auditace.tech

AuditAce
FROM INCEPTION TO SUCCESS

Smart Contract Audit

FOR

EZP

DATED : 11 Feb 2025

AUDIT SUMMARY

Project name – EZP

Date: 11 Feb 2025

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

Audit Status: **High risk major flag**

Issues Found

Status	Critical	High	Medium	Low	Suggestion
Open	0	2	1	0	2
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://bscscan.com/address/0x50387a24253150e808875f3cac8111546623ba64#code

Token Information

Token Address:
0x50387A24253150E808875F3CaC8111546623ba64

Name: EZP

Symbol: EziPay Coin

Decimals: 18

Network: Ether Scan

Token Type: ERC-20

Owner: 0x097Da4E456f9d7c5C5446d595cF9F8E5E4D2564C

Deployer:
0x097Da4E456f9d7c5C5446d595cF9F8E5E4D2564C

Token Supply: 1,000,000,000

Checksum: e313bfbe2f031a75cd6420812f40348c

Testnet:
https://bscscan.com/address/0x50387a24253150e808875f3cac8111546623ba64#code

TOKEN OVERVIEW

Buy Fee: 0-0%

Sell Fee: 0-0%

Transfer Fee: 0-0%

Fee Privilege: Owner

Ownership: Owned

Minting: Yes

Max Tx: No

Blacklist: No

AUDIT METHODOLOGY

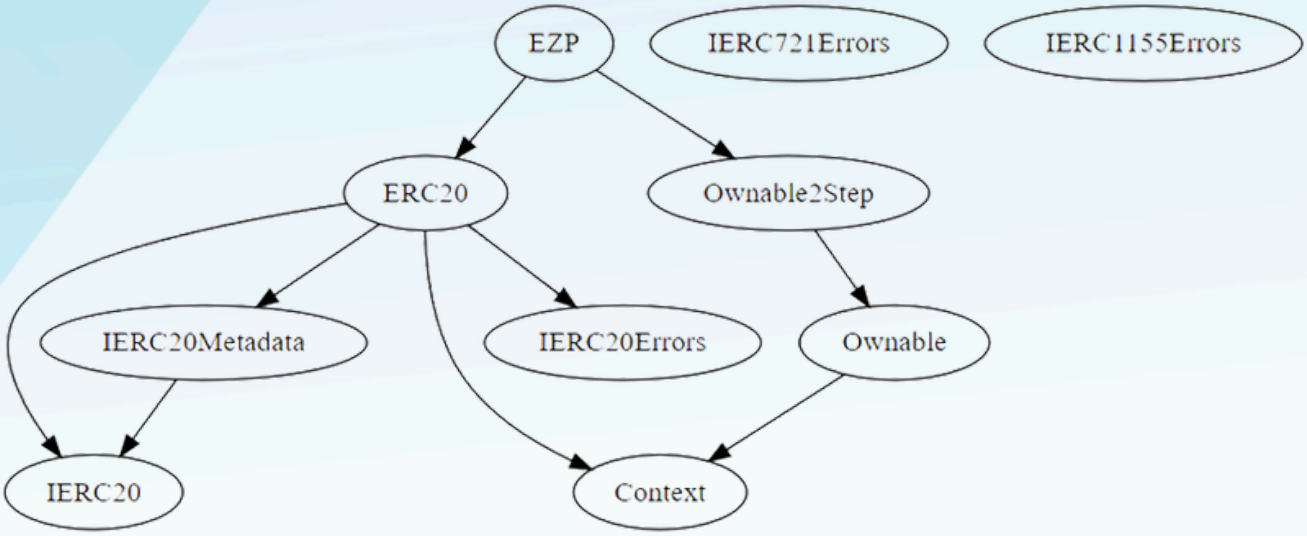
The auditing process will follow a routine as special considerations by Auditace:

- 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

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

INHERITANCE TREE



POINTS TO NOTE

- The owner can mint and burn tokens.

STATIC ANALYSIS

```
Contract locking ether found:
        Contract EZP (EZP.sol#812-890) has payable functions:
         - EZP.constructor() (EZP.sol#844)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
```

```
Ownable2Step.transferOwnership(address).newOwner (EZP.sol#769) lacks a zero-check on :
                - _pendingOwner = newOwner (EZP.sol#770)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
2 different versions of Solidity are used:
        - Version constraint ^0.8.26 is used by:
                -^0.8.26 (EZP.sol#10)
                -^0.8.26 (EZP.sol#798)
        - Version constraint ^0.8.20 is used by:
                -^0.8.20 (EZP.sol#92)
                -^0.8.20 (EZP.sol#120)
                -^0.8.20 (EZP.sol#150)
                -^0.8.20 (EZP.sol#315)
                -^0.8.20 (EZP.sol#629)
                -^0.8.20 (EZP.sol#731)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Context._contextSuffixLength() (EZP.sol#141-143) is never used and should be removed
Context._msgData() (EZP.sol#137-139) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

STATIC ANALYSIS

