Ultrade CODEX

Ultrade EVM

Smart Contract Audit Report

DATE: 8th Nov 2024 | **PREPARED BY:** Entersoft Pty Ltd

Contents

Revision History & Version Control

Version	Date	Author(s)	Description
1.0	08-Nov-2024	Gurkirat singh, Shashank, Laxmi Prasad	Interim Report

1. Executive Summary

Entersoft has conducted a static and dynamic smart contract audit of the Ultrade EVM project through a comprehensive smart contract audit. The primary objective was to identify potential vulnerabilities and risks within the codebase, ensuring adherence to industry-leading standards while prioritizing security, reliability, and performance. Our focus was on prompt and efficient identification and resolution of vulnerabilities to enhance the overall robustness of the solidity smart contract.

Testing Methodology:

Our testing methodology in Solidity adhered to industry standards and best practices, integrating partially implemented OWASP and NIST SP 800 standards for encryption and signatures.

We have performed a detailed manual analysis, adherence to industry standards, and the use of a comprehensive toolset. Our approach ensured a thorough evaluation within the designated Solidity code files.

Findings and Security Posture:

Our primary focus was on Access Control Policies, Transaction Signature Validations, Reentrancy, Time Manipulation, Default Visibility, Outdated Compiler Version, Input Validation, Deprecated Solidity Functions, Shadowing State Variables, Presence of Unused Variables, Overflow and Underflow Conditions, Assets Integrity, Errors and Exception.

Importantly, our audit process intentionally avoided reliance solely on automated tools, emphasizing a more in-depth and nuanced approach to security analysis. Conducted from June 6, 2024 to November 6, 2024, our team diligently assessed and validated the security posture of the solidity smart contract, ultimately classifying it as "Secure," reflecting the absence of identified vulnerabilities and the robustness of the codebase against potential threats.

2. Overview

2.1 Project Overview

Entersoft has meticulously audited the smart contract project from August 1, 2024 to November 8, 2024 with a primary focus on Solidity code files integral to blockchain functionality, emphasizing vulnerabilities in associated gas claiming. The working of basic functionalities was also tested during the review.

2.2. Scope

The audit scope covers the CODEX-dev smart contract available in the GitHub private repository:

Github	https://github.com/ultrade-org/CODEX
Branch	Dev Branch
Commit ID	2b696ebe28016021a996d7172effaf79ee3dc339
Scope	codex\CODEX-dev\dev\contracts\chains\evm

OUT-OF-SCOPE: External contracts, other imported smart contracts.

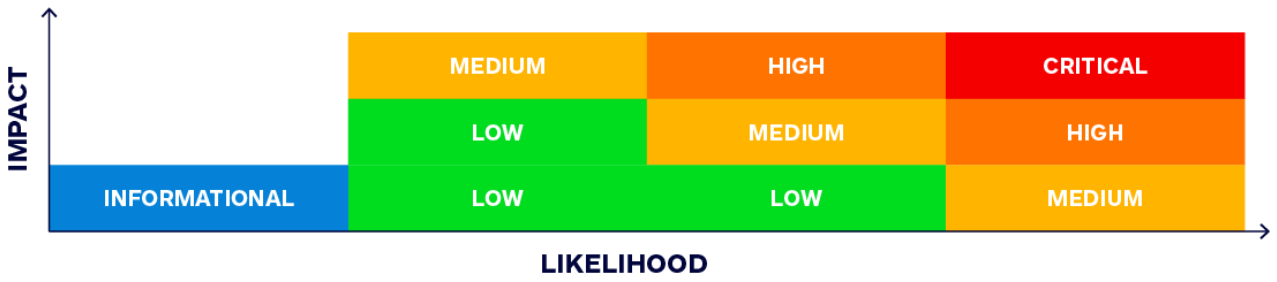
2.3 Summary of Findings

The following table is the summary of findings, which summarizes the overall risks identified during the audit.

Critical	High	Medium	Low	Informational
0	0	1	5	4

2.4 Security Level References

Every vulnerability in this report was assigned a severity level from the following classification table:



3. Comprehensive Analysis findings: A Dual Approach through Static and Manual Examination

Phase 1: Static Analysis

In our rigorous smart contract audit process, we employed a multifaceted approach combining both static and dynamic analysis methodologies to comprehensively evaluate the security posture of the protocol. Our static analysis phase involved the utilization of static analyzing tools such as Slither, Aderyn. These tools enabled us to conduct automated code analysis to identify potential vulnerabilities within the smart contracts.

Tools and Efforts:

Slither, Aderyn were instrumental in performing static analysis, allowing us to efficiently scan the codebase for common vulnerabilities such as reentrancy, denial-of-service (DOS) attacks, front-running vulnerabilities, time dependencies, token approval issues, and arithmetic errors. Through meticulous examination of the code, we meticulously identified and categorized potential risks, laying the groundwork for further in-depth analysis.

Phase 2: Dynamic Analysis

After the static analysis phase, we transitioned to dynamic analysis, which involved a more hands-on approach to scrutinizing the intended functionality and security of the smart contracts. Further, we devised a comprehensive suite of unit tests to validate the expected behavior of the smart contracts under various scenarios.

Processes and Test Cases:

Our dynamic analysis encompassed a systematic exploration of the smart contracts' functionalities, focusing on critical areas prone to vulnerabilities. We meticulously crafted test cases to assess the resilience of the contracts against potential attack vectors, including but not limited to reentrancy attacks, DOS vulnerabilities, front-running exploits, time-sensitive vulnerabilities, token approval vulnerabilities, and arithmetic errors.

Throughout both the static and dynamic analysis phases, our team dedicated substantial efforts to meticulously review the codebase, identify vulnerabilities, and develop robust test cases to assess the security posture of the protocol comprehensively. By combining automated analysis with manual examination and testing, we ensured a thorough evaluation of the smart contracts, ultimately enhancing the security and reliability of the protocol.

4. Vulnerabilities

Note:

The following values for “Severity” mean:

- Critical: This vulnerability poses a direct and severe threat to the funds or the main functionality of the protocol.
- High: Direct impact on the funds or the main functionality of the protocol.
- Medium: Indirect impact on the funds or the protocol's functionality.
- Low: Minimal to no impact on the funds or the protocol's main functionality.

The following values for “Result” mean:

- PASS: indicates that there is no security risk.
- FAIL : indicates that there is a security risk that needs to be remediated.
- Informational : Suggestions related to good coding practices and gas-efficient code.
- **Not Applicable**: means the attack vector is Not applicable or Not available

4.1 Private state variable is being returned in a public function

Severity	Medium
Description	The private state variable `current_msg` is being returned by the public function `getCurrentMsg()`, which exposes the private variable externally. This contradicts the intended behavior of a private variable. Returning private variables through public functions undermines the purpose of keeping them private.
Location / Source File	TokenManagerUpgradeable – 305
Observation	The function `getCurrentMsg()` is public but returns the private state variable `current_msg`. Since `getCurrentMsg()` is publicly accessible, this makes the private variable publicly available within the contract, contradicting the principle that private variables should not be exposed publicly.

Remediation	Make the function `getCurrentMsg()` as private/internal
	
	Add
	
	`function getCurrentMsg() private view returns (string memory) { return current_msg; }`
	
	Remove
	
	`function getCurrentMsg() public view returns (string memory) {`
	`  return current_msg;`
	`}`

Remark	Issue fixed in commit:

4.2 Bytes data type is being used instead of bytes32 for login Address

Severity	Low
Description	In the functions `depositToCodex` and `depositGasTokenToCodex` bytes type is being used for the `loginAddress`. The `loginAddress` must be of 32 bytes of length which is being checked by require statements inside the functions. In this regard `bytes memory loginAddress` can be replaced by `bytes32 loginAddress`, by doing this we can also remove the require statement as it's already 32 bytes now. It will save gas and make the implementation more efficient.
Location / Source File	TokenManagerUpgradeable – 123, 136
Observation	`bytes` data type uses more gas than `bytes32` as it is a dynamic data type. If `bytes` is used for `loginAddress`, an additional required check is needed inside the function to verify that the value is 32 bytes in length. By using the bytes32 data type for loginAddress, additional checks for length are unnecessary, which results in gas savings.
Remediation	Change the data type of `loginAddress` from `bytes` to `bytes32`. Add: • `bytes32 loginAddress` Remove: • `bytes memory loginAddress` • `require(loginAddress.length == 32, "Check loginAddress has the normalized type");`
Remark	Issue fixed in commit:

4.3 Functions not used internally could be marked as external

Severity	Low
Description	Functions intended to be called externally by users or other contracts, rather than internally within the contract, should be marked as `external` instead of `public` for clarity and potential optimization.
Location / Source File	TokenManagerUpgradeable
Observation	The functions are declared as `public`, but they are not called internally within the contract. Since they are meant to be called externally, marking them as `external` would make the code more gas-optimized.
Remediation	Change the visibility of these functions to external if they are intended to be called from outside the contract. This improves readability and provides a hint to developers about the intended usage.
Remark	Issue fixed in commit:

4.4 Missing Zero Address Validation

Severity	Low
Description	Zero address validation is a security measure to prevent transactions to or from a specific address that represents nothing. Missing this validation could allow accidental or malicious transfers to this address, resulting in permanently locked funds that cannot be recovered.
Location / Source File	TokenManagerUpgradeable – 84,86,87,
Observation	Missing zero address validation in TokenManagerUpgradeable.initialize() can lead to accidental or malicious transfers to the zero address, resulting in permanently locked and unrecoverable funds. This oversight, affecting the wormhole_core_bridge_address, cctpUSDC, and wrappedNative, compromises the security and reliability of the contract, potentially causing financial losses and diminishing user trust.
Remediation	To address this vulnerability, implement a validation step to ensure that the addresses involved are not zero. By incorporating this check, the risk of unintended ownership transfers or other undesirable outcomes can be mitigated effectively.
Remark	Issue fixed in commit:

```
66  //TokenManagerUpgradeable.sol
67      function initialize(
68          // address initialOwner,
69          address _wormholeRelayer,
70          address _wormhole,
71          address _circleMessageTransmitter,
72          address _circleTokenMessenger,
73          address _USDC,
74          address _WETH
75      ) public initializer {
76          __Pausable_init();
77          // __Ownable_init(initialOwner);
78          __Ownable_init(msg.sender);
79          __UUPSUpgradeable_init();
80          __WormholeCCTP_init(_wormholeRelayer, _wormhole, _circleMessageTransmitter, _circleTokenMessenger, _USDC);
81
82          GAS_LIMIT = 250_000;
83          UNIFIED_USDC = "CCTPUSDC";
84          wormhole_core_bridge_address = _wormhole;
85          core_bridge = IWormhole(wormhole_core_bridge_address);
86          cctpUSDC = _USDC;
87          wrappedNative = _WETH;
88          nonce = 0;
89
```

4.5 Uninitialized Local Variables

Severity	Low
Description	Uninitialized local variables in Solidity smart contracts can lead to unpredictable behavior and security vulnerabilities. These variables have default values leading to logical errors and potential exploits. Using uninitialized local variables can result in incorrect calculations, faulty contract states, and unintended contract behavior.
Location / Source File	WormholeCCTP.sol – 259-283
Observation	Uninitialized local variables in Solidity smart contracts, such as amountUSDCReceived in WormholeCCTP.receiveWormholeMessages(), can lead to unpredictable behaviour and security vulnerabilities. This can cause incorrect calculations, faulty contract states, and unintended contract behaviour, potentially resulting in financial losses and exploitation by malicious actors.

