Smart Contract Audit Report

MinMax

CHAINSHIELD

ChainShield Team (https://chainshield.io)

20th February, 2022

Table of Contents

Introduction

MinMax

Built on top of Saddle, MinMax is the first next generation stablecoin AMM protocol on IoTeX. MinMax provides the liquidity foundation for all kind of stablecoins. As IoTeX 's core cross-chain liquidity network, MinMax helps facilitate the transfer of assets between IoTeX and other blockchains. Users deposit crypto assets into our IoTeX liquidity pools to earn passive yield from transaction fees, token-based incentives, and eventually automated DeFi strategies.

ChainShield

ChainSheild Inc. is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystems by offering top-notch, industry-leading services and products, including smart contract auditing, project consultation and penetration test of blockchain infrastructure. We can be reached via Website (https://chainshield.io), or Email (support@chainsheild.io).

Scope

This report covers the auditing methodology and finding of MinMax Contract in https://github.com/minmaxfinance/minmax-stableswap-contracts at commit ff752c7. Smart contracts in this repo and based the Saddle which has been audited

by OpenZeppelin, Quantstamp and Certik. So in this audit we focus on the patches in the original Saddle which may affect the MinMax. And we follow our own audit progress to figure out if there are new problems in this contract.

Disclaimer

Note that this audit does not give any warranties on finding all possible security issues of the given smart contracts. In other words, the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Last but not least, this security audit should not be used as investment advice.

Findings

Summary

Here is a summary of our findings after analyzing the design and implementation of MinMax. During the first phase of our audit, we study the smart contract source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify issues reported by our tools. We further manually review business logics such as Swaping,Vesting, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Finding	**Severity**
2 - Missing input validation	Low
3 - Block Timestamp Manipulation	Low
4 - Outdated Solidity version in use	Low

Overall, MinMax smart contracts are well-designed and well-engineered. We have so far identified a few non-critical issues: some of them involve subtle corner cases that might not be previously thought of, while others refer to unusual interactions among multiple contracts. For each uncovered issue, we have therefore developed test cases for reasoning, reproduction, and/or verification. After further analysis and internal discussion, we determined a few issues of varying severities that need to be brought up and paid more attention to, which are listed in the above table. More information can be found in the next subsection.

Key Findings

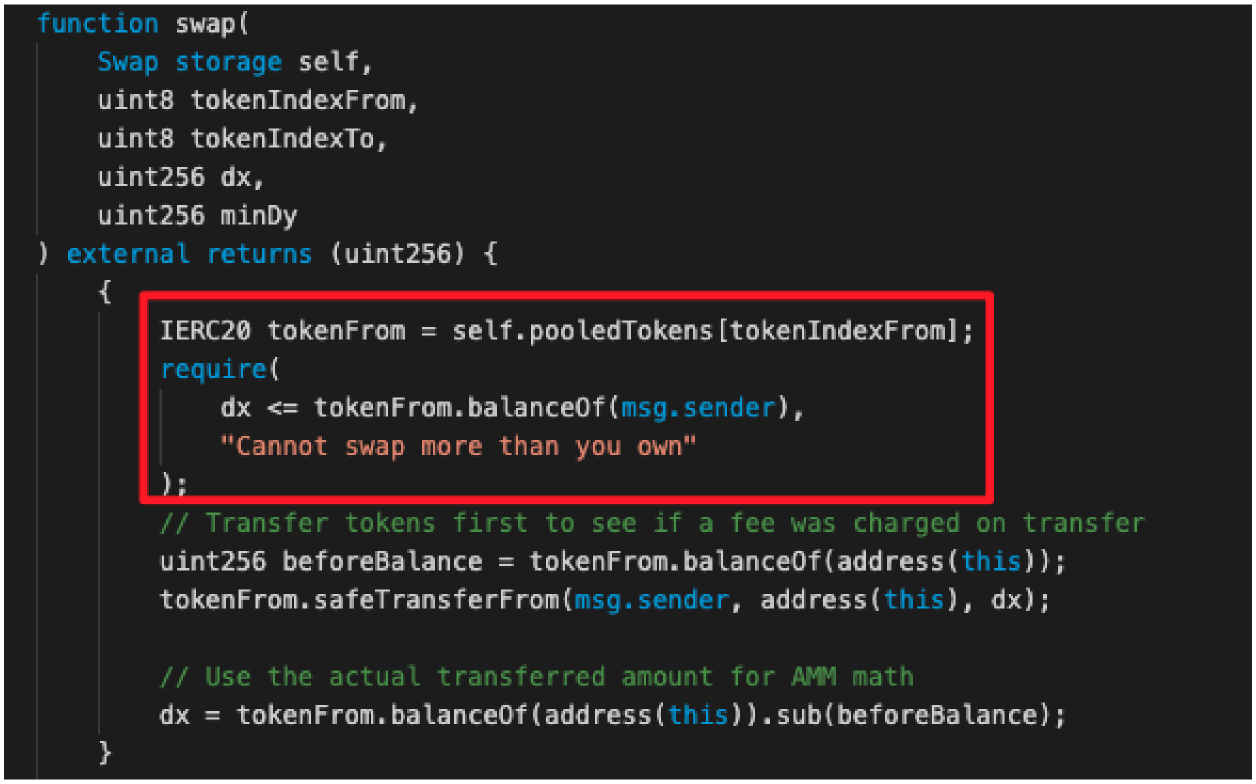
In this section, we list the key findings we have during the audit.