```
Version constraint ^0.8.20 contains known severe issues (https://solidity.readthedocs.io/en/latest/bugs.html)
        - VerbatimInvalidDeduplication
        - FullInlinerNonExpressionSplitArgumentEvaluationOrder
        - MissingSideEffectsOnSelectorAccess.
It is used by:
        - ^0.8.20 (EZP.sol#92)
        - ^0.8.20 (EZP.sol#120)
        - ^0.8.20 (EZP.sol#150)
        - ^0.8.20 (EZP.sol#315)
        - ^0.8.20 (EZP.sol#629)
        - ^0.8.20 (EZP.sol#731)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Slither:EZP.sol analyzed (10 contracts with 93 detectors), 6 result(s) found
```

Result => A static analysis of contract's source code has been performed using slither,

No major issues were found in the output

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	2
Medium-Risk	1
Low-Risk	0
Gas Optimization / Suggestions	2

MANUAL TESTING

Centralization – Owner can mint token.

Severity: High

Function: mint

Status: Open

Overview:
The owner is able to mint unlimited tokens which is not recommended as this functionality can cause the token to lose it's value and the owner can also use it to manipulate the price of the token.

```
function mint(address receiverAddress, uint256 amount) public onlyOwner {
require(amount > 0, "INVALID_AMOUNT");
require(receiverAddress != address(0), "INVALID_ADDRESS");
    _mint(receiverAddress, amount);
emit MintToken(msg.sender, receiverAddress, amount);
  }
```

Suggestion: There should be a locking period so that the wallet cannot be locked for an indefinite. Period of time.

MANUAL TESTING

Centralization – Owner can burn token.

Severity: High

Function: burn

Status: Open

Overview:
The owner is able to burn unlimited tokens which is not recommended as this functionality can cause the token to lose it's value and the owner can also use it to manipulate the price of the token.

```
function burnToken(address burnAddress, uint256 amount) public onlyOwner {
require(amount != 0, "INVALID_AMOUNT");
require(burnAddress != address(0), "INVALID_ADDRESS");
    _burn(burnAddress, amount);
emit BurnToken(msg.sender, burnAddress, amount);
  }
```

Suggestion: There should be a locking period so that the wallet cannot be locked for an indefinite. Period of time.

MANUAL TESTING

Centralization – The owner can regain ownership.

Severity: Medium

Function: pendingOwner

Status: Open

Overview:

The owner can regain ownership after transferring it with the following steps:

1. Call lock function to set previous owner to the own address
2. Call unlock function to get ownership back
3. Transfer/renounce ownership
4. Call unlock function to get ownership back

```
abstract contract Ownable2Step is Ownable {
  address private _pendingOwner;

event OwnershipTransferStarted(address indexed previousOwner, address indexed newOwner);

/**
   * @dev Returns the address of the pending owner.
   */
function pendingOwner() public view virtual returns (address) {
return _pendingOwner;
  }

/**
   * @dev Starts the ownership transfer of the contract to a new account. Replaces the pending transfer if there is one.
   * Can only be called by the current owner.
   *
```

MANUAL TESTING

```
  * Setting `newOwner` to the zero address is allowed; this can be used to
cancel an initiated ownership transfer.
  */
function transferOwnership(address newOwner) public virtual override
onlyOwner {
   _pendingOwner = newOwner;
emit OwnershipTransferStarted(owner(), newOwner);
 }

/**
  * @dev Transfers ownership of the contract to a new account (`newOwner`)
and deletes any pending owner.
  * Internal function without access restriction.
  */
function _transferOwnership(address newOwner) internal virtual override {
delete _pendingOwner;
super._transferOwnership(newOwner);
 }

/**
  * @dev The new owner accepts the ownership transfer.
  */
function acceptOwnership() public virtual {
   address sender = _msgSender();
if (pendingOwner() != sender) {
revert OwnableUnauthorizedAccount(sender);
   }
   _transferOwnership(sender);
 }
}
```

Suggestion: Make sure to set the previous ownership back to address zero after using the unlock function.

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) {
returnmsg.data;
  }

function _contextSuffixLength() internal view virtual returns (uint256) {
return0;
  }
}
```

MANUAL TESTING

Optimization

Severity: Informational

Subject: Floating Pragma.

Status: Open

Overview:
It is considered best practice to pick one compiler version and stick with it. With a floating pragma, contracts may accidentally be deployed using an outdated.

pragma solidity ^0.8.20;

Suggestion:
Adding the latest constant version of solidity is recommended, as this prevents the unintentional deployment of a contract with an outdated compiler that contains unresolved bugs.

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