Remediation	Entersoft recommends that, for uninitialized local variables in Solidity contracts, it's crucial to initialize all variables properly. Explicitly set each variable to its intended initial value upon declaration. Even if a variable is meant to start at zero, assign it explicitly for clarity.
Remark	Issue fixed in commit:

4.6 Usage of different Pragma directives

Severity	Low
Description	Usage of different Pragma directives
Location / Source File	TokenManagerUpgradeable

Observation	Pragma directives are compiler instructions that can optimize performance, manage warnings, and control data alignment. They enhance code efficiency and readability but may introduce bugs or runtime errors if misused. Proper usage balances benefits and potential risks.
	`//TokenManagerUpgradeable.sol`
	`pragma solidity ^0.8.13;`
	`//WormholeCCTP.sol`
	`pragma solidity ^0.8.13;`
	`//BytesLib.sol`
	`pragma solidity >=0.8.0 <0.9.0;`
	`//IWETH.sol`
	`pragma solidity ^0.8.13;`

Remediation	To remediate the issue of different Solidity versions being used, standardize on a single, stable version of Solidity for all contracts and their dependencies. This approach minimizes compatibility issues and potential security vulnerabilities, leading to more robust and secure smart contracts. Use ^ or >= Carefully: When specifying versions, using ^ (e.g., pragma solidity ^0.8.0;) or >= (e.g., pragma solidity >=0.8.0;) allows for some flexibility but can introduce compatibility risks if not managed carefully.
Remark	Issue fixed in commit:

4.7 Constants should be defined and used instead of literals

Severity	Informational

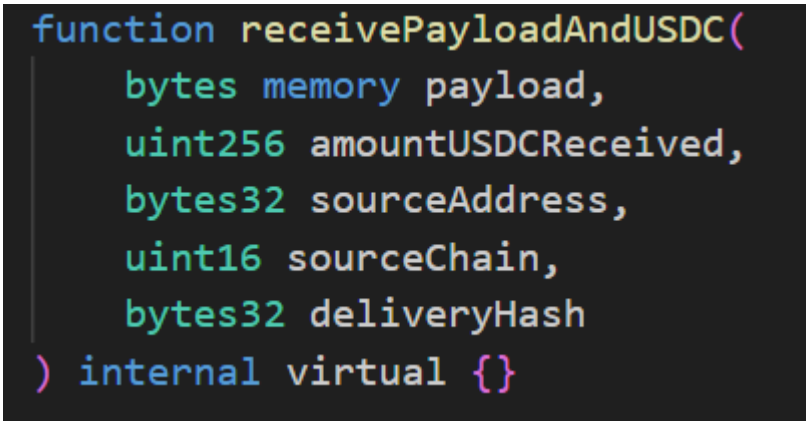
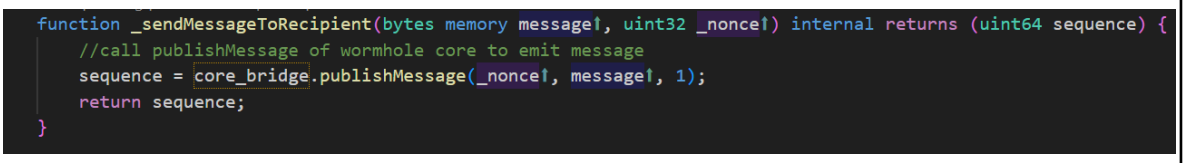
Description	Constants should be declared instead of using literals in the code like in ` require(loginAddress.length == 32, "Check loginAddress has the normalized type"); ` These literals represent fixed values, fees, or conditions.
Location / Source File	TokenManagerUpgradeable.sol

Observation	It makes the code more efficient.
	`function depositToCodex(address token, uint256 amount, bytes memory loginAddress, uint256 loginChainId) public returns (`
	`    //check login address`
	`    require(loginAddress.length == 32, "Check loginAddress has the normalized type");`
	`    //receive token from user wallet`
	`    _transferFrom(token, amount);`
	`    //emit deposit message to wormhole`
	`    sequence = _sendDepositMessage(token, amount, loginAddress, loginChainId);`
	`    return sequence;`
	`}`
	`function depositGasTokenToCodex(bytes memory loginAddress, uint256 loginChainId) public payable returns (uint64 sequence`
	`    //check login address`
	`    require(loginAddress.length == 32, "Check loginAddress has the normalized type");`
	`    //check deposit amount`
	`    require(msg.value > 0, "Deposit amount should be bigger than zero");`
	`    //convert ETH to WETH`
	`    IWETH(wrappedNative).deposit{value: msg.value}();`

Remediation	Replace the literals with the defined constants to improve code readability and maintainability.
Remark	Issue fixed in commit:

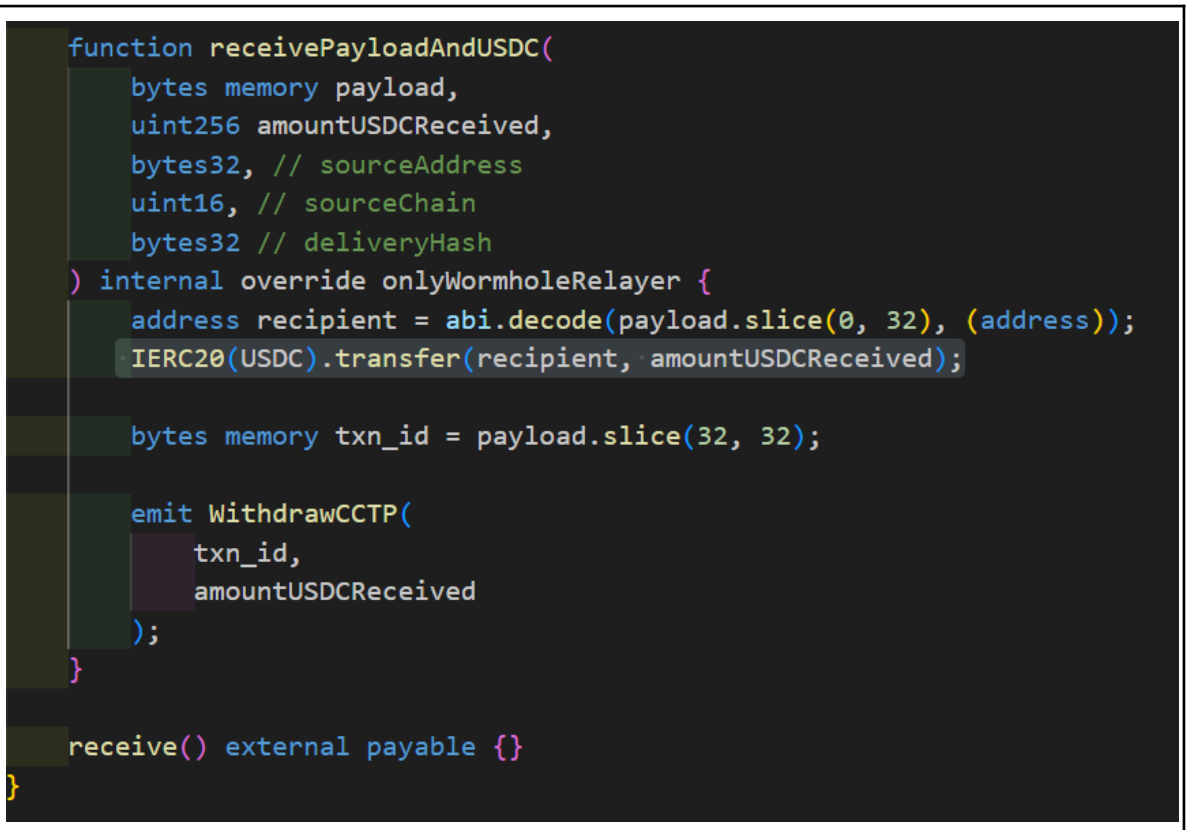
4.8 Presence of Dead Code

Severity	Informational
Description	Dead code in Solidity refers to functions or sections of code within a smart contract that are declared but never called or executed during the contract's lifecycle. These unused functions can accumulate over time due to changes in contract requirements or development iterations, resulting in unnecessary complexity and potentially increasing the contract's attack surface.
Location / Source File	TokenManagerUpgradeable – 13-89,183-187,300-306
Observation	In Entersoft's review of the Solidity smart contract, we identified the presence of dead code sections of the codebase that are never invoked or utilized throughout the contract's operations. The existence of such a dead code entails several significant drawbacks. Primarily, it introduces superfluous complexity, thereby complicating the understanding and maintenance of the contract. This added complexity can lead to increased deployment costs due to the larger contract size. Additionally, dead code enlarges the contract's attack surface, potentially obscuring vulnerabilities and unintended behaviors.

	 
Remediation	Entersoft recommends removing or commenting out these unused portions to eliminate dead code, thereby ensuring a streamlined and efficient contract. Moreover, dead code enlarges the contract's attack surface, potentially concealing vulnerabilities and unintended behaviors.
Remark	Issue fixed in commit:

4.9 Unchecked Transfer

Severity	Informational
Description	Unchecked transfer vulnerabilities occur when the return value of an external transfer or transferFrom call in a smart contract is not validated. In blockchain and smart contract development, especially with Ethereum's ERC-20 tokens, these functions are used to move tokens between addresses.
Location / Source File	TokenManagerUpgradeable – 319-337
Observation	Ignoring the return value of IERC20(USDC).transfer() in TokenManagerUpgradeable.receivePayloadAndUSDC() can lead to significant issues. Ensuring the return value is checked is crucial for maintaining contract integrity and user trust. In Solidity 0.8.0 and later, the `transfer` function will automatically revert if it fails, so there's no need for an explicit check of the return value. However, it's good practice to explicitly handle potential failures to ensure your contract is robust and clear in its error handling.

