SMART CONTRACT AUDIT

ZOKYO.

August 8th 2022 | v. 2.0

PASS

Zokyo Security has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

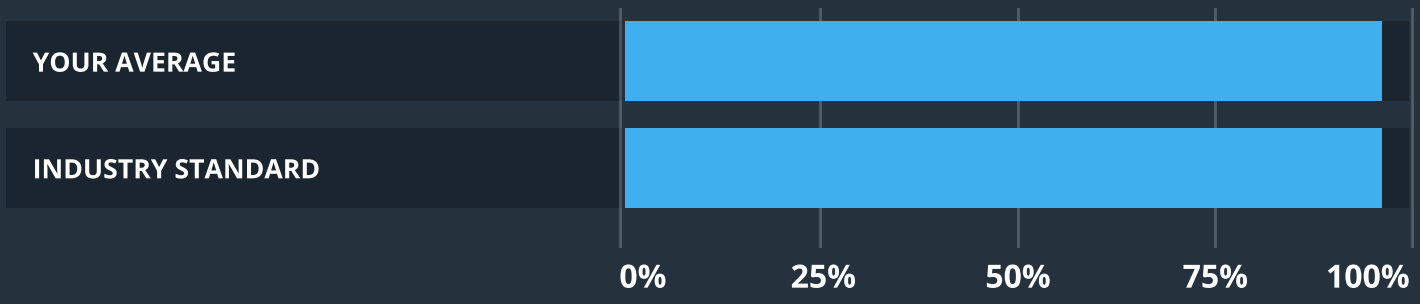
TECHNICAL SUMMARY

This document outlines the overall security of the TrustSwap smart contracts evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the TrustSwap smart contract codebase for quality, security, and correctness.

Contract Status

Testable Code



The 95% of the code is testable, which corresponds the standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the TrustSwap team put in place a bug bounty program to encourage further active analysis of the smart contract.

TABLE OF CONTENTS

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the TrustSwap repository.
https://github.com/trustswap

First audit revision.
https://github.com/trustswap/team-finance-contracts/tree/master-merge-nft-liq-v2v3-migrate

Initial commit: 6a99baee5ff8de2aa1ce797633f68588080de9c0
Last audited commit: a1e581cfe9e8bd67e25444abd90cc80be3bfed7d

Second audit revision.
https://github.com/trustswap/team-finance-contracts/tree/master-merge-nft-migrate-nftsvg

Initial commit: 40ab9b57eda7aaa68557038d1a32df843ec7e57a
Last audited commit: 40ab9b57eda7aaa68557038d1a32df843ec7e57a

Within the scope of this audit, Zokyo auditors have reviewed the following contract(s):

- LockNFT.sol
- NFTDescriptor.sol
- NFTSVG.sol
- LockToken.sol

AUDITING STRATEGY AND TECHNIQUES APPLIED

Throughout the review process, Zokyo Security ensures that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of resources, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of TrustSwap smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented any issues as they were discovered. A part of this work included writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1	Due diligence in assessing the overall code quality of the codebase.	3	Testing contract logic against common and uncommon attack vectors.
2	Cross-comparison with other, similar smart contracts by industry leaders.	4	Thorough manual review of the codebase, line by line.

EXECUTIVE SUMMARY

During the the first iteration of the audit, the Zokyo Security team has carefully checked both contracts within the scope.
There were two critical issues found during the audit. One them was connected to the possