



DApp Developers and Smart Contract Auditors

**SMART CONTRACT SECURITY AUDIT**  
of  
**ZAPIT CONTRACTS**



Smart Contract Audit of Zapit  
June 10th, 2024 | v. 1.0



## TABLE OF CONTENTS

---

AUDIT INTRODUCTION	3
--------------------	---

---

AUDIT DOCUMENT	4
----------------	---

---

AUDIT SCOPE	4
• Initial Review Scope	
• Final Review Scope	

---

AUDIT SUMMARY	8
---------------	---

---

AUDIT METHODOLOGY	9
-------------------	---

---

SYSTEM OVERVIEW	13
-----------------	----

---

FINDINGS	14
----------	----

---

STATIC ANALYSIS REPORT	15
------------------------	----

---

MANUAL REVIEW	18
---------------	----

---

UNIT TEST REPORT	21
------------------	----

---

DISCLAIMER	24
------------	----

---

ABOUT SECUREDAPP	<b>27</b>
------------------	-----------



## AUDIT INTRODUCTION

<b>Auditing Firm</b>	SecureDApp Auditors
<b>Audit Architecture</b>	SecureDApp Auditing Standard
<b>Language</b>	Solidity
<b>Client Firm</b>	Zapit
<b>Website</b>	<a href="#">Zapit</a>
<b>Twitter</b>	<a href="https://x.com/zapit_io">https://x.com/zapit_io</a>
<b>Linkedin</b>	<a href="https://www.linkedin.com/company/zapit-io">https://www.linkedin.com/company/zapit-io</a>
<b>Report Date</b>	July 16th, 2024

## About Zapit

Zapit is a comprehensive, self-custodial cryptocurrency platform designed to revolutionize peer-to-peer payments and decentralized services. As a universal payment app, Zapit provides seamless access to Web 3.0, enabling users to engage with decentralized applications, execute transactions, store tokens, and trade—all within a single, integrated platform.



## AUDIT DOCUMENT

<b>Name</b>	Smart Contract Code Review and Security Analysis Report for Zapit
<b>Approved By</b>	Himanshu Gautam   CTO at SecureDApp
<b>Type</b>	Decentralized P2P service with non-custodial escrow system
<b>Platform</b>	EVM
<b>Language</b>	Solidity
<b>Changelog</b>	16.07.2024 – Final Review

## AUDIT SCOPE

The scope of this report is to audit the smart contract source code of Zapit P2P contracts. Our client provided us with four facets of smart contracts of diamond proxy pattern.

- AdminFacet.sol
- EscrowFacet.sol
- EscrowFacetERC20.sol
- SignatureFacet.sol

All the contracts were written in Solidity and based on the Diamond Proxy Standard (EIP-2535). Smart contracts are to be deployed on multiple EVM compatible networks. AdminFacet implements configurations with respect to fee, currencies allowed, arbitrator and pausable features. EscrowFacet implements functionalities for P2P service of native chain tokens while EscrowFacetERC20 implements P2P functions for ERC20 tokens. SignatureFacet implements functions to verify signed messages based on EIP 712 standard.

After initial research, we agreed to perform the following tests and analyses as part of our well-rounded audit:

- Smart contract behavioral consistency analysis
- Test coverage analysis
- Penetration testing: checking against our database of vulnerabilities and simulating manual attacks against the contracts
- Static analysis
- Manual code review and evaluation of code quality
- Analysis of GAS usage
- Contract analysis with regards to the host network



## Initial Review Scope

<b>Repository</b>	<a href="https://github.com/zapit-io/p2p-evmContract/tree/audit">https://github.com/zapit-io/p2p-evmContract/tree/audit</a>
<b>Commit Hash</b>	5e04062aff23fcb561993ca8914478401710616b
<b>Functional Requirements</b>	Partial documentation provided. README.md
<b>Technical Requirements</b>	Partial documentation provided. README.md
<b>Contracts Addresses</b>	-
<b>Contracts</b>	EscrowFacetERC20.sol EscrowFacet.sol AdminFacet.sol SignatureFacet.sol

## Final Review Scope

<b>Repository</b>	<a href="https://github.com/zapit-io/p2p-evmContract/tree/audit">https://github.com/zapit-io/p2p-evmContract/tree/audit</a>
<b>Commit Hash</b>	e249fc03bb4e721165606021cd1db885c8ef3322
<b>Functional Requirements</b>	Partial documentation provided. README.md
<b>Technical Requirements</b>	Partial documentation provided. README.md
<b>Contracts Addresses</b>	-
<b>Contracts</b>	EscrowFacetERC20.sol EscrowFacet.sol AdminFacet.sol SignatureFacet.sol



## Severity Definitions

Risk Level	Description
<b>Critical</b>	Critical vulnerabilities are usually straightforward to exploit and can lead to asset loss or data manipulations.
<b>High</b>	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
<b>Medium</b>	Medium-level vulnerabilities are important to fix; however, they cannot lead to asset loss or data manipulations.
<b>Low</b>	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.
<b>Informational</b>	Issue listed to improve understanding, readability and quality of code

All statuses which are identified in the audit report are categorized here for the reader to review:

Status Type	Definition
<b>Open</b>	Risks are open.
<b>Acknowledged</b>	Risks are acknowledged, but not fixed.
<b>Resolved</b>	Risks are acknowledged and fixed.



## AUDIT SUMMARY

The SecureDApp team has performed a line-by-line manual analysis and automated review of smart contracts. Smart contracts were analyzed mainly for common contract vulnerabilities, exploits, and manipulation hacks. According to the audit:

Status	Critical	High	Medium	Low	Informative
Open	0	0	0	0	0
Acknowledged	1	0	0	0	0
Resolved	4	3	3	0	0



## AUDIT METHODOLOGY

SecureDApp scans contracts and reviews codes for common vulnerabilities, exploits, hacks and back- doors.

Mentioned are the steps used by SecureDApp to audit smart contracts:

- a. Smart contract source code reviewal:
  - i. Review of the specifications, sources, and instructions provided to SecureDApp to make sure we understand the audit scope, intended business behavior, overall architecture, and project's goal.
  - ii. Manual review of code, which is the process of reading source code line-by-line to identify potential vulnerabilities.
- b. Test coverage analysis: (Unit testing)
  - i. Test coverage analysis is the process of determining whether the test cases are covering the code and how much code is exercised when we run those test cases.
- c. Static analysis:
  - i. Run a suite of vulnerability detectors to find security concerns in smart contracts with different impact levels.
- d. Symbolically executed tests: (SMTChecker testing) (Taint analysis)
  - i. Symbolic execution is analyzing a program to determine what inputs cause each part of a program to execute.
  - ii. Check for security vulnerabilities using static and dynamic analysis
- e. Property based analysis (Fuzz tests)(Invariant testing)
  - i. Run the execution flow multiple times by generating random sequences of calls to the contract.
  - ii. Asserts that all the invariants hold true for all scenarios.
- f. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
- g. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

Automated 5S frameworks used to assess the smart contract vulnerabilities

- Consensys Tools
- SWC Registry
- Solidity Coverage
- Open Zeppelin Code Analyzer
- Solidity Shield Scan





We have audited the smart contracts for commonly known and more specific vulnerabilities. Below is the list of smart contract tests, vulnerabilities, exploits, and hacks:

ID	Description	Status
EEA 3.3	<a href="#">Oracle Manipulation</a>	N/A
EEA 3.3	<a href="#">Bad Randomness - VRF</a>	N/A
S60	<a href="#">Assembly Usage</a>	Passed
S59	<a href="#">Dangerous usage of block.timestamp</a>	Passed
EEA 3.7	<a href="#">Front-Running Attacks</a>	N/A
EEA 3.7	<a href="#">Back-Running Attacks</a>	N/A
EEA 3.7	<a href="#">Sandwich Attacks</a>	N/A
DASP	<a href="#">Gas Griefing Attacks</a>	Passed
DASP	<a href="#">Force Feeding</a>	Passed
SCSVS V2	<a href="#">Access Control</a>	Passed
DASP	<a href="#">Short Address Attack</a>	Passed
DASP	<a href="#">Checks Effects Interactions</a>	Passed
EEA 4.1	<a href="#">No Self-destruct</a>	Passed
SCSVS V14	<a href="#">Decentralized Finance Checks</a>	Passed



Slither Tests	<a href="#">Checks for ERC's conformance</a>	Passed
Coverage	<a href="#">Unit tests with 100% coverage</a>	-
Gas Reporter	<a href="#">Gas usage &amp; limitations</a>	Passed
Echidna Tests	<a href="#">Malicious input handling</a>	Passed
SWC-101	<a href="#">Integer Overflow and Underflow</a>	Passed
SWC-102	<a href="#">Outdated Compiler Version</a>	Passed
SWC-103	<a href="#">Floating Pragma</a>	Passed
SWC-104	<a href="#">Unchecked Call Return Value</a>	Passed
SWC-105	<a href="#">Unprotected Ether Withdrawal</a>	Passed
SWC-106	<a href="#">Unprotected SELF-DESTRUCT Instruction</a>	Passed
SWC-107	<a href="#">Re-entrancy</a>	Passed
SWC-108	<a href="#">State Variable Default Visibility</a>	Passed
SWC-109	<a href="#">Uninitialized Storage Pointer</a>	Passed
SWC-110	<a href="#">Assert Violation</a>	Passed
SWC-111	<a href="#">Use of Deprecated Solidity Functions</a>	Passed
SWC-112	<a href="#">Delegate Call to Untrusted Callee</a>	Passed



SWC-113	<a href="#">DoS with Failed Call</a>	Passed
SWC-114	<a href="#">Transaction Order Dependence</a>	Passed
SWC-115	<a href="#">Authorization through tx.origin</a>	Passed
SWC-116	<a href="#">Block values as a proxy for time</a>	Passed
SWC-117	<a href="#">Signature Malleability</a>	Passed
SWC-134	<a href="#">Message call with the hardcoded gas amount</a>	Passed
SWC-135	<a href="#">Code With No Effects (Irrelevant/Dead Code)</a>	Informational
SWC-136/SCSVS V3	<a href="#">Unencrypted Private Data On-Chain</a>	Passed



## SYSTEM OVERVIEW

Zapit provides users with P2P Buy & Sell of assets without the involvement of a central authority. Keeping the trades confidential and safe using an open source non-custodial escrow system. System architect uses four core facets contracts.

AdminFacet contract manages the administrative configurations of the smart contract system. It includes functions for setting and adjusting transaction fees, defining allowed currencies, configuring the arbitrator for dispute resolution, and implementing pausable features for maintenance or emergencies.

EscrowFacet contract facilitates peer-to-peer (P2P) transactions involving native chain tokens by providing a secure mechanism for holding funds in escrow during transactions, managing the deposit, release, and refund of these tokens. Similarly, the EscrowFacetERC20 contract handles P2P transactions specifically for ERC20 tokens, offering equivalent escrow services tailored to ERC20 tokens.

Finally, the SignatureFacet contract handles the verification of signed messages based on the EIP-712 standard, ensuring the integrity and authenticity of the signatures used in the system. The scope of the audit is the above four facets contracts.

### Privileged roles

1. Contract Owner Role : OwnershipFacet
  - a. Manage contract upgradability
  - b. Manage Admin functions:
    - i. Pause Contracts
    - ii. setWhitelistedCurrencies
    - iii. setArbitrator
    - iv. setFees and setFeeAddress
2. AdminFacet - Arbitrator Role:
  - a. claimDisputedOrder

### Risks

1. The impact of the owner role being compromised would have a huge impact on the protocol.
2. Centralization risk is the most common cause of cryptography asset loss.
3. Compromising the Owner Role may lead to all user's asset loss.