	
Remediation	Entersoft recommends implementing the following measures to ensure robust handling of token transfers. • Always verify the return values of transfer functions to confirm successful execution, using constructs such as require(token.transfer(recipient, amount), "Transfer failed");. • Additionally, adopting safe transfer functions from trusted libraries, like OpenZeppelin's SafeERC20, can automate these checks and ensure proper handling, enhancing the security and reliability of token operations.
Remark	Issue fixed in commit:

```
    function receivePayloadAndUSDC(
        bytes memory payload,
        uint256 amountUSDCReceived,
        bytes32, // sourceAddress
        uint16, // sourceChain
        bytes32 // deliveryHash
    ) internal override onlyWormholeRelayer {
        address recipient = abi.decode(payload.slice(0, 32), (address));
        IERC20(USDC).transfer(recipient, amountUSDCReceived);

        bytes memory txn_id = payload.slice(32, 32);

        emit WithdrawCCTP(
            txn_id,
            amountUSDCReceived
        );
    }

    receive() external payable {}
}
```

4.10 Unused State Variable

Severity	Informational
Description	The issue typically indicates that a state variable defined within a Solidity smart contract is not utilized anywhere within the contract's logic. This situation arises when a variable is declared but never referenced or manipulated in any of the contract's functions or statements. This results in potential increases in gas prices due to inefficient code execution, contributes to storage problems, and exacerbates performance issues.
Location / Source File	WormholeCCTP – 62

Observation	Throughout the entire contract, these sections of code remain unused. The ramifications of this issue transcend mere storage consumption, affecting gas costs, code clarity, maintainability, readability, and ultimately, security. Resolving this matter is imperative for optimizing gas usage, elevating code quality, and fortifying the overall efficiency and security of the smart contract. `uint256[50] __gap;`
Remediation	Entersoft suggests remediating the issue by identifying and removing or repurposing any unused variables within the contract's logic. By optimizing storage usage and improving code clarity, you can enhance the efficiency, readability, and security of the smart contract.
Remark	Issue fixed in commit:

5. Tested for Scenarios

5.1 Successful USDC Deposit to Codex

Result	**PASS**
Function	depositToCodex
Objective	To verify that the depositToCodex function correctly transfers USDC from a user to the TokenManagerUpgradeable contract and emits the appropriate event.
Test Case	

```
function testDepositToCodex() public {
    uint256 amount = 1 * 10 ** usdc.decimals();
    bytes memory loginAddress = abi.encodePacked(bytes32("loginAddress"));

    // Impersonate MetaMask address
    vm.startPrank(user);

    // Ensure MetaMask address has enough USDC
    uint256 initialBalance = usdc.balanceOf(user);
    require(
        initialBalance >= amount,
        "Not enough USDC in MetaMask address"
    );

    // Approve tokenManager to spend MetaMask's USDC
    usdc.approve(tokenManagerAddress, amount);

    // Call depositToCodex
    tokenManager.depositToCodex(address(usdc), amount, loginAddress, 1);

    // Stop impersonation
    vm.stopPrank();

    // Verify the USDC balance of tokenManager
    uint256 tokenManagerBalance = usdc.balanceOf(tokenManagerAddress);
    assertEq(tokenManagerBalance, amount);
}
```

Expected Behavior	This test checks if a user can successfully deposit a specified amount of USDC into the contract, ensuring the balance is updated correctly and the event is emitted.

5.2 Invalid Login Address Handling in Deposit

Result	**PASS**
Function	depositToCodex
Objective	To ensure the depositToCodex function reverts when provided with an invalid login address.
Test Case	

```
function testDepositToCodexWithInvalidLoginAddress() public {
    uint256 amount = 1 * 10 ** usdc.decimals();
    bytes memory invalidLoginAddress = "invalidAddress"; // Invalid login address (not 32 bytes)

    // Impersonate MetaMask address
    vm.startPrank(user);

    // Ensure MetaMask address has enough USDC
    uint256 initialBalance = usdc.balanceOf(user);
    require(
        initialBalance >= amount,
        "Not enough USDC in MetaMask address"
    );

    // Approve tokenManager to spend MetaMask's USDC
    usdc.approve(tokenManagerAddress, amount);

    // Expect the transaction to revert due to invalid login address
    vm.expectRevert("Check loginAddress has the normalized type");
    tokenManager.depositToCodex(
        address(usdc),
        amount,
        invalidLoginAddress,
        1
    );
    vm.stopPrank();
}
```

Expected Behavior	This test verifies that the function correctly handles an invalid login address (not 32 bytes) by reverting the transaction.

5.3 Zero Amount Deposit Rejection

Result	**PASS**
Function	depositToCodex
Objective	To verify that the depositToCodex function reverts when attempting to deposit a zero amount.
Test Case	

```
function testDepositToCodexWithZeroAmount() public {
    uint256 zeroAmount = 0;
    bytes memory loginAddress = abi.encodePacked(bytes32("loginAddress"));

    // Impersonate MetaMask address
    vm.startPrank(user);

    // Ensure MetaMask address has enough USDC (though it won't be used in this test)
    uint256 initialBalance = usdc.balanceOf(user);
    require(
        initialBalance >= zeroAmount,
        "Not enough USDC in MetaMask address"
    );

    // Approve tokenManager to spend MetaMask's USDC
    usdc.approve(tokenManagerAddress, zeroAmount);

    // Expect the transaction to revert due to zero amount
    vm.expectRevert("You need to sell at least some tokens");
    tokenManager.depositToCodex(address(usdc), zeroAmount, loginAddress, 1);

    // Stop impersonation
    vm.stopPrank();
}
```

Expected Behavior	This test ensures that the function checks for a non-zero amount and reverts if the amount is zero.

5.4 Insufficient Allowance Handling in Deposit

Result	**PASS**
Function	depositToCodex
Objective	To ensure the depositToCodex function reverts when the allowance is less than the deposit amount.
Test Case	

```
function testDepositToCodexWithInsufficientAllowance() public {
    uint256 amount = 2 * 10 ** usdc.decimals();
    uint256 insufficientAllowance = 1 * 10 ** usdc.decimals(); // Less than the deposit amount
    bytes memory loginAddress = abi.encodePacked(bytes32("loginAddress"));

    // Impersonate MetaMask address
    vm.startPrank(user);

    // Ensure MetaMask address has enough USDC
    uint256 initialBalance = usdc.balanceOf(user);
    require(
        initialBalance >= amount,
        "Not enough USDC in MetaMask address"
    );

    // Approve tokenManager to spend less than the deposit amount
    usdc.approve(tokenManagerAddress, insufficientAllowance);

    // Expect the transaction to revert due to insufficient allowance
    vm.expectRevert("Check the token allowance");
    tokenManager.depositToCodex(address(usdc), amount, loginAddress, 1);

    // Stop impersonation
    vm.stopPrank();
}
```

Expected Behavior	This test checks that the function correctly handles cases where the user's allowance is insufficient for the deposit.

5.5 Insufficient Balance Handling in Deposit

Result	**PASS**
Function	depositToCodex
Objective	To verify that the depositToCodex function reverts when the user's balance is insufficient for the deposit.
Test Case	

```
function testDepositToCodexWithInsufficientBalance() public {
    uint256 amount = 15 * 10 ** usdc.decimals();
    bytes memory loginAddress = abi.encodePacked(bytes32("loginAddress"));

    // Impersonate MetaMask address
    vm.startPrank(user);

    // Approve tokenManager to spend the insufficient balance
    usdc.approve(tokenManagerAddress, amount);

    // Expect the transaction to revert due to insufficient balance
    vm.expectRevert("ERC20: transfer amount exceeds balance");
    tokenManager.depositToCodex(address(usdc), amount, loginAddress, 1);

    // Stop impersonation
    vm.stopPrank();
}
```

Expected Behavior	This test ensures that the function checks the user's balance and reverts if it is less than the deposit amount.

5.6 Successful Gas Token (ETH) Deposit to Codex

Result	PASS
Function	depositGasTokenToCodex
Objective	To verify that the depositGasTokenToCodex function correctly converts ETH to WETH and deposits it into the contract.
Test Case	
Expected Behavior	This test checks if a user can successfully deposit ETH, which is converted to WETH, ensuring the balance is updated correctly.

```
function testDepositGasTokenToCodex() public {
    uint256 amount = 0.01 ether; // Valid amount of gas token (ETH)
    bytes memory loginAddress = abi.encodePacked(bytes32("loginAddress"));

    // Impersonate MetaMask address
    vm.startPrank(user);

    // Fund MetaMask address with ETH
    vm.deal(user, amount);

    // Call depositGasTokenToCodex
    tokenManager.depositGasTokenToCodex{value: amount}(loginAddress, 1);

    // Stop impersonation
    vm.stopPrank();

    // Verify the WETH balance of tokenManager
    uint256 wethBalance = weth.balanceOf(tokenManagerAddress);
    assertEq(wethBalance, amount);

}
```

5.7 Invalid Chain ID Handling in Cross-Chain Deposit Quote

Result	PASS
Function	quoteCrossChainDeposit
Objective	To ensure the quoteCrossChainDeposit function reverts when provided with an invalid chain ID.
Test Case	
Expected Behavior	This test verifies that the function handles invalid chain IDs by reverting the transaction.

```
function testQuoteCrossChainDepositWithInvalidChainId() public {
    uint256 invalidTargetChainId = 99999; // Example invalid target chain ID

    // Expect the transaction to revert due to invalid chain ID
    vm.expectRevert(); // Adjust the expected revert message if the contract provides a specific one
    tokenManager.quoteCrossChainDeposit(uint16(invalidTargetChainId));
}
```

5.8 Successful Application Contract Registration

Result	**PASS**
Function	registerApplicationContracts
Objective	To verify that the registerApplicationContracts function correctly registers application contracts for a given chain ID.
Test Case	
Expected Behavior	This test checks that the function updates the mapping of application contracts when called by the owner.

```
function testRegisterApplicationContracts() public {
    uint16 validChainId = 1; // Example valid chain ID
    bytes32 validApplicationAddress = bytes32("applicationAddrs"); // Example valid application address

    // Impersonate the owner address
    vm.startPrank(tokenManager.owner());
    // Call registerApplicationContracts
    tokenManager.registerApplicationContracts(
        validChainId,
        validApplicationAddress
    );

    // Stop impersonation
    vm.stopPrank();

    // Verify the application contract was registered correctly
    bytes32 registeredAddress = tokenManager._applicationContracts(
        validChainId
    );
    assertEq(registeredAddress, validApplicationAddress);
}
```

5.9 Unauthorized Application Contract Registration Attempt

Result	**PASS**
Function	registerApplicationContracts
Objective	To ensure the registerApplicationContracts function reverts when called by a non-owner.
Test Case	

```
function testUnauthorizedRegisterApplicationContracts() public {
    uint16 validChainId = 1; // Example valid chain ID
    bytes32 validApplicationAddress = bytes32("applicationAddrs"); // Example valid application address

    // Impersonate a non-owner address
    address nonOwner = address(0x458a64653f613d932a37E95557af8085b39A2f3F);
    vm.startPrank(nonOwner);

    // Expect the transaction to revert due to unauthorized access

    vm.expectRevert();
    tokenManager.registerApplicationContracts(
        validChainId,
        validApplicationAddress
    );

    // Stop impersonation
    vm.stopPrank();
}
```

Expected Behavior	This test verifies that only the owner can register application contracts, ensuring unauthorized access is prevented.

