auditace.tech

AuditAce

FROM INCEPTION TO SUCCESS

Smart Contract Audit

FOR

Swobbels

DATED : 2 March, 2024

AUDIT SUMMARY

Project name – Swobbels

Date: 2 March, 2024

Scope of Audit- Audit Ace was consulted to conduct the smart contract audit of the solidity source codes.

Audit Status: **Passed**

Issues Found

Status	Critical	High	Medium	Low	Suggestion
Open	0	0	0	0	1
Acknowledged	0	0	0	0	0
Resolved	0	0	0	0	0

USED TOOLS

Tools:

1- Manual Review:
A line by line code review has been performed by audit ace team.

2- BSC Test Network: All tests were conducted on the BSC Test network, and each test has a corresponding transaction attached to it. These tests can be found in the "Functional Tests" section of the report.

3- Slither :
The code has undergone static analysis using Slither.

Testnet version:
The tests were performed using the contract deployed on the BSC Testnet, which can be found at the following address:
https://testnet.bscscan.com/address/0x7cdd07482b59f18548e21913e29b373e2d403f4e#code

Token Information

Token Name : Swobbels

Token Symbol: SWOB

Decimals: 6

Total Supply: 20000000

Network: Etherscan

Token Type: ERC-20

Token Address:
0x712e3354d0b5340B9Ab6Ed1480d45a1C86CE306b

Checksum:
Ae1c3a4fbb6e83e8393a57617b5a112

Owner:
0x0000000000000000000000000000000000000000
(at time of writing the audit)

Deployer:
0xACcb23a1c9AE5e2d48aA0022a4675c92d43d9f82

TOKEN OVERVIEW

Fees:
Total Buy Tax: 0%
Total **Sell Tax: 0%**
Transfer Tax: 0%

Fees Privilege: Owner

Ownership: Renounced

Minting: None

Max Tx Amount/ Max Wallet Amount: No

Blacklist: No

AUDIT METHODOLOGY

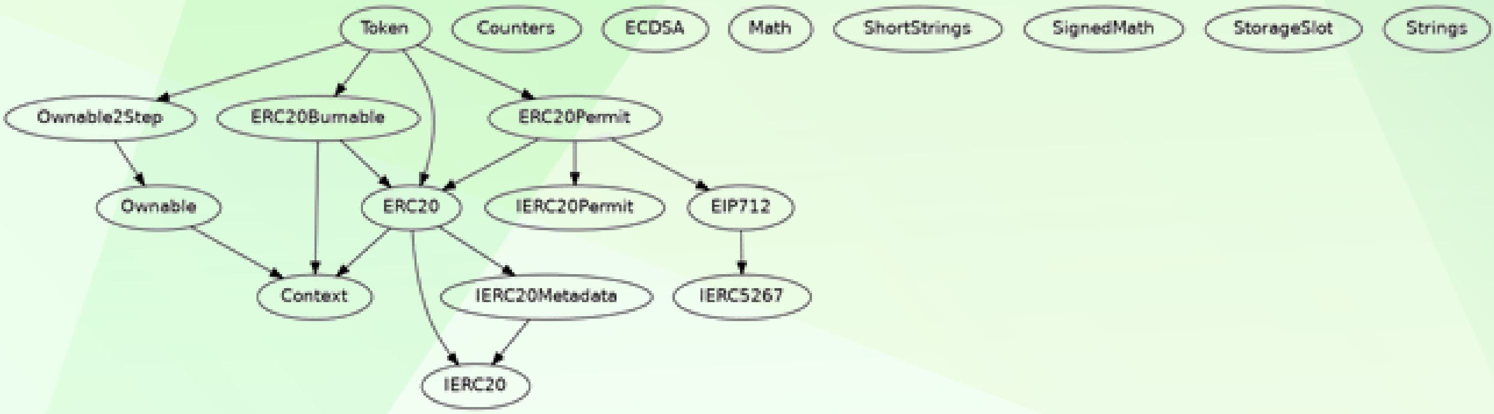
The auditing process will follow a routine as special considerations by Auditace:

- Review of the specifications, sources, and instructions provided to Auditace to make sure the contract logic meets the intentions of the client without exposing the user's funds to risk.
- Manual review of the entire codebase by our experts, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
- Specification comparison is the process of checking whether the code does what the specifications, sources, and instructions provided to Auditace describe.
- Test coverage analysis determines whether the test cases are covering the code and how much code isexercised when we run the test cases.
- Symbolic execution is analysing a program to determine what inputs cause each part of a program to execute.
- Reviewing the codebase to improve maintainability, security, and control based on the established industry and academic practices.