## FINDINGS

### Centralization Risk

Centralization risk is the most common cause of dapp's hacks. When a smart contract has an active contract ownership, the risk related to centralization is elevated. There are some well-intended reasons to be an active contract owner, such as:

- Contract owners can be granted the power to `pause()` or `lock()` the contract in case of an external attack.
- Contract owners can use functions like, `include()`, and `exclude()` to add or remove wallets from fees, swap checks, and transaction limits. This is useful to run a presale, and to list on an exchange.

Authorizing a full centralized power to a single body can be dangerous. Unfortunately, centralization related risks are higher than common smart contract vulnerabilities. Centralization of ownership creates a risk of rug pull scams, where owners cash out tokens in such quantities that they become valueless. Most important question to ask here is, how to mitigate centralization risk? Here's SecureDApp's recommendation to lower the risks related to centralization hacks:







- Smart contract owner's private key must be carefully secured to avoid any potential hack.
- Smart contract ownership should be shared by multi-signature (multi-sig) wallets.
- Smart contract ownership can be locked in a contract, user voting, or community DAO can be introduced to unlock the ownership.

### Zapit Centralization Status

- Zapit smart contract has Contract Ownership Role.



## STATIC ANALYSIS REPORT

Symbol	Meaning
:-----: -----	
	Function can modify state
	Function is payable
Contract	Type   Bases
<b>**EscrowFacet**</b>	Implementation   Modifiers, SignatureFacet
<b>**EscrowFacetERC20**</b>	Implementation   Modifiers, SignatureFacet
<b>**SignatureFacet**</b>	Implementation
<b>**AdminFacet**</b>	Implementation   Modifiers
<b>**Function Name**</b>	<b>**Visibility**</b>   <b>**Mutability**</b>   <b>**Modifiers**</b>
<b>**EscrowFacet**</b>	Implementation   Modifiers, SignatureFacet
createEscrowNative	External       nonReentrant nonContract onlyWhitelistedCurrencies
claimDisputedOrder	External       nonReentrant nonContract
executeOrder	External       nonReentrant nonContract
buyerCancel	External       nonReentrant nonContract
<b>**EscrowFacetERC20**</b>	Implementation   Modifiers, SignatureFacet
createEscrowERC20	External       nonReentrant nonContract onlyWhitelistedCurrencies
claimDisputedOrderERC20	External       nonReentrant nonContract
executeOrderERC20	External       nonReentrant nonContract



```
|| buyerCancelERC20 | External ! | ● | nonReentrant nonContract |
```

```
|||||
```

```
| **SignatureFacet** | Implementation | |||
```

```
|| getMessageHash | Public ! | |NO ! |
```

```
|| getEthSignedMessageHash | Public ! | |NO ! |
```

```
|| recoverSigner | Public ! | |NO ! |
```

```
|| splitSignature | Public ! | |NO ! |
```

```
|||||
```

```
| **AdminFacet** | Implementation | Modifiers |||
```

```
|| pause | External ! | ● | whenNotPaused onlyOwner |
```

```
|| unpause | External ! | ● | whenPaused onlyOwner |
```

```
|| setWhitelistedCurrencies | External ! | ● | onlyOwner |
```

```
|| setArbitrator | External ! | ● | onlyOwner |
```

```
|| setFees | Public ! | ● | onlyOwner |
```

```
|| setFeeAddress | External ! | ● | onlyOwner |
```

```
|| paused | External ! | |NO ! |
```

```
|| getWhitelistedCurrencies | External ! | |NO ! |
```

```
|| getArbitrator | External ! | |NO ! |
```

```
|| getFees | External ! | |NO ! |
```

```
|| getFeeAddress | External ! | |NO ! |
```

```
|| getEscrow | External ! | |NO ! |
```