5.10 Successful Token Manager Contract Registration

Result	PASS
Function	registerApplicationContracts
Objective	To verify that the registerTokenManagerContracts function correctly registers token manager contracts for a given chain ID.
Test Case	
Expected Behavior	This test checks that the function updates the mapping of token manager contracts when called by the owner.

```
function testRegisterTokenManagerContracts() public {
    uint16 validChainId = 1; // Example valid chain ID
    address validTokenManagerAddress = address(0x789); // Example valid token manager address

    // Impersonate the owner address
    vm.startPrank(tokenManager.owner());

    // Call registerTokenManagerContracts
    tokenManager.registerTokenManagerContracts(
        validChainId,
        validTokenManagerAddress
    );

    // Stop impersonation
    vm.stopPrank();

    // Verify the token manager contract was registered correctly
    address registeredAddress = tokenManager._tokenManagerContracts(
        validChainId
    );
    assertEq(registeredAddress, validTokenManagerAddress);
}
```

5.11 Passing Checks

Here is the POC attached that depicts the execution of the test scripts and the corresponding results.

```
Ran 10 tests for test/TokenManagerForkTest.t.sol:TokenManagerForkTest
[PASS] testDepositGasTokenToCodex() (gas: 129035)
[PASS] testDepositToCodex() (gas: 149959)
[PASS] testDepositToCodexWithInsufficientAllowance() (gas: 70254)
[PASS] testDepositToCodexWithInsufficientBalance() (gas: 71983)
[PASS] testDepositToCodexWithInvalidLoginAddress() (gas: 66430)
[PASS] testDepositToCodexWithZeroAmount() (gas: 42897)
[PASS] testQuoteCrossChainDepositWithInvalidChainId() (gas: 42615)
[PASS] testRegisterApplicationContracts() (gas: 41563)
[PASS] testRegisterTokenManagerContracts() (gas: 42579)
[PASS] testUnauthorizedRegisterApplicationContracts() (gas: 16791)
Suite result: ok. 10 passed; 0 failed; 0 skipped; finished in 14.58s (42.85s CPU time)

Ran 1 test suite in 14.59s (14.58s CPU time): 10 tests passed, 0 failed, 0 skipped (10 total tests)
```

6. EVM To Algorand Transactions

6.1 Depositing USDC

Initial Balance of the user 0xf0ba633d8c8b2efea8b988fe3b4ce3951784b44d

```
(venv) gurkiratsingh@Gurkirats-MacBook-Air codex % python3 scripts/run.py
login chain id 10002
login private key 80a92e95c5aea441ce86fe5ece9d6f6db366e80919a8ed1cb1167e908e75087d
login address 0xf0ba633d8c8b2efea8b988fe3b4ce3951784b44d
assest Chain id 65537
assest address 4343545055534443000000000000000000000000000000000000000000000000
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02q\x0
0' 32
userBalance 160000
```

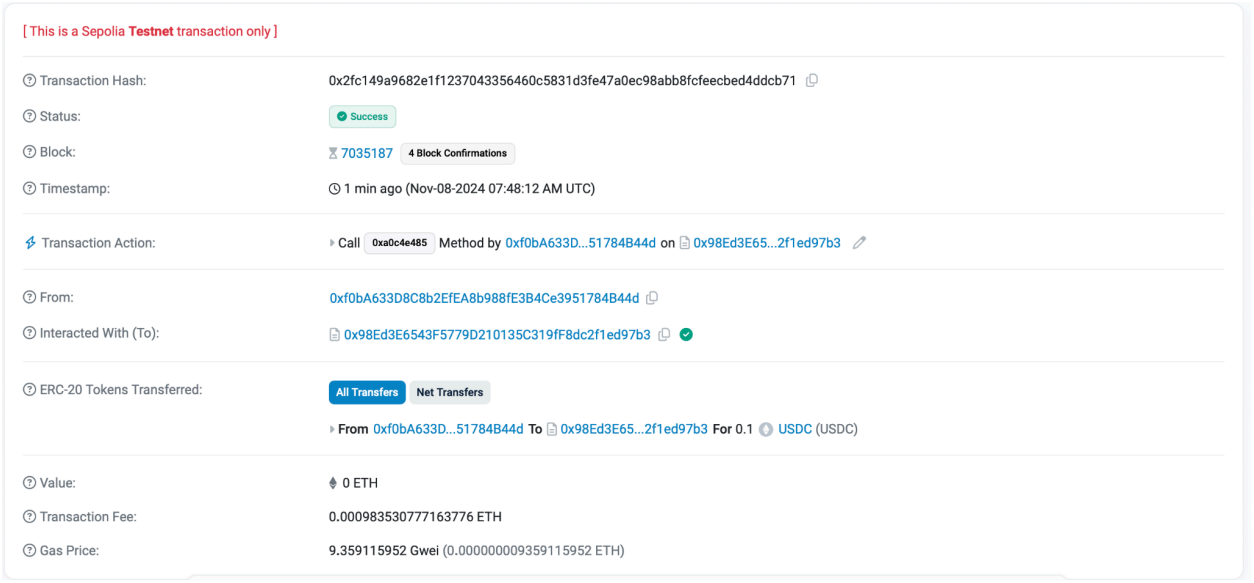
Calling the depositToCodex function

```
gurkiratsingh@Gurkirats-MacBook-Air evm % npm run depositToCodex    a8b988fe3b4ce3951784b44d  Aa  ab  .*  9 of 9

> TokenManager@1.0.0 depositToCodex
> ts-node ./ts-scripts/main.ts --depositToCodex

chain {
  description: 'Ethereum Sepolia',
  chainId: 10002,
  rpc: 'https://ethereum-sepolia-rpc.publicnode.com',
  tokenBridge: '0xDB5492265f6038831E89f495670FF909aDe94bd9',
  wormholeRelayer: '0x7B1bD7a6b4E61c2a123AC6BC2cbfC614437D0470',
  wormhole: '0x4a8bc80Ed5a4067f1CCf107057b8270E0cC11A78',
  cctpTokenMessenger: '0x9f3B8679c73C2Fef8b59B4f3444d4e156fb70AA5',
  cctpMessageTransmitter: '0x7865fAfC2db2093669d92c0F33AeEF291086BEFD',
  USDC: '0x1c7D4B196Cb0C7B01d743Fbc6116a902379C7238',
  WETH: '0xfff9976782d46cc05630d1f6ebab18b2324d6b14',
  testToken: '0x3DdB199aFB250ee004aC0747b2954291fB66E56A'
}
token 0x1c7D4B196Cb0C7B01d743Fbc6116a902379C7238
amount 100000n
_proxy {
  '4': '0x718CaC2Ff57123D4328C6F9e49c3A90126Da8c69',
  '6': '0xC18F0D7153DE49C3841DC32D29F20C384396839d',
  '10002': '0x98Ed3E6543F5779D210135C319fF8dc2f1ed97b3',
  '10003': '0x3FAFc9cd92b57eb3b2C8Abd80aF38a440C2a6d4e',
  '10004': '0x938a9036a984E23f30e2861799358A919Dc77e62',
  '10005': '0xC18F0D7153DE49C3841DC32D29F20C384396839d',
  '10007': '0x99737Ec4B815d816c49A385943baf0380e75c0Ac'
}
currentGasPrice 8040820587
```

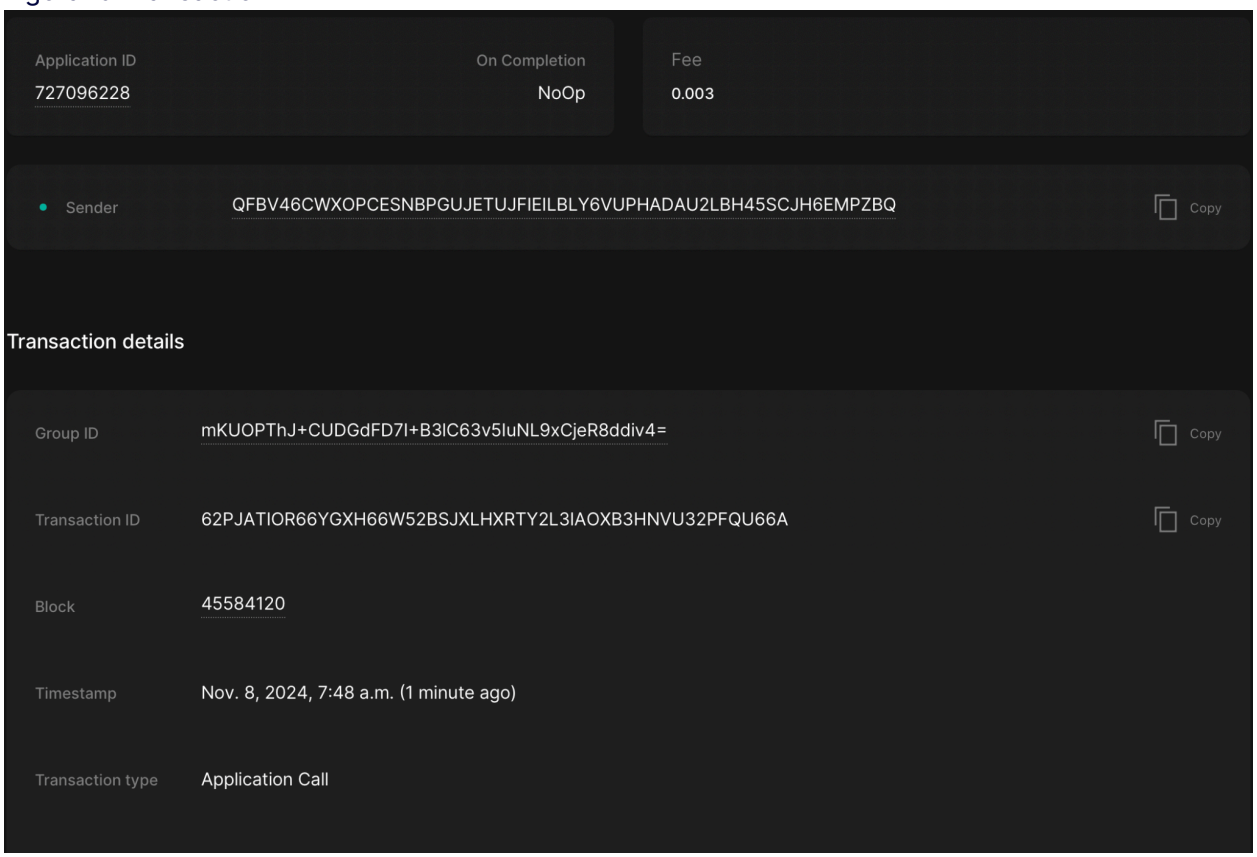
Relayer

```
info: Finished missed VAAs check. Found: 0. Enable debug to see a log with precise results. {"emitterAddress":"00000000000
0000000000001274ad4eda332b482b89ddd9816d85258ed75937","emitterChain":10003}
debug: Finished missed vaas check. Results: {"missedVaas":{"processed":[],"failedToRecover":[],"failedToReprocess":[],"loo
kAheadSequences":[],"seenSequences":[],"missingSequences":[]},"lastSafeSequence":0,"lastSeenSequence":0,"firstSeenSequence
":0}} {"emitterAddress":"0000000000000000000000001274ad4eda332b482b89ddd9816d85258ed75937","emitterChain":10003}
userbalance 260000n
tmcbalance 200000n
debug: Finished VAA processing {"emitterAddress":"00000000000000000000000098ed3e6543f5779d210135c319ff8dc2f1ed97b3","emitt
erChain":10002,"sequence":12}
```

