
└── LatteSwapYieldStrategy.sol
```
MD5 : 11afa1572b91e0b917471437d014c76e

Details We have reviewed the yield farming strategies including PCS and LatteSwap. Each of the strategy works well.

Additional resources and comments

ORACLE ALGORITHMS ✓

File Name : `CompositeOracle.sol`

File Location : `flat-contract/contracts/v8/oracles`
```
└── CompositeOracle.sol
```
MD5 : 3d28949d95083402bdf0f239295f45d9

Details We have confirmed that the system uses proper oracles including Chainlink and Off-chain oracles. Moreover, the contract composites several oracles for the stabilized system. We have confirmed the selection logics of the oracles.

Additional resources and comments

UPGRADABILTIES ✓

Details We have confirmed that the contracts construct properly with upgradability.

Additional resources and comments

OVERFLOWS ✓

Details We have confirmed that the contract uses solidity v0.8.9 which is natively guarded against the overflow. Even overflow intentionally uses, each of the statements guarded well.

Additional resources and comments

CONCLUSION

The source code of the flat-contract developed by LatteSwap is easy to read and very well organized. We have to remark that contracts are well architected and all the additional features are implemented. **The detected issues are as follows: Note 2.** However, most of the codes are found out to be compliant with all the best practices. It is recommended to promote the stability of service through continuous code audit and analyze potential vulnerabilities.

Project	flat-contract/v8
Commit #	da97c4bf
# of Files	40
# of Lines	4,167

File Tree

```
flat-contract/contracts/v8
├── Clerk.sol
├── FLAT.sol
├── FlatMarket.sol Note
├── FlatMarketConfig.sol
├── TreasuryHolder.sol Note
├── interfaces
├── libraries
│   ├── LatteConversion.sol
│   ├── LatteMath.sol
│   └── WadRayMath.sol
├── mock
│   ├── MockChainlinkAggregator.sol
│   ├── MockFlatMarket.sol
│   ├── MockLatteSwapPairLPChainlinkAggregator.sol
│   ├── MockLatteSwapYield.sol
│   ├── MockOracle.sol
│   ├── MockPCSMasterChef.sol
│   ├── MockYieldStrategy.sol
│   ├── NonNativeReceivableToken.sol
│   └── SimpleToken.sol
├── oracles
│   ├── ChainlinkOracle.sol
│   ├── CompositeOracle.sol
│   ├── OffChainOracle.sol
│   └── aggregators
│       └── chainlink
│           ├── LPChainlinkAggregator.sol
│           └── TokenChainlinkAggregator.sol
└── strategies
    ├── liquidation
    │   └── LatteSwapLiquidationStrategy.sol
    └── yield
        ├── LatteSwapYieldStrategy.sol
        └── PCSYieldStrategy.sol
```