VULNERABILITY CHECKLIST

- Return values of low-level calls
- Private modifier
- Multiple Sends
- Using Suicide
- Gas Limitand Loops
- Address hardcoded
- Exception Disorder
- Using inline assembly
- Divide before multiply
- Missing Zero Address Validation
- Compiler version not fixed
- **Gasless Send**
- Using block.timestamp
- Re-entrancy
- Tautology or contradiction
- Timestamp Dependence
- Revert/require functions
- Use of tx.origin
- Integer overflow/underflow
- Dangerous strict equalities
- Using SHA3
- Using throw

INHERITANCE TREE



STATIC ANALYSIS

A static analysis of the code was performed using Slither.

No issues were found.

```
INFO:Detectors:
Math.mulDiv(uint256,uint256,uint256) (Math.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (Math.sol#101)
        - inverse = (3 * denominator) ^ 2 (Math.sol#116)
Math.mulDiv(uint256,uint256,uint256) (Math.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (Math.sol#101)
        - inverse *= 2 - denominator * inverse (Math.sol#120)
Math.mulDiv(uint256,uint256,uint256) (Math.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (Math.sol#101)
        - inverse *= 2 - denominator * inverse (Math.sol#121)
Math.mulDiv(uint256,uint256,uint256) (Math.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (Math.sol#101)
        - inverse *= 2 - denominator * inverse (Math.sol#122)
Math.mulDiv(uint256,uint256,uint256) (Math.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (Math.sol#101)
        - inverse *= 2 - denominator * inverse (Math.sol#123)
Math.mulDiv(uint256,uint256,uint256) (Math.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (Math.sol#101)
        - inverse *= 2 - denominator * inverse (Math.sol#124)
Math.mulDiv(uint256,uint256,uint256) (Math.sol#55-134) performs a multiplication on the result of a division:
        - denominator = denominator / twos (Math.sol#101)
        - inverse *= 2 - denominator * inverse (Math.sol#125)
Math.mulDiv(uint256,uint256,uint256) (Math.sol#55-134) performs a multiplication on the result of a division:
        - prod0 = prod0 / twos (Math.sol#104)
        - result = prod0 * inverse (Math.sol#131)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
Contract locking ether found:
        Contract Token (Token.sol#16-48) has payable functions:
         - Token.receive() (Token.sol#28)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
INFO:Detectors:
ERC20Permit.constructor(string).name (ERC20Permit.sol#44) shadows:
        - ERC20.name() (ERC20.sol#62-64) (function)
        - IERC20Metadata.name() (IERC20Metadata.sol#17) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Ownable2Step.transferOwnership(address).newOwner (Ownable2Step.sol#35) lacks a zero-check on :
                - _pendingOwner = newOwner (Ownable2Step.sol#36)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
ERC20Permit.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (ERC20Permit.sol#49-68) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,ERC20Permit: expired deadline) (ERC20Permit.sol#58)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

STATIC ANALYSIS