Sepolia Etherscan Transaction



Algorand Transaction



Final Balance Increased

```
(venv) gurkiratsingh@Gurkirats-MacBook-Air codex % python3 scripts/run.py
login chain id 10002
login private key 80a92e95c5aea441ce86fe5ece9d6f6db366e80919a8ed1cb1167e908e75087d
login address 0xf0ba633d8c8b2efea8b988fe3b4ce3951784b44d
assest Chain id 65537
assest address 4343545055534443000000000000000000000000000000000000000000000000
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x03\xf7
\xa0' 32
userBalance 260000
```

6.2 Withdrawing USDC

Initial Balance

```
(venv) gurkiratsingh@Gurkirats-MacBook-Air codex % python3 scripts/run.py
login chain id 10002
login private key 80a92e95c5aea441ce86fe5ece9d6f6db366e80919a8ed1cb1167e908e75087d
login address 0xf0ba633d8c8b2efea8b988fe3b4ce3951784b44d
assest Chain id 65537
assest address 4343545055534443000000000000000000000000000000000000000000000000
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x03\xd0
\x90' 32
userBalance 250000
```

Running withdraw

```
(venv) gurkiratsingh@Gurkirats-MacBook-Air codex % python3 scripts/runWithdraw.py
AFIAAAPOXO7ZN3CUWXIT2FQWD3VJCEPFU2DU6U642AV6XFJRKM54GU64JU
727096157
before algo amount of codex 2000000
codex min balance 212400
before algo amount of msg processor 2000000
10002 0xf0ba633d8c8b2efea8b988fe3b4ce3951784b44d
dest_private_key 80a92e95c5aea441ce86fe5ece9d6f6db366e80919a8ed1cb1167e908e75087d
4343545055534443000000000000000000000000000000000000000000000000 0xf0ba633d8c8b2efea8b988fe3b4ce3951784b44d 10002
emitterId from 727096228 11fcc2bbdbb39ce1d500ac068b234eccf21bb0d8f421cc69df0e63ff68fec08b
sig_addr 86525623 11fcc2bbdbb39ce1d500ac068b234eccf21bb0d8f421cc69df0e63ff68fec08b FOVXINPB2TH3CULASIBM4QTF2CKPLESZXE6WOSWRIQL
KVNGXKQ4OCKMU5Y
login 0xf0ba633d8c8b2efea8b988fe3b4ce3951784b44d 10002
super AFIAAAPOXO7ZN3CUWXIT2FQWD3VJCEPFU2DU6U642AV6XFJRKM54GU64JU 10002
236
10002
b'\x19Ethereum Signed Message:\n550{"Login_Address": "0xf0ba633d8c8b2efea8b988fe3b4ce3951784b44d", "Login_ChainId": "10002", "
Token_Address": "4343545055534443000000000000000000000000000000000000000000000000", "Receiver_Address": "0xf0ba633d8c8b2efea8b
988fe3b4ce3951784b44d", "Receiver_ChainId": "10002", "Amount": "100000", "Withdraw_Fee": "False"}AAAAAAAAAAAAAAAA8LpjPYyLLv6ou
Yj+O0zjlReEtE0AAAAAAAAnEkNDVFBVU0RDAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA8LpjPYyLLv6ouYj+O0zjlReEtE0AAAAAAAAnEgAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAPo'
userBalance 150000
after algo amount of codex 2000000
after algo amount of msg processor 2000000
```

Sepolia etherscan transaction Balance being updating the user's wallet 0xf0ba633d8c8b2efea8b988fe3b4ce3951784b44d

[This is a Sepolia **Testnet** transaction only]

Transaction Hash:	0x2654df0757ea1ea7ededecbbd20492b979d10916622bb84fc7808c996b9f6d7b
Status:	Success
Block:	7035356 2 Block Confirmations
Timestamp:	1 min ago (Nov-08-2024 08:28:24 AM UTC)
Transaction Action:	Call Receive Encoded Msg Function by 0xf0bA633D...51784B44d on 0x98Ed3E65...2f1ed97b3
From:	0xf0bA633D8C8b2EfEA8b988fE3B4Ce3951784B44d
Interacted With (To):	0x98Ed3E6543F5779D210135C319fF8dc2f1ed97b3
ERC-20 Tokens Transferred:	All Transfers Net Transfers From 0x98Ed3E65...2f1ed97b3 To 0xf0bA633D...51784B44d For 0.099 USDC (USDC)
Value:	0 ETH
Transaction Fee:	0.001653543321842712 ETH
Gas Price:	9.598890783 Gwei (0.000000009598890783 ETH)

Algorand explorer Transaction

Application ID	On Completion	Fee
727096157	NoOp	0.007

Sender AFIAAAPOXO7ZN3CUWXIT2FQWD3VJCEPFU2DU6U642AV6XFJRKM54GU64JU Copy

Transaction details

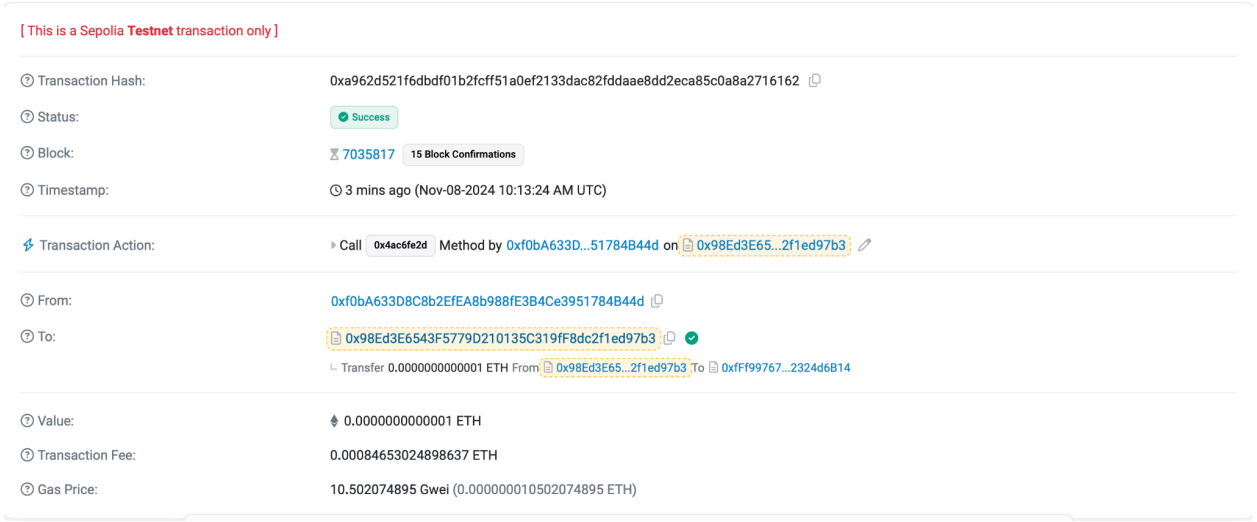
Group ID	jukr/NTXntflvBE4uc1ZmjYX179AGVrxG2SDOKPmPjQ=
Transaction ID	BKN5Z7SBY4WC4YGE45UHFJ3CO5NJL6IK7EQESFN7YCEGZKXARFBQ
Block	45584994
Timestamp	Nov. 8, 2024, 8:28 a.m. (4 minutes ago)
Transaction type	Application Call

Final Balance is decreased.

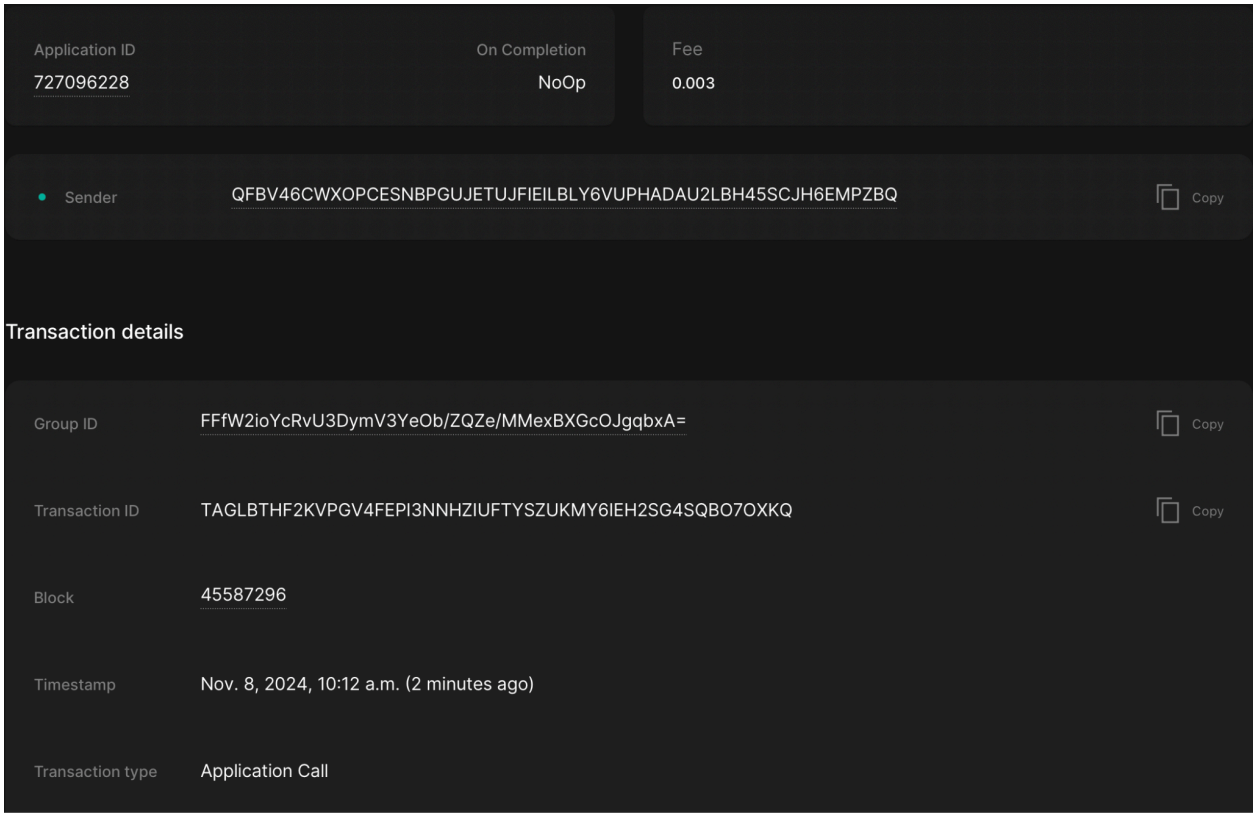
```
(venv) gurkiratsingh@Gurkirats-MacBook-Air codex % python3 scripts/run.py
login chain id 10002
login private key 80a92e95c5aea441ce86fe5ece9d6f6db366e80919a8ed1cb1167e908e75087d
login address 0xf0ba633d8c8b2efea8b988fe3b4ce3951784b44d
assest Chain id 65537
assest address 4343545055534443000000000000000000000000000000000000000000000000
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02I\xf0' 32
userBalance 150000
```

