PILGRIM SECURITY ASSESSMENT REPORT

JAN. 10 ~ JAN. 28, 2022

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

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 Pilgrim smart contract from Jan. 10 until Jan. 28, 2022. 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 expert participated in a vulnerability analysis of the contract. The expert is professional hacker 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: Low 1. We have confirmed that the issue is resolved. 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	Pilgrim
Platform	Ethereum
# Commits	5793043aee0054057f44247ad3538bb94d06b820
# of Files	45 (Uniswap 51 files excluded)
# of Lines	3,096

KEY AUDIT POINTS & PROCESS

Pilgrim is securitized NFT AMM Decentralized Exchange developed by Pilgrim team. Pilgrim consists of securitized NFT AMM DEX and MetaNFT, and a staking program called PilgrimTemple. Accordingly, we focused on issues that may occur in the DEX and staking programs.
For example, we've checked whether the DEX's trading volume is calculated as expected, whether there are any malfunctions caused by sudden price fluctuations, whether it is not affected by maliciously implemented tokens, whether staking rewards are calculated as expected, etc.
However, we did not take any internal hackings by administrators into account (e.g., Rug Pull). In addition, the design of protocol is also excluded from technical review.

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. (e.g., Mint)

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

OVERVIEW

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

AMM RESERVE MIGHT BE DEPLETED WHEN BASE TOKEN IS DEFLATIONARY Low

Additional resources and comments

File Name : AMMFacet.sol
File Location : pilgrim-contract/contracts/core/facets/AMMFacet.sol
MD5 : 7a7478cf9674ff758b2b919336f99b4e

```
257     function _buy(
258         uint256 _metaNftId,
259         uint128 _baseIn,
260         uint128 _virtualBaseIn,
261         uint128 _baseFee,
262         uint128 _roundOut,
263         uint128 _roundFee
264     ) private {
265         address baseToken = LibGetters._getPairInfo(_metaNftId).baseToken;
266         uint256 beforeBalance = IERC20(baseToken).balanceOf(address(this));
267 
268         /// Transfer base tokens from the caller.
269         require(
270             IERC20(baseToken).transferFrom(msg.sender, address(this), _baseIn + _baseFee)
271         );
```

Details

If the base token is deflationary like SAFEMOON, the actual amount of token received may differ with the transfer parameter because part of the transfer volume is burned when transfer calls. This can cause the Pilgrim Pair base token reserve to dry out.

The issue can be mitigated by adding a statement as below in order to check whether the actual amount has been transferred as much as expected.

```
260         uint128 _virtualBaseIn,
261         uint128 _baseFee,
262         uint128 _roundOut,
263         uint128 _roundFee
264     ) private {
265         address baseToken = LibGetters._getPairInfo(_metaNftId).baseToken;
266         uint256 beforeBalance = IERC20(baseToken).balanceOf(address(this));
267 
268         /// Transfer base tokens from the caller.
269         require(
270             IERC20(baseToken).transferFrom(msg.sender, address(this), _baseIn + _baseFee)
271         );
272         /// Check whether balance is as expected.
273         require(
274             beforeBalance + _baseIn + _baseFee == IERC20(baseToken).balanceOf(address(this))
275         );
```

(VERIFIED) POSSIBLE REENTRANCY ATTACK ✓

Additional resources and comments

Details

We checked whether a Reentrancy attack was possible by ill-formed base token, but confirmed that there was no possibility of an attack since Pilgrim properly implemented logics with the Checks-effects-interactions pattern.

(VERIFIED) POSSIBLE FLASH LOAN ATTACK ✓

Additional resources and comments

Details

We checked whether it is possible to attack using Flash Loan from an external DeFi service, but confirmed that it did not affect much since round token price cannot be manipulated by external cash flow.

OVERVIEW

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

(VERIFIED) VERIFYING EXCHANGE PARAMETERS ✓

Additional resources and comments

Details We've verified whether the value of parameters related to Round Token and Base Token exchange was calculated as expected.

(VERIFIED) EIP-2535 IMPLEMENTATION ✓

Additional resources and comments

Details We've verified that Pilgrim properly implemented the EIP-2535.

(VERIFIED) ERC20 / ERC721 COMPATIBLE ✓

Additional resources and comments

Details We've verified that ERC-20 / ERC-721 Spec was properly implemented in Pilgrim.

(VERIFIED) ACCESS CONTROL ✓

Additional resources and comments

Details We've confirmed that all permissions are set as expected.

CONCLUSION

The source code of the Pilgrim contract developed by Test in Prod team 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 vulnerabilities are as follows: Low 1. We have confirmed that the issue is resolved. 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	Pilgrim
Platform	Ethereum
# Commits	5793043aee0054057f44247ad3538bb94d06b820
# of Files	45 (Uniswap 51 files excluded)
# of Lines	3,096

File Tree

```
contracts
├── IPilgrimTreasury.sol
├── PilgrimTreasury.sol
├── core
│   ├── InitDiamond.sol
│   ├── facets
│   │   ├── AMMFacet.sol
│   │   ├── DistributionFacet.sol
│   │   ├── ERC721ReceiverFacet.sol
│   │   ├── ListingFacet.sol
│   │   ├── ManagingFacet.sol
│   │   └── ViewFacet.sol
│   ├── interfaces
│   │   ├── IViewFacet.sol
│   │   └── external
│   │       └── INonfungiblePositionManager.sol
│   └── libraries
│       ├── LibAppStorage.sol
│       ├── LibBidderQueue.sol
│       ├── LibDistribution.sol
│       ├── LibGetters.sol
│       ├── LibRoundToken.sol
│       ├── LibTradingFee.sol
│       └── Modifiers.sol
├── mocks
│   ├── CoreExploitMock.sol
│   ├── ERC20Mock.sol
│   ├── ERC721Mock.sol
│   └── PilgrimMakerExploitMock.sol
├── shared
│   ├── Diamond.sol
│   ├── facets
│   │   ├── DiamondCutFacet.sol
│   │   ├── DiamondLouperFacet.sol
│   │   └── OwnershipFacet.sol
│   ├── interfaces
│   │   ├── IDiamondCut.sol
│   │   ├── IDiamondLoupe.sol
│   │   ├── IERC165.sol
│   │   └── IERC173.sol
│   └── libraries
│       ├── LibDiamond.sol
│       └── Math.sol
├── staking
│   ├── InitDiamond.sol
│   ├── facets
│   │   ├── PilgrimMakerFacet.sol
│   │   └── PilgrimTempleFacet.sol
│   ├── interfaces
│   │   ├── IPilgrimMakerFacet.sol
│   │   ├── IPilgrimTempleFacet.sol
│   │   └── IXPilgrimLockupFacet.sol
│   └── libraries
│       ├── LibAppStorage.sol
│       ├── LibXPilgrimLockup.sol
│       └── Modifiers.sol
└── token
    ├── PilgrimMetaNFT.sol
    ├── PilgrimToken.sol
    ├── XPilgrim.sol
    └── interfaces
        └── IPilgrimMetaNFT.sol
```