```
INFO:Detectors:
Pragma version^0.8.0 (Context.sol#4) allows old versions
Pragma version^0.8.0 (Counters.sol#4) allows old versions
Pragma version^0.8.0 (ECDSA.sol#4) allows old versions
Pragma version^0.8.8 (EIP712.sol#4) is known to contain severe issues (https://solidity.readthedocs.io/en/latest/bugs.html)
Pragma version^0.8.0 (ERC20.sol#4) allows old versions
Pragma version^0.8.0 (ERC20Burnable.sol#4) allows old versions
Pragma version^0.8.0 (ERC20Permit.sol#4) allows old versions
Pragma version^0.8.0 (IERC20.sol#4) allows old versions
Pragma version^0.8.0 (IERC20Metadata.sol#4) allows old versions
Pragma version^0.8.0 (IERC20Permit.sol#4) allows old versions
Pragma version^0.8.0 (IERC5267.sol#4) allows old versions
Pragma version^0.8.0 (Math.sol#4) allows old versions
Pragma version^0.8.0 (Ownable.sol#4) allows old versions
Pragma version^0.8.0 (Ownable2Step.sol#4) allows old versions
Pragma version^0.8.8 (ShortStrings.sol#4) is known to contain severe issues (https://solidity.readthedocs.io/en/latest/bugs.html)
Pragma version^0.8.0 (SignedMath.sol#4) allows old versions
Pragma version^0.8.0 (StorageSlot.sol#5) allows old versions
Pragma version^0.8.0 (Strings.sol#4) allows old versions
Pragma version0.8.19 (Token.sol#9) necessitates a version too recent to be trusted. Consider deploying with 0.8.18.
solc-0.8.19 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Function ERC20Permit.DOMAIN_SEPARATOR() (ERC20Permit.sol#81-83) is not in mixedCase
Variable ERC20Permit._PERMIT_TYPEHASH_DEPRECATED_SLOT (ERC20Permit.sol#37) is not in mixedCase
Function IERC20Permit.DOMAIN_SEPARATOR() (IERC20Permit.sol#59) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
ShortStrings.slitherConstructorConstantVariables() (ShortStrings.sol#40-123) uses literals with too many digits:
        - _FALLBACK_SENTINEL = 0x00000000000000000000000000000000000000000000000000000000000000FF (ShortStrings.sol#42)
Token.constructor() (Token.sol#18-26) uses literals with too many digits:
        - _mint(supplyRecipient,200000000 * (10 ** decimals()) / 10) (Token.sol#24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Slither:Token.sol analyzed (19 contracts with 93 detectors), 95 result(s) found
```

FUNCTIONAL TESTING

1- Approve (passed):

https://testnet.bscscan.com/tx/0x6a9d513aaacffde6575a7feb9074a6d119e1fd6c045b4de0887db55e3b025c3d

CLASSIFICATION OF RISK

Severity	Description
Critical	These vulnerabilities could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
High-Risk	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.
Medium-Risk	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.
Low-Risk	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.
Gas Optimization /Suggestion	A vulnerability that has an informational character but is not affecting any of the code.

Findings

Severity	Found
Critical	0
High-Risk	0
Medium-Risk	0
Low-Risk	0
Gas Optimization / Suggestions	1

MANUAL TESTING

Optimization

Severity: Optimization

Subject: Remove unused code

Status: Open

Overview:

Unused variables are allowed in Solidity, and they do. not pose a direct security issue. It is the best practice though to avoid them.

```
function _msgData() internal view virtual returns (bytes calldata) {
        return msg.data;
    }
function decrement(Counter storage counter) internal {
        uint256 value = counter._value;
        require(value > 0, "Counter: decrement overflow");
        unchecked {
            counter._value = value - 1;
        }
    }
function recover(bytes32 hash, bytes memory signature) internal pure returns (ad-
dress) {
        (address recovered, RecoverError error) = tryRecover(hash, signature);
        _throwError(error);
        return recovered;
    }
function recover(bytes32 hash, bytes32 r, bytes32 vs) internal pure returns (ad-
dress) {
        (address recovered, RecoverError error) = tryRecover(hash, r, vs);
        _throwError(error);
        return recovered;
    }
function toEthSignedMessageHash(bytes32 hash) internal pure returns (bytes32 mes-
sage) {
        // 32 is the length in bytes of hash,
        // enforced by the type signature above
        /// @solidity memory-safe-assembly
        assembly {
            mstore(0x00, "\x19Ethereum Signed Message:\n32")
            mstore(0x1c, hash)
            message := keccak256(0x00, 0x3c)
        }
    }
function toEthSignedMessageHash(bytes memory s) internal pure returns (bytes32) {
        return keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n",
Strings.toString(s.length), s));
    }

function toDataWithIntendedValidatorHash(address validator, bytes memory data) in-
ternal pure returns (bytes32) {
        return keccak256(abi.encodePacked("\x19\x00", validator, data));
    }
```

DISCLAIMER

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document. Always Do your own research and protect yourselves from being scammed. The Auditace team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools. Under no circumstances did Auditace receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Always Do your own research and protect yourselves from scams. This document should not be presented as a reason to buy or not buy any particular token. The Auditace team disclaims any liability for the resulting losses.

ABOUT AUDITACE

We specializes in providing thorough and reliable audits for Web3 projects. With a team of experienced professionals, we use cutting-edge technology and rigorous methodologies to evaluate the security and integrity of blockchain systems. We are committed to helping our clients ensure the safety and transparency of their digital assets and transactions.

https://auditace.tech/

https://t.me/Audit_Ace

https://twitter.com/auditace_

https://github.com/Audit-Ace