6.3 Deposit Gas Token To Codex

Sepolia Transaction



Algorand Transaction



6.4 Withdrawing zero amount

```python
if __name__ == "__main__":
    tokenType = TokenType.USDC
    login_chain = SupportedChains.Etherem
    token_chain = SupportedChains.Etherem
    withdrawByEnv(login_chain, token_chain, int(0), int(1e3), tokenType)
```

The transaction doesn't take place which is correct

```
434354505553444300000000000000000000000000000000000000000000000 0xf0ba633d8c8b2efea8b988fe3b4ce3951784b44d 10002
emitterId from 727096228 11fcc2bbdbb39ce1d500ac068b234eccf21bb0d8f421cc69df0e63ff68fec08b
sig_addr 86525623 11fcc2bbdbb39ce1d500ac068b234eccf21bb0d8f421cc69df0e63ff68fec08b FOVXINPB2TH3CULASIBM4QTF2CKPLESZXE6WOSWRIQL
KVNGXKQ4OCKMU5Y
login 0xf0ba633d8c8b2efea8b988fe3b4ce3951784b44d 10002
super AFIAAAPOXO7ZN3CUWXIT2FQWD3VJCEPFU2DU6U642AV6XFJRKM54GU64JU 10002
236
10002
b'\x19Ethereum Signed Message:\n545{"Login_Address": "0xf0ba633d8c8b2efea8b988fe3b4ce3951784b44d", "Login_ChainId": "10002", "
Token_Address": "434354505553444300000000000000000000000000000000000000000000000", "Receiver_Address": "0xf0ba633d8c8b2efea8b
988fe3b4ce3951784b44d", "Receiver_ChainId": "10002", "Amount": "0", "Withdraw_Fee": "False"}AAAAAAAAAAAAAAA8LpjPYyLLv6ouYj+O0
zjlReEtE0AAAAAAAAnEkNDVFBVU0RDAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA8LpjPYyLLv6ouYj+O0zjlReEtE0AAAAAAAAnEgAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAPo'
Traceback (most recent call last):
```

6.5 Withdrawing with zero fee amount

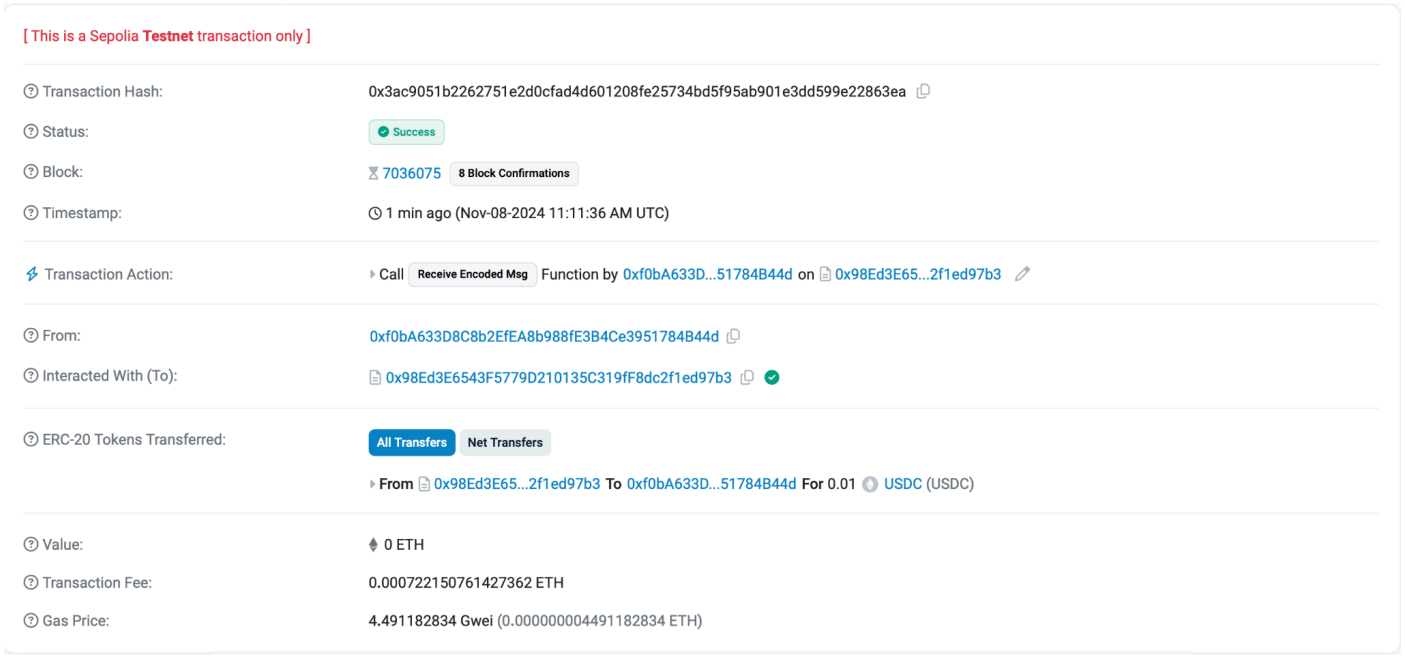
Initial Balance

```
(venv) gurkiratsingh@Gurkirats-MacBook-Air codex % python3 scripts/run.py
login chain id 10002
login private key 80a92e95c5aea441ce86fe5ece9d6f6db366e80919a8ed1cb1167e908e75087d
login address 0xf0ba633d8c8b2efea8b988fe3b4ce3951784b44d
assest Chain id 65537
assest address 434354505553444300000000000000000000000000000000000000000000000
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\xd4
\xc0' 32
userBalance 120000
```

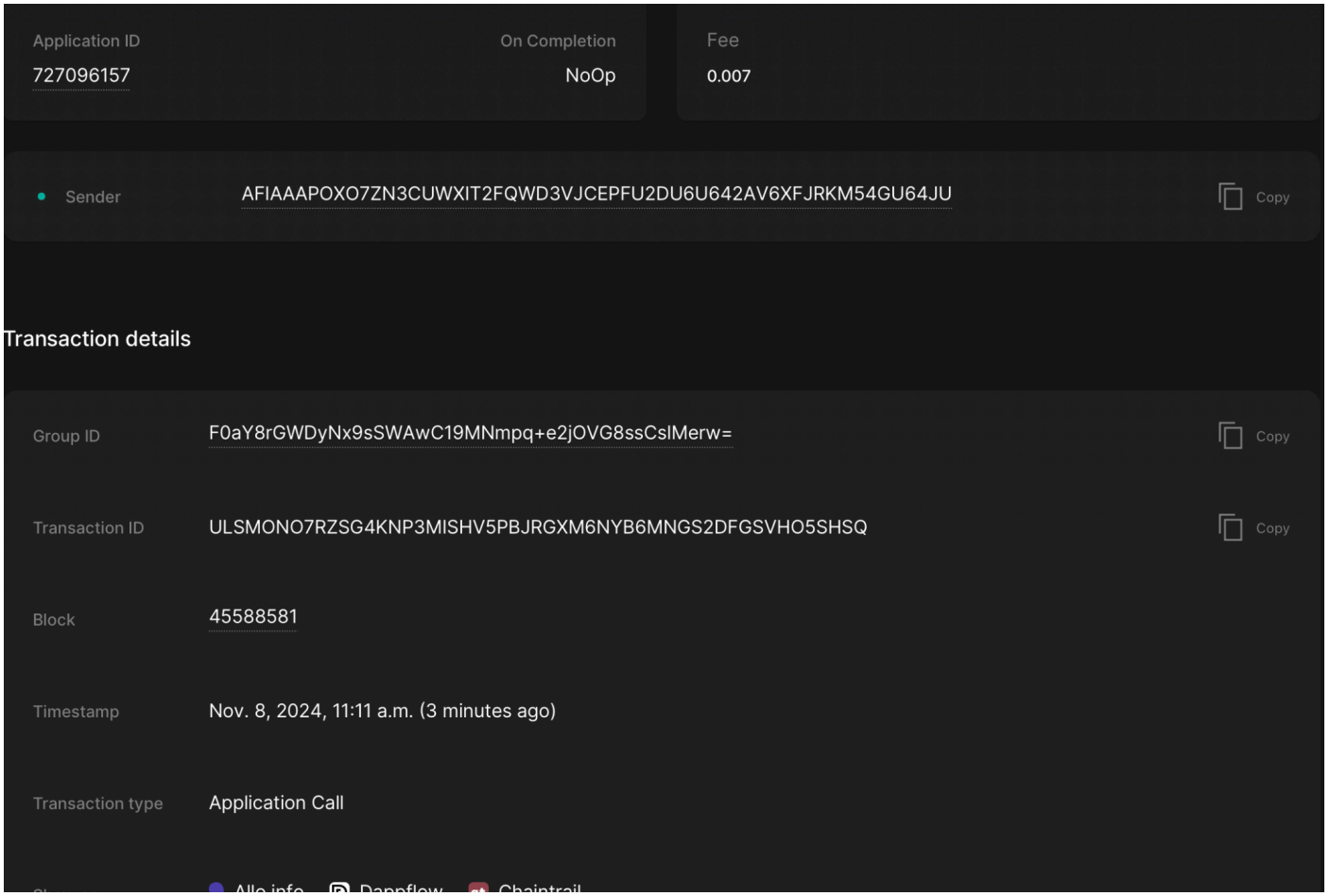
Running Withdraw

```
(venv) gurkiratsingh@Gurkirats-MacBook-Air codex % python3 scripts/runWithdraw.py
AFIAAAPOXO7ZN3CUWXIT2FQWD3VJCEPFU2DU6U642AV6XFJRKM54GU64JU
727096157
before algo amount of codex 2000000
codex min balance 240500
before algo amount of msg processor 2000000
10002 0xf0ba633d8c8b2efea8b988fe3b4ce3951784b44d
dest_private_key 80a92e95c5aea441ce86fe5ece9d6f6db366e80919a8ed1cb1167e908e75087d
434354505553444300000000000000000000000000000000000000000000000 0xf0ba633d8c8b2efea8b988fe3b4ce3951784b44d 10002
emitterId from 727096228 11fcc2bbdbb39ce1d500ac068b234eccf21bb0d8f421cc69df0e63ff68fec08b
sig_addr 86525623 11fcc2bbdbb39ce1d500ac068b234eccf21bb0d8f421cc69df0e63ff68fec08b FOVXINPB2TH3CULASIBM4QTF2CKPLESZXE6WOSWRIQL
KVNGXKQ4OCKMU5Y
login 0xf0ba633d8c8b2efea8b988fe3b4ce3951784b44d 10002
super AFIAAAPOXO7ZN3CUWXIT2FQWD3VJCEPFU2DU6U642AV6XFJRKM54GU64JU 10002
236
10002
b'\x19Ethereum Signed Message:\n549{"Login_Address": "0xf0ba633d8c8b2efea8b988fe3b4ce3951784b44d", "Login_ChainId": "10002", "
Token_Address": "434354505553444300000000000000000000000000000000000000000000000", "Receiver_Address": "0xf0ba633d8c8b2efea8b
988fe3b4ce3951784b44d", "Receiver_ChainId": "10002", "Amount": "10000", "Withdraw_Fee": "False"}AAAAAAAAAAAAAAA8LpjPYyLLv6ouY
j+O0zjlReEtE0AAAAAAAAnEkNDVFBVU0RDAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA8LpjPYyLLv6ouYj+O0zjlReEtE0AAAAAAAAnEgAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACcQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
userBalance 110000
after algo amount of codex 2000000
after algo amount of msg processor 2000000
```