## TRANSACTION GAS CHART

Solc version: 0.8.24		Optimizer enabled: false		Runs: 200	Block limit: 30000000 gas	
<b>Methods</b>						
Contract	Method	Min	Max	Avg	# calls	usd (avg)
AdminFacet	grantRole	-	-	55082	2	-
AdminFacet	pause	-	-	52483	2	-
AdminFacet	renounceRole	-	-	30751	1	-
AdminFacet	revokeRole	-	-	33183	1	-
AdminFacet	setArbitrator	-	-	35782	1	-
AdminFacet	setFeeAddress	-	-	32938	1	-
AdminFacet	setWhitelistedCurrencies	51826	52066	51946	2	-
AdminFacet	unpause	-	-	30580	2	-
DiamondCutFacet	diamondCut	211173	1356208	723668	3	-
EscrowFacet	buyerCancel	-	-	58004	1	-
EscrowFacet	claimDisputedOrder	65247	79231	72239	2	-
EscrowFacet	createEscrowNative	206432	243444	215685	8	-
EscrowFacet	executeOrder	-	-	79317	1	-
EscrowFacetERC20	buyerCancelERC20	-	-	63827	1	-
EscrowFacetERC20	claimDisputedOrderERC20	73606	88960	81283	2	-
EscrowFacetERC20	createEscrowERC20	-	-	271723	8	-
EscrowFacetERC20	executeOrderERC20	-	-	103699	1	-
OwnershipFacet	transferOwnership	-	-	34080	2	-
Token	increaseAllowance	30244	47344	33094	6	-
Token	mint	51756	68868	57464	3	-
<b>Deployments</b>					% of limit	
AdminFacet	-	-	-	1222737	4.1 %	-
Diamond	-	-	-	4122291	13.7 %	-
DiamondInit	-	-	-	489853	1.6 %	-
EscrowFacet	-	-	-	2090281	7 %	-
EscrowFacetERC20	-	-	-	2255250	7.5 %	-
SignatureFacet	-	-	-	465328	1.6 %	-
Token	-	-	-	1248365	4.2 %	-





## MANUAL REVIEW

Identifier	Definition	Severity
CEN-01	Centralization privileges of Zapit Contract Owner	Critical

Centralized privileges are listed below:

- Contract Owner Role : OwnershipFacet
  - Control contract upgradability
  - Manage Admin functions:
    - Pause Contracts
    - setWhitelistedCurrencies
    - setArbitrator
    - setFees and setFeeAddress

### RECOMMENDATION

Use Openzeppelin Access Control framework instead of Ownable module to avoid single point of failure. Usage of Multi-Sig wallet for authorisation is recommended. Please refer to CENTRALIZED PRIVILEGES for a detailed understanding.

**Status: Resolved**



Identifier	Definition	Severity
CEN-02	Protecting the Initialization Function in DiamondInit	Critical

DiamondInit contract includes an external init function that currently lacks access controls. This function initializes various storage variables and sets up supported interfaces, making it crucial for the contract's proper operation. Without access control, any external entity can call this function, potentially leading to malicious reinitialization or manipulation of the contract's state. This vulnerability can have severe consequences, including unauthorized fee adjustments, changing the arbitrator, or triggering unintended contract behaviors.

## RECOMMENDATION

Safeguard the init function from unauthorized access, implement an access control mechanism.

**Status: Resolved**



Identifier	Definition	Severity
CEN-03	Use of proxy and Diamond upgradeable pattern	Critical

Contract upgradeability allows privileged roles to change current contract implementation.

## RECOMMENDATION

Test and validate the current contract thoroughly before deployment. Future contract upgradeability negatively elevates centralization risk.