Finding 1: Missing input validation

• Severity: Low

The following functions are missing checks for user input:

1. The Swap.swap/SwapUtils.swap function does not check if value of the 2 tokenIndexFrom and tokenIndexTo parameters are lower than the length of the array.
2. The SwapV1.swap/SwapUtilsV1.swap function does not check if value of the 2 tokenIndexFrom and tokenIndexTo parameters are lower than the length of the array.



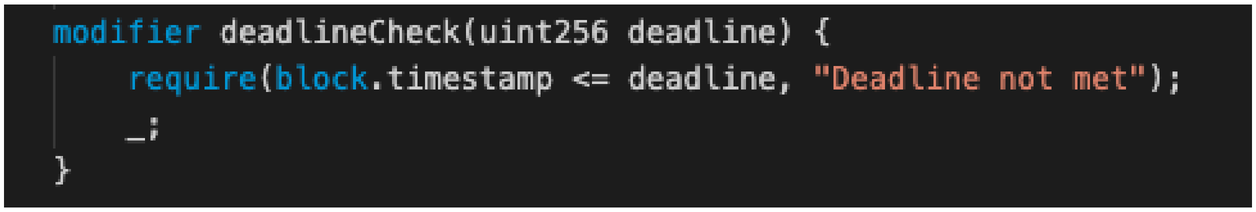
```
function swap(
    Swap storage self,
    uint8 tokenIndexFrom,
    uint8 tokenIndexTo,
    uint256 dx,
    uint256 minDy
) external returns (uint256) {
    {
        IERC20 tokenFrom = self.pooledTokens[tokenIndexFrom];
        require(
            dx <= tokenFrom.balanceOf(msg.sender),
            "Cannot swap more than you own"
        );
        // Transfer tokens first to see if a fee was charged on transfer
        uint256 beforeBalance = tokenFrom.balanceOf(address(this));
        tokenFrom.safeTransferFrom(msg.sender, address(this), dx);

        // Use the actual transferred amount for AMM math
        dx = tokenFrom.balanceOf(address(this)).sub(beforeBalance);
    }
```

To fix this issue, MinMax can add require statements in the functions enumerated above, which should check that the input arguments are within bounds.

Finding 2: Block Timestamp Manipulation

• Severity: Low

There are many functions in the codebase that depend on the block timestamps. Such as deadlineCheck and rampA, it's important to realize that miners can set the timestamp of a block, and attackers may be able to manipulate timestamps for their own purposes.



```
modifier deadlineCheck(uint256 deadline) {
    require(block.timestamp <= deadline, "Deadline not met");
    _;
}
```

Finding 3: Outdated Solidity version in use

• Severity: Low

The codebase is currently using solidity version 0.6.12, there are some serious bugs exists in version 0.6.x, such as Solidity Dynamic Array Cleanup Bug.

To fix this issue, we advise using the latest version of the compiler at the time of deployment.

Conclusion

In this audit, we have analyzed the design and implementation of MinMax smart contract. We have so far identified 3 non-critical issues, most of them involve subtle corner cases that might not be previously thought of. Overall, MinMax smart contracts are well-designed and well-engineered.

As a final precaution, we need to emphasize that smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedback or suggestions on our methodology and findings.