Sepolia Transaction



Algorand Transaction



Final Balance decreased

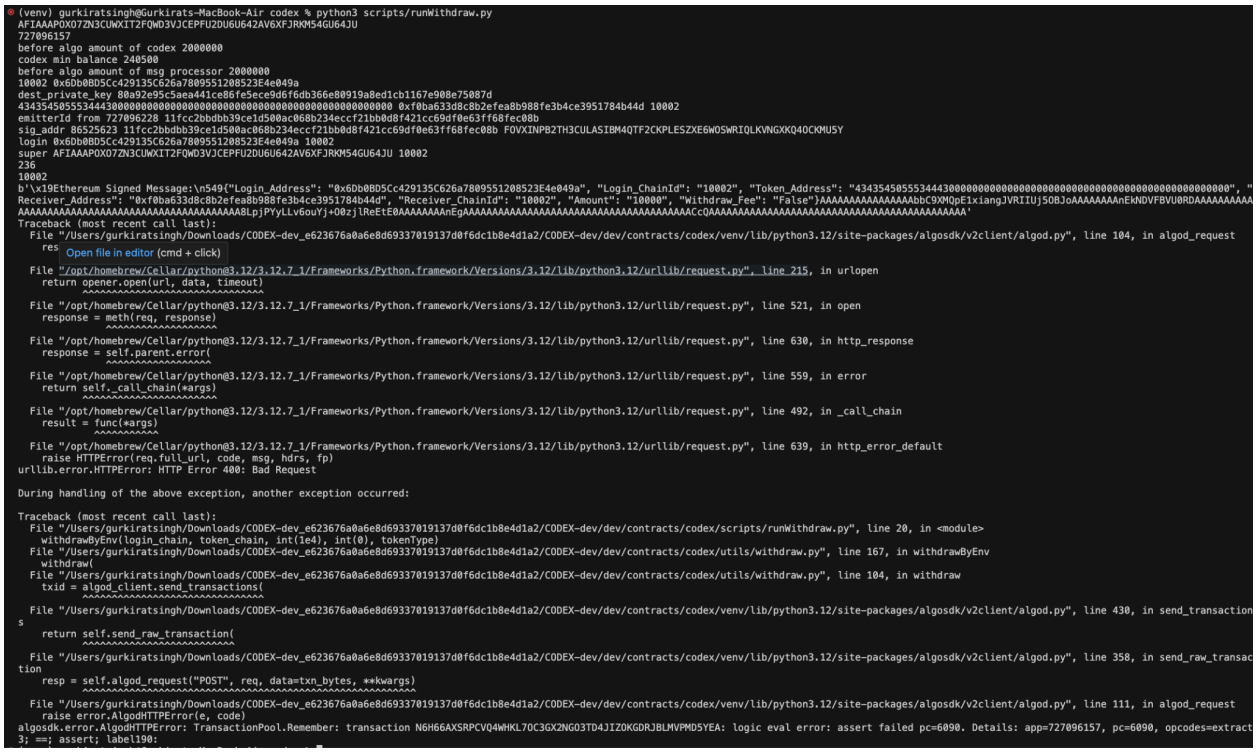
```
(venv) gurkiratsingh@Gurkirats-MacBook-Air codex % python3 scripts/run.py
login chain id 10002
login private key 80a92e95c5aea441ce86fe5ece9d6f6db366e80919a8ed1cb1167e908e75087d
login address 0xf0ba633d8c8b2efea8b988fe3b4ce3951784b44d
assest Chain id 65537
assest address 434354505553444300000000000000000000000000000000000000000000000000
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\xad
\xb0' 32
userBalance 110000
```

6.6 Withdrawing with the user that has not deposited any USDC

Initial Balance of the user 0x6Db0BD5Cc429135C626a7809551208523E4e049a Balance was not found, Which is the correct expected behavior..

```
(venv) gurkiratsingh@Gurkirats-MacBook-Air codex % python3 scripts/run.py
login chain id 10002
login private key 80a92e95c5aea441ce86fe5ece9d6f6db366e80919a8ed1cb1167e908e75087d
login address 0x6Db0BD5Cc429135C626a7809551208523E4e049a
assest Chain id 65537
assest address 434354505553444300000000000000000000000000000000000000000000000000
Traceback (most recent call last):
  File "/Users/gurkiratsingh/Downloads/CODEX-dev_e623676a0a6e8d69337019137d0f6dc1b8e4d1a2/CODEX-dev/dev/contracts/codex/venv/lib/python3.12/site-packages/algosdk/v2c
nt/algod.py", line 104, in algod_request
    resp = urlopen(req)
           ^^^^^^^^^^^^
  File "/opt/homebrew/Cellar/python@3.12/3.12.7_1/Frameworks/Python.framework/Versions/3.12/lib/python3.12/urllib/request.py", line 215, in urlopen
    return opener.open(url, data, timeout)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/opt/homebrew/Cellar/python@3.12/3.12.7_1/Frameworks/Python.framework/Versions/3.12/lib/python3.12/urllib/request.py", line 521, in open
    response = meth(req, response)
               ^^^^^^^^^^^^^^^^^^
  File "/opt/homebrew/Cellar/python@3.12/3.12.7_1/Frameworks/Python.framework/Versions/3.12/lib/python3.12/urllib/request.py", line 630, in http_response
    response = self.parent.error(
               ^^^^^^^^^^^^^^^^^^
  File "/opt/homebrew/Cellar/python@3.12/3.12.7_1/Frameworks/Python.framework/Versions/3.12/lib/python3.12/urllib/request.py", line 559, in error
    return self._call_chain(*args)
           ^^^^^^^^^^^^^^^^^^^^^^^
  File "/opt/homebrew/Cellar/python@3.12/3.12.7_1/Frameworks/Python.framework/Versions/3.12/lib/python3.12/urllib/request.py", line 492, in _call_chain
    result = func(*args)
             ^^^^^^^^^^^
  File "/opt/homebrew/Cellar/python@3.12/3.12.7_1/Frameworks/Python.framework/Versions/3.12/lib/python3.12/urllib/request.py", line 639, in http_error_default
    raise HTTPError(req.full_url, code, msg, hdrs, fp)
urllib.error.HTTPError: HTTP Error 404: Not Found

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/Users/gurkiratsingh/Downloads/CODEX-dev_e623676a0a6e8d69337019137d0f6dc1b8e4d1a2/CODEX-dev/dev/contracts/codex/scripts/run.py", line 7, in <module>
    readUserBalanceBoxByEnv(SupportedChains.Etherem, SupportedChains.Etherem, TokenType.USDC)
  File "/Users/gurkiratsingh/Downloads/CODEX-dev_e623676a0a6e8d69337019137d0f6dc1b8e4d1a2/CODEX-dev/dev/contracts/codex/utils/box.py", line 71, in readUserBalanceBoxI
nv
    return readUserBalanceBox(algod_client, codex_id, loginAddress, loginChainId, assetAddr, assetChainId)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/Users/gurkiratsingh/Downloads/CODEX-dev_e623676a0a6e8d69337019137d0f6dc1b8e4d1a2/CODEX-dev/dev/contracts/codex/utils/box.py", line 35, in readUserBalanceBox
    box = readBox(algod_client, codex_id, boxKey)
          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/Users/gurkiratsingh/Downloads/CODEX-dev_e623676a0a6e8d69337019137d0f6dc1b8e4d1a2/CODEX-dev/dev/contracts/codex/utils/box.py", line 24, in readBox
    return algod_client.application_box_by_name(app_id, boxKey)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/Users/gurkiratsingh/Downloads/CODEX-dev_e623676a0a6e8d69337019137d0f6dc1b8e4d1a2/CODEX-dev/dev/contracts/codex/venv/lib/python3.12/site-packages/algosdk/v2c
nt/algod.py", line 189, in application_box_by_name
    return self.algod_request("GET", req, params=params, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/Users/gurkiratsingh/Downloads/CODEX-dev_e623676a0a6e8d69337019137d0f6dc1b8e4d1a2/CODEX-dev/dev/contracts/codex/venv/lib/python3.12/site-packages/algosdk/v2c
nt/algod.py", line 111, in algod_request
    raise error.AlgodHTTPError(e, code)
algosdk.error.AlgodHTTPError: box not found
```

The run withdraw function did not go through which is correct.