**Status: Acknowledged**



Identifier	Definition	Severity
CEN-04	Ensuring Proper Functionality of the Pausable Mechanism	Critical

In EscrowFacetERC20 and EscrowFacet contracts, the Pausable mechanism is not effectively halting all platform activities. This inadequacy can lead to vulnerabilities where the platform operations continue even when they should be paused, potentially exposing the platform to risks or misuse during these periods.

## RECOMMENDATION

To ensure the Pausable mechanism properly halts all platform activities, you need to integrate the whenNotPaused modifier into all critical functions that should be stopped when the contract is paused.

**Status: Resolved**



Identifier	Definition	Severity
CEN-05	Mismatch in Fee Calculation Precision Between Off-chain and On-chain Calculations	Critical

The contract calculates seller fees based on a precise formula involving `_value` and `ds.escrowFeeBP`, ensuring fees are deducted correctly. However, off-chain fee calculation in the client-side code uses a simplified calculation which may lead to precision mismatches.

## RECOMMENDATION

**Consistent Calculation Methods:** Ensure that fee calculations in both off-chain and on-chain environments use the same precise arithmetic methods to avoid discrepancies. Try to isolate fee calculation function in contract and use that to calculate even for off chain use cases.

**Status: Resolved**



Identifier	Definition	Severity
HGH-01	Improving Ether Transfer Methods in EscrowFacet Contract	High

The EscrowFacet contract currently uses the outdated transfer function for Ether transfers, which can fail due to gas limitations. The recommended approach is to use the call method for greater flexibility and reliability.

## RECOMMENDATION

Replace all instances of transfer with the call method to improve the security and reliability of Ether transfers. Ref: <https://solidity-by-example.org/sending-ether/>

**Status: Resolved**



Identifier	Definition	Severity
HGH-02	Enhancing Security with SafeERC20 Library in EscrowFacetERC20 Contract	High

The EscrowFacetERC20 contract currently performs unchecked ERC20 token transfers, which can lead to vulnerabilities such as failed transactions not being properly handled. Using the OpenZeppelin SafeERC20 library ensures that all token transfers are executed safely, handling potential errors gracefully and increasing overall contract security.

## RECOMMENDATION

Replace all instances of unchecked ERC20 token transfers with the SafeERC20 library to ensure safe and reliable token transactions.

**Status: Resolved**



Identifier	Definition	Severity
HGH-03	Missing Ether Withdrawal Function to EscrowFacetERC20 Contract	High

The EscrowFacetERC20 contract has payable functions but lacks a mechanism to withdraw Ether, which can lead to Ether being locked in the contract indefinitely. This is a critical issue as it can result in the loss of funds if Ether is accidentally sent to the contract. To prevent this, a withdrawal function should be implemented, allowing the contract owner to retrieve any Ether stored in the contract or remove the payable modifier.

## RECOMMENDATION

Add a function to withdraw Ether from the contract to ensure that any Ether received can be recovered.

**Status: Resolved**





Identifier	Definition	Severity
MED-01	Enhancing Test Coverage for Upgradability and Pausable Features	Medium

The current test suite lacks unit and end-to-end test cases for the upgradability functionality and the pausable features of the contract. This gap in test coverage can result in undetected issues or vulnerabilities, potentially compromising the reliability and security of the contract.

### RECOMMENDATION

Develop unit tests to validate the upgradability functionality and end-to-end tests to verify the pausability features.

**Status: Resolved**



Identifier	Definition	Severity
MED-02	Mitigating Risk with Multi-Signature Framework for Arbitrator Role	Medium

The current contract employs a single wallet arbitrator, which poses a risk if the private key associated with the arbitrator account is compromised.

### RECOMMENDATION

Integrate a multi-signature framework for contract arbitration to enhance security and mitigate the risk of private key exposure.

**Status: Resolved**



Identifier	Definition	Severity
MED-03	Enhancing Contract Stability with Fixed Pragma Directive	Medium

The contract currently utilizes a pragma version that may introduce breaking changes or unexpected behavior due to its recentness.

