CyberCrime Shield

cybercrimeshield.org

Smart Contract Audit Report

Polar Finance

https://www.polardefi.com

AUDIT TYPE: **PUBLIC**

https://cybercrimeshield.org/secure/polar
ID:2390320

April 10, 2021

TABLE OF CONTENTS

SMART CONTRACTS

https://bscscan.com/address/0xae9af0f99346dee3f9545151dd57f341ed6c14b8

https://bscscan.com/address/0xc55b4f7f3bf253f9abb93795e4425e70c9cc09b8

https://bscscan.com/address/0xB975514CBE9640917a5Dc747Ab9eA75F4d6d915C

Mirror: https://cybercrimeshield.org/secure/uploads/polar.zip

CRC32: 287A87EE

MD5: 52A825A2E27457ED3F98830C0D9EEAB7

SHA-1: 8F596EBD381BC1079B97064853AD09360133D893

INTRODUCTION

Blockchain platforms, such as Nakamoto's Bitcoin, enable the trade of crypto-currencies between mutually mistrusting parties.

To eliminate the need for trust, Nakomoto designed a peer-to-peer network that enables its peers to agree on the trading transactions.

Vitalik Buterin identified the applicability of decentralized computation beyond trading, and designed the Ethereum blockchain which supports the execution of programs, called smart contracts, written in Turing-complete languages.

Smart contracts have shown to be applicable in many domains including financial industry, public sector and cross-industry.

The increased adoption of smart contracts demands strong security guarantees. Unfortunately, it is challenging to create smart contracts that are free of security bugs.

As a consequence, critical vulnerabilities in smart contracts are discovered and exploited every few months.

In turn, these exploits have led to losses reaching billions worth of USD in the past few years.

It is apparent that effective security checks for smart contracts are strictly needed.

Our company provides comprehensive, independent smart contract auditing.

We help stakeholders confirm the quality and security of their smart contracts using our standardized audit process.

The scope of this audit was to analyze and document the Polar Finance contracts.

This document is not financial advice, you perform all financial actions on your own responsibility.

AUDIT METHODOLOGY

1. Design Patterns

We inspect the structure of the smart contract, including both manual and automated analysis.

2. Static Analysis

The static analysis is performed using a series of automated tools, purposefully designed to test the security of the contract.
All the issues found by tools were manually checked (rejected or confirmed).

3. Manual Analysis

Contract reviewing to identify common vulnerabilities. Comparing of requirements and implementation. Reviewing of a smart contract for compliance with specified customer requirements. Checking for energy optimization and self-documentation. Running tests of the properties of the smart contract in test net.

ISSUES DISCOVERED

Issues are listed from most critical to least critical. Severity is determined by an assessment of the risk of exploitation or otherwise unsafe behavior.

Severity Levels

Critical - Funds may be allocated incorrectly, lost or otherwise result in a significant loss.
Medium - Affects the ability of the contract to operate.
Low - Minimal impact on operational ability.
Informational - No impact on the contract.

AUDIT SUMMARY

The summary result of the audit performed is presented in the table below

Findings list:

LEVEL	AMOUNT
Critical	0
Medium	0
Low	0
Informational	2

FINDINGS (2)

1. **PolarToken.sol**

```
970 abi.encode(
971 DOMAIN_TYPEHASH,
972 keccak256(bytes(name())),
973 getChainId(),
974 address(this)
```

Potentially unbounded data structure passed to builtin.

Gas consumption in function "delegateBySig" in contract "PolarToken" depends on the size of data structures that may grow unboundedly. Specifically the "1-st" argument to builtin "keccak256" may be able to grow unboundedly causing the builtin to consume more gas than the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

2. **PolarToken.sol**

Loop over unbounded data structure.

Gas consumption in function "getPriorVotes" in contract "PolarToken" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

```
1045 uint32 lower = 0;
1046 uint32 upper = nCheckpoints - 1;
1047 while (upper > lower) {
1048 uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overflow
1049 Checkpoint memory cp = checkpoints[account][center];
```

CONCLUSION

- Contracts have high code readability
- Gas usage is optimal
- Contracts are fully BSC completable
- No backdoors or overflows are present in the contracts