



BlockSec

Security Audit Report for Swap Contracts

Date: Feb 6, 2024

Version: 1.0

Contact: contact@blocksec.com

Contents

1	Introduction	1
1.1	About Target Contracts	1
1.2	Disclaimer	1
1.3	Procedure of Auditing	2
1.3.1	Software Security	2
1.3.2	DeFi Security	2
1.3.3	NFT Security	2
1.3.4	Additional Recommendation	3
1.4	Security Model	3
2	Findings	4
2.1	Note	4
2.1.1	Pairs for weird ERC20 tokens may not behave as expected	4
2.1.2	Potential front-running issue	4

Report Manifest

Item	Description
Client	PotatoSwap
Target	Swap Contracts

Version History

Version	Date	Description
1.0	Feb 6, 2024	First Release

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 5 million dollars by blocking multiple attacks. They can be reached at [Email](#), [Twitter](#) and [Medium](#).

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The scope of this audit is centered on the Swap Contracts of PotatoSwap, including the Core ¹ and Periphery ² contracts. Please note that the testing-purpose smart contracts such as those under the contracts/test folders are not included in the audit. Furthermore, the repositories in question are derivatives of the well-established Uniswap protocol (i.e., v2-core and v2-periphery), which have been considered reliable over an extended period. Our audit will exclusively examine the deviations from the forked code in comparison to its original Uniswap counterparts.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version ([Version 1](#)), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
Core	Version 1	1cabac0244da177b0128f895ba490f13bb5cdadf
Periphery	Version 1	7d996c33116349cfddce44b71a60ff46d164e27f

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

¹<https://github.com/PotatoSwap-Finance/potato-swap-core>

²<https://github.com/PotatoSwap-Finance/potato-swap-periphery>

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
 - **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
 - **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.
- We show the main concrete checkpoints in the following.

1.3.1 Software Security

- * Reentrancy
- * DoS
- * Access control
- * Data handling and data flow
- * Exception handling
- * Untrusted external call and control flow
- * Initialization consistency
- * Events operation
- * Error-prone randomness
- * Improper use of the proxy system

1.3.2 DeFi Security

- * Semantic consistency
- * Functionality consistency
- * Permission management
- * Business logic
- * Token operation
- * Emergency mechanism
- * Oracle security
- * Whitelist and blacklist
- * Economic impact
- * Batch transfer

1.3.3 NFT Security

- * Duplicated item
- * Verification of the token receiver
- * Off-chain metadata security

1.3.4 Additional Recommendation

- * Gas optimization
- * Code quality and style

 **Note** The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology³ and Common Weakness Enumeration⁴. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact	Likelihood	
	High	Medium
High	High	Medium
Low	Medium	Low
	High	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

³https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

⁴<https://cwe.mitre.org/>

Chapter 2 Findings

In summary, we did not find any potential security issues. However, there are **two** notes that need to be considered.

- Note: 2

ID	Severity	Description	Category	Status
1	-	Pairs for weird ERC20 tokens may not behave as expected	Note	-
2	-	Potential front-running issue	Note	-

The details are provided in the following sections.

2.1 Note

2.1.1 Pairs for weird ERC20 tokens may not behave as expected

Description The PotatoSwap adopts a Uniswap-like structure, meaning that if a pair incorporates *weird* ERC20 tokens ¹ as the underlying assets, the behavior may not align with expectations. For example, some weird ERC20 tokens have the potential to arbitrarily burn the pair's token holdings, while others, like deflationary tokens, might adjust the pair's token balances without any notification. Utilizing such tokens could lead to unforeseen financial losses for liquidity providers.

2.1.2 Potential front-running issue

Description When users (externally owned accounts, or EOAs) directly interact with the `PotatoPair` contract, the process can be divided into two separate transactions. In the first transaction, users transfer their tokens to the `PotatoPair`. Subsequently, in a different transaction, they invoke the `swap` function to exchange those tokens. However, the `swap` function calculates the user's `amountIn` by determining the difference between the current balance and the last recorded reserves. Additionally, the `PotatoPair` contract's `skim` function allows users to retrieve the tokens they transferred. A malicious actor could exploit these methods to appropriate users' tokens, transferred in the first transaction, through a front-running attack.

```
159   function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock
160   {
161     require(amount0Out > 0 || amount1Out > 0, 'Potato: INSUFFICIENT_OUTPUT_AMOUNT');
162     (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
163     require(amount0Out < _reserve0 && amount1Out < _reserve1, 'Potato: INSUFFICIENT_LIQUIDITY')
164     ;
165     uint balance0;
166     uint balance1;
167     { // scope for _token{0,1}, avoids stack too deep errors
168       address _token0 = token0;
```

¹<https://github.com/d-xo/weird-erc20>

```

168     address _token1 = token1;
169     require(to != _token0 && to != _token1, 'Potato: INVALID_TO');
170     if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); // optimistically transfer
171         tokens
172     if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); // optimistically transfer
173         tokens
174     if (data.length > 0) IPotatoCallee(to).potatoCall(msg.sender, amount0Out, amount1Out, data)
175         ;
176     balance0 = IERC20(_token0).balanceOf(address(this));
177     balance1 = IERC20(_token1).balanceOf(address(this));
178 }
179
180     uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out) :
181         0;
182     uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out) :
183         0;
184     require(amount0In > 0 || amount1In > 0, 'Potato: INSUFFICIENT_INPUT_AMOUNT');
185     { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
186     uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(2));
187     uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(2));
188     require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul
189             (1000**2), 'Potato: K');
190 }
191
192     _update(balance0, balance1, _reserve0, _reserve1);
193     emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
194 }
```

Listing 2.1: PotatoPair.sol

```

190 function skim(address to) external lock {
191     address _token0 = token0; // gas savings
192     address _token1 = token1; // gas savings
193     _safeTransfer(_token0, to, IERC20(_token0).balanceOf(address(this)).sub(reserve0));
194     _safeTransfer(_token1, to, IERC20(_token1).balanceOf(address(this)).sub(reserve1));
195 }
```

Listing 2.2: PotatoPair.sol