6.7 Deposit to codex function with zero amount

```
async function depositToCodex() {
  // const recipient = getArg(["--recipient", "-r"]) || "";
  const chainId = utils.getCurrentChainId();
  const chain = utils.getChain(chainId);
  const token = utils.getTestTokenAddress(chainId);
  const amount = utils.getDepositAmount();

  console.log("chain", chain);
  console.log("token", token);
  console.log("amount", amount);

  const loginChainId = chainId

  // const loginAddr = "0x6f7ccF76681dA80cBD7AD4b28697A2881A70286d";
  const loginAddr = "0xf0ba633d8c8b2efea8b988fe3b4ce3951784b44d";


  const tokenManager = utils.getTokenManager(chainId)
  const normalizedAddr = utils.getNormalizedAddress(loginAddr);

  // console.log(normalizedAddr,"this is normalizzer address")

  const currentGasPrice = await utils.getGasPrice(chainId);
  console.log("currentGasPrice", currentGasPrice.toNumber());

  const rx = await tokenManager
    .depositToCodex(
      token,
      0,
      utils.hexToBytes(normalizedAddr.substring(2)),
      loginChainId,
      // {gasLimit: 500000}
    )
    .then(utils.wait)
}
```

Getting error while depositing which is correct

```
gurkiratsingh@Gurkirats-MacBook-Air evm % npm run depositToCodex

> TokenManager@1.0.0 depositToCodex
> ts-node ./ts-scripts/main.ts --depositToCodex

chain {
  description: 'Ethereum Sepolia',
  chainId: 10002,
  rpc: 'https://ethereum-sepolia-rpc.publicnode.com',
  tokenBridge: '0xDB5492265f6038831E89f495670FF909aDe94bd9',
  wormholeRelayer: '0x7B1bD7a6b4E61c2a123AC6BC2cbfC614437D0470',
  wormhole: '0x4a8bc80Ed5a4067f1CCf107057b8270E0cC11A78',
  cctpTokenMessenger: '0x9f3B8679c73C2Fef8b59B4f3444d4e156fb70AA5',
  cctpMessageTransmitter: '0x7865fAfC2db2093669d92c0F33AeEF291086BEFD',
  USDC: '0x1c7D4B196Cb0C7B01d743Fbc6116a902379C7238',
  WETH: '0xfff9976782d46cc05630d1f6ebab18b2324d6b14',
  testToken: '0x3DdB199aFB250ee004aC0747b2954291fB66E56A'
}
token 0x1c7D4B196Cb0C7B01d743Fbc6116a902379C7238
amount 100000n
_proxy {
  '4': '0x718CaC2Ff57123D4328C6F9e49c3A90126Da8c69',
  '6': '0xC18F0D7153DE49C3841DC32D29F20C384396839d',
  '10002': '0x98Ed3E6543F5779D210135C319fF8dc2f1ed97b3',
  '10003': '0x3FAFc9cd92b57eb3b2C8Abd80aF38a440C2a6d4e',
  '10004': '0x938a9036a984E23f30e2861799358A919Dc77e62',
  '10005': '0xC18F0D7153DE49C3841DC32D29F20C384396839d',
  '10007': '0x99737Ec4B815d816c49A385943baf0380e75c0Ac'
}
currentGasPrice 3460357825
<ref *1> Error: cannot estimate gas; transaction may fail or may require manual gas limit [ See: https://links.ethers.org/v5-errors-UNPREDICTABLE_GAS_LIMIT ] (erro
r={"reason":"execution reverted: You need to sell at least some tokens","code":"UNPREDICTABLE_GAS_LIMIT","method":"estimateGas","transaction":{"from":"0xf0bA633D8C
8b2EfEA8b988fE3B4Ce3951784B44d","maxPriorityFeePerGas":{"type":"BigNumber","hex":"0x59682f00"},"maxFeePerGas":{"type":"BigNumber","hex":"0x01f58f1a4a"},"to":"0x98E
d3E6543F5779D210135C319fF8dc2f1ed97b3","data":"0xa0c4e4850000000000000000000000001c7d4b196cb0c7b01d743fbc6116a902379c7238000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000f0ba633d8c8b2efea8b988fe3b4ce3951784b44d0000000000000000000000000000000000000000000000000000000000002712","type":2,"a
ccessList":null},"error":{"reason":"processing response error","code":"SERVER_ERROR","body":"{\"jsonrpc\":\"2.0\",\"id\":49,\"error\":{\"code\":3,\"message\":\"exe
cution reverted: You need to sell at least some tokens\",\"data\":\"0x08c379a000000000000000000000000000000000000000000000000000000000000000020000000000000000000000
0000000000000000000000000000000000000000025596f75206e65656420746f2073656c6c206174206c6561737420736f6d6520746f6b656e730000000000000000000000000000000000000000000000
00000000\"}}\n","error":{"code":3,"data":"0x08c379a0000000000000000000000000000000000000000000000000000000000000002000000000000000000000000000000000000000000000000
00000000000000025596f75206e65656420746f2073656c6c206174206c6561737420736f6d6520746f6b656e73000000000000000000000000000000000000000000000000000000"},"requestBody":"
{\"method\":\"eth_estimateGas\",\"params\":[{\"type\":\"0x2\",\"maxFeePerGas\":\"0x1f58f1a4a\",\"maxPriorityFeePerGas\":\"0x59682f00\",\"from\":\"0xf0ba633d8c8b2ef
ea8b988fe3b4ce3951784b44d\",\"to\":\"0x98ed3e6543f5779d210135c319ff8dc2f1ed97b3\",\"data\":\"0xa0c4e4850000000000000000000000001c7d4b196cb0c7b01d743fbc6116a902379c
72380000000000000000000000000000000000000000000000000000000000000000000000000000000000000000f0ba633d8c8b2efea8b988fe3b4ce3951784b44d0000000000000000000000000000000
00000000000000000000000000002712\"}],\"id\":49,\"jsonrpc\":\"2.0\"}","requestMethod":"POST","url":"https://ethereum-sepolia-rpc.publicnode.com"}}, tx={"data":"0xa
0c4e4850000000000000000000000001c7d4b196cb0c7b01d743fbc6116a902379c72380000000000000000000000000000000000000000000000000000000000000000000000000000000000000000f0ba
633d8c8b2efea8b988fe3b4ce3951784b44d0000000000000000000000000000000000000000000000000000000000002712","to":{},"from":"0xf0bA633D8C8b2EfEA8b988fE3B4Ce3951784B44d","
type":2,"maxFeePerGas":{"type":"BigNumber","hex":"0x01f58f1a4a"},"maxPriorityFeePerGas":{"type":"BigNumber","hex":"0x59682f00"},"nonce":{},"gasLimit":{},"chainId":
{}}, code=UNPREDICTABLE_GAS_LIMIT, version=abstract-signer/5.7.0)
    at Logger.makeError (/Users/gurkiratsingh/Downloads/CODEX-dev_e623676a0a6e8d69337019137d0f6dc1b8e4d1a2/CODEX-dev/dev/contracts/chains/evm/node_modules/@ethersp
roject/logger/src.ts/index.ts:269:28)
    at Logger.throwError (/Users/gurkiratsingh/Downloads/CODEX-dev_e623676a0a6e8d69337019137d0f6dc1b8e4d1a2/CODEX-dev/dev/contracts/chains/evm/node_modules/@ethers
project/logger/src.ts/index.ts:281:20)
    at /Users/gurkiratsingh/Downloads/CODEX-dev_e623676a0a6e8d69337019137d0f6dc1b8e4d1a2/CODEX-dev/dev/contracts/chains/evm/node_modules/@ethersproject/abstract-si
gner/src.ts/index.ts:301:31
    at processTicksAndRejections (node:internal/process/task_queues:95:5)
    at async Promise.all (index 7) {
  reason: 'cannot estimate gas; transaction may fail or may require manual gas limit',
  code: 'UNPREDICTABLE_GAS_LIMIT',
  error: Error: cannot estimate gas; transaction may fail or may require manual gas limit [ See: https://links.ethers.org/v5-errors-UNPREDICTABLE_GAS_LIMIT ] (reas
on="execution reverted: You need to sell at least some tokens", method="estimateGas", transaction={"from":"0xf0bA633D8C8b2EfEA8b988fE3B4Ce3951784B44d","maxPriority
FeePerGas":{"type":"BigNumber","hex":"0x59682f00"},"maxFeePerGas":{"type":"BigNumber","hex":"0x01f58f1a4a"},"to":"0x98Ed3E6543F5779D210135C319fF8dc2f1ed97b3","data
":"0xa0c4e4850000000000000000000000001c7d4b196cb0c7b01d743fbc6116a902379c723800000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

7. Auditing Approach and Methodologies Applied

The solidity smart contract was audited in a comprehensive approach to ensure the highest level of security and reliability. Careful attention was given to the following key areas to ensure the overall quality of code:

- **Code quality and structure**: We conducted a detailed review of the codebase to identify any potential issues related to code structure, readability, and maintainability. This included analyzing the overall architecture of the solidity smart contract and reviewing the code to ensure it follows best practices and coding standards.
- **Security vulnerabilities**: Our team used manual techniques to identify any potential security vulnerabilities that could be exploited by attackers. This involved a thorough analysis of the code to identify any potential weaknesses, such as buffer overflows, injection vulnerabilities, Signatures, and deprecated functions.
- **Documentation and comments**: Our team reviewed the code documentation and comments to ensure they accurately describe the code's intended behavior and logic. This helps developers to better understand the codebase and make modifications without introducing new issues.
- **Compliance with best practices**: We checked that the code follows best practices and coding standards that are recommended by the solidity community and industry experts. This ensures that the solidity smart contract is secure, reliable, and efficient.

Our audit team followed OWASP and Ethereum(Solidity) community security guidelines for this audit. As a result, we were able to identify potential issues and provide recommendations to improve the smart contract security and performance.

Throughout the audit of the smart contract, our team placed great emphasis on ensuring the overall quality of the code and the use of industry best practices. We meticulously reviewed the codebase to ensure that it was thoroughly documented and that all comments and logic aligned with the intended behavior. Our approach to the audit was comprehensive, methodical, and aimed at ensuring that the smart contract was secure, reliable, and optimized for performance.

8. Limitations on Disclosure and Use of this Report

This report contains information concerning potential details of the Ultrade EVM Project and methods for exploiting them. Entersoft recommends that special precautions be taken to protect the confidentiality of both this document and the information contained herein. Security Assessment is an uncertain process, based on past experiences, currently available information, and known threats. All information security systems, which by their nature are dependent on human beings, are vulnerable to some degree. Therefore, while Entersoft considers the major security vulnerabilities of the analyzed systems to have been identified, there can be no assurance that any exercise of this nature will identify all possible vulnerabilities or propose exhaustive and operationally viable recommendations to mitigate those exposures. In addition, the analysis set forth herein is based on the technologies and known threats as of the date of this report. As technologies and risks change over time, the vulnerabilities associated with the operation of the Ultrade EVM solidity smart contract described in this report, as well as the actions necessary to reduce the exposure to such vulnerabilities will also change. Entersoft makes no undertaking to supplement or update this report based on changed circumstances or facts of which Entersoft becomes aware after the date hereof, absent a specific written agreement to perform the supplemental or updated analysis. This report may recommend that Entersoft use certain software or hardware products manufactured or maintained by other vendors. Entersoft bases these recommendations upon its prior experience with the capabilities of those products. Nonetheless, Entersoft does not and cannot warrant that a particular product will work as advertised by the vendor, nor that it will operate in the manner intended. This report was prepared by Entersoft for the exclusive benefit of Ultrade EVM and is proprietary information. The Non-Disclosure Agreement (NDA) in effect between Entersoft and Ultrade EVM governs the disclosure of this report to all other parties including product vendors and suppliers.