### RECOMMENDATION

To ensure stability and reliability, it is advisable to use a fixed pragma directive, such as version 0.8.18, which has undergone extensive testing and is less likely to encounter compatibility issues.

**Status: Resolved**



## UNIT TEST REPORT

Zapit Unit Test Cases

```
const diamond = '0x5FbDB2315678afecb367f032d93F642f64180aa3'
```

```
const diamondInit = '0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512'
```

```
Token deployed: 0xa513E6E4b8f2a923D98304ec87F64353C4D5C853
```

- ✓ UPGRADABILITY: Check all facets within diamond
- ✓ UPGRADABILITY: Remove a AdminContract from diamond
- ✓ UPGRADABILITY: Check all facets within diamond
- ✓ UPGRADABILITY: Fail to call methods from admin contract
- ✓ UPGRADABILITY: Add a AdminContract to diamond
- ✓ UPGRADABILITY: Check all facets within diamond
- ✓ UPGRADABILITY: Fail to add AdminContract to diamond again
- ✓ UPGRADABILITY: Non owner cannot remove AdminContract from diamond
- ✓ ADMIN: [OWNERSHIP] Should fetch and verify the ownership of the contract
- ✓ ADMIN: [OWNERSHIP] Should transfer ownership to account[1] i.e arbitrator
- ✓ ADMIN: [OWNERSHIP] Should transfer ownership back to account[0] i.e deployer
- ✓ ADMIN: [ARBITER] Should verify the arbiter
- ✓ ADMIN: [FEE ADDRESS] Should fetch the default market fee address
- ✓ ADMIN: [FEE ADDRESS] SET+GET Should set and fetch the default market fee address
- ✓ ADMIN: [Fees] Should fetch the market fee set
- ✓ ADMIN: [PAUSABLE] Market should not be paused initially
- ✓ ADMIN: [ROLE] Check deployer hasRole
- ✓ ADMIN: [ROLE] secondaryDeployer hasRole must be false
- ✓ ADMIN: [ROLE] Grant Role to secondaryDeployer
- ✓ ADMIN: [ROLE] secondaryDeployer hasRole must be true
- ✓ ADMIN: [ROLE] secondaryDeployer must be able to pause and unpaue the market
- ✓ ADMIN: [ROLE] Revert Unauthorized Grant Role invocation from non admin account
- ✓ ADMIN: [ROLE] Revoke Role
- ✓ ADMIN: [ROLE] secondaryDeployer hasRole must be false
- ✓ ADMIN: [ROLE] Assign role to secondaryDeployer and it must renounce the Role
- ✓ CORE: [DimaondInit] Should not be able to execute as it can only be called by the owner
- ✓ CORE: [DimaondInit] Oth storage slot must be owner for diamind and address(0) for diamond init
- ✓ ADMIN [PAUSABLE] Should pause the market
- ✓ PAUSABLE: Fail to create order due to paused contract
- ✓ PAUSABLE: Should unpaue the market
- ✓ WHITELIST: Fail to create order as currency is not whitelisted
- ✓ WHITELIST: Whitelist base currency
- ✓ EscrowFacet: Create and Complete Native currency trade
- ✓ EscrowFacet: Create and Cancel order
- ✓ EscrowFacet: Create and Claim dispute (Buyer)
- ✓ EscrowFacet: Create and Claim dispute (Seller)
- ✓ EscrowFacetERC20Contract: Create and Complete Native currency trade
- ✓ EscrowFacet: Create and Cancel order
- ✓ EscrowFacet: Create and Claim dispute (Seller)
- ✓ EscrowFacet: Create and Claim dispute (Buyer)

19 passing (913ms)



## DISCLAIMER

SecureDApp Auditors provides the easy-to-understand audit of solidity source codes (commonly known as smart contracts).

The smart contract for this particular audit was analyzed for common contract vulnerabilities, and centralization exploits. This audit report makes no statements or warranties on the security of the code. This audit report does not provide any warranty or guarantee regarding the absolute bug-free nature of the smart contract analyzed, nor do they provide any indication of the client's business, business model or legal compliance. This audit report does not extend to the compiler layer, any other areas beyond the programming language, or other programming aspects that could present security risks. Cryptographic tokens are emergent technologies, they carry high levels of technical risks and uncertainty. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. This audit report could include false positives, false negatives, and other unpredictable results.

### CONFIDENTIALITY

This report is subject to the terms and conditions (including without limitations, description of services, confidentiality, disclaimer and limitation of liability) outlined in the scope of the audit provided to the client. This report should not be transmitted, disclosed, referred to, or relied upon by any individual for any purpose without SecureDApp's prior written consent.

### NO FINANCIAL ADVICE

This audit report does not indicate the endorsement of any particular project or team, nor guarantees its security. No third party should rely on the reports in any way, including to make any decisions to buy or sell a product, service or any other asset. The information provided in this report does not constitute investment advice, financial advice, trading advice, or any other sort of advice and you should not treat any of the report's content as such. This audit report should not be used in any way to make decisions around investment or involvement. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. For avoidance of doubt, services, including any associated audit reports or materials, shall not be considered or relied upon as any form of financial, tax, legal, regulatory, or other advice.

### TECHNICAL DISCLAIMER



All services, audit reports, smart contract audits, other materials, or any products or results of the use thereof are provided “as is” and “as available” and with all faults and defects without warranty of any kind. To the maximum extent permitted under applicable law, SecureDApp hereby disclaims all warranties, whether expressed, implied, statutory, or otherwise with respect to services, audit report, or other materials. Without limiting the foregoing, SecureDApp specifically disclaims all implied warranties of merchantability, fitness for a particular purpose, title and non-infringement, and all warranties arising from the course of dealing, usage, or trade practice. Without limiting the foregoing, SecureDApp makes no warranty of any kind that all services, audit reports, smart contract audits, or other materials, or any products or results of the use thereof, will meet the client’s or any other individual’s requirements, achieve any intended result, be compatible or work with any software, system, or other services, or be secure, accurate, complete, free of harmful code, or error-free.

### TIMELINESS OF CONTENT

The content contained in this audit report is subject to change without any prior notice. SecureDApp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following the publication.

### LINKS TO OTHER WEBSITES

This audit report provides, through hypertext or other computer links, access to websites and social accounts operated by individuals other than SecureDApp. Such hyperlinks are provided for your reference and convenience only and are the exclusive responsibility of such websites’ and social accounts’ owners. You agree that SecureDApp is not responsible for the content or operation of such websites and social accounts and that SecureDApp shall have no liability to you or any other person or entity for the use of third-party websites and social accounts. You are solely responsible for determining the extent to which you may use any content at any other websites and social accounts to which you link from the report.



## ABOUT SECUREDAPP

SecureDApp Auditor provides intelligent blockchain solutions. SecureDApp is developing an ecosystem that is seamless and responsive. Some of our services: Blockchain Security, Token Launchpad, NFT Marketplace, etc. SecureDApp's mission is to interconnect multiple services like Blockchain Security, DeFi, Gaming, and Marketplace under one ecosystem that is seamless, multi-chain compatible, scalable, secure, fast, responsive, and easy to use.

SecureDApp is built by a decentralized team of UI experts, contributors, engineers, and enthusiasts from all over the world. Our team currently consists of 6+ core team members, and 10+ casual contributors. SecureDApp provides manual, static, and automatic smart contract analysis, to ensure that the project is checked against known attacks and potential vulnerabilities.

To learn more, visit : <https://securedapp.in/>

To view our audit portfolio, visit : [github.securedapp.in](https://github.com/securedapp)

To book an audit, message : [securedapp.telegram](https://t.me/securedapp)





[Securedapp.in](mailto:Securedapp.in)



[Securedapp\\_Linkedin](#)



[Securedapp\\_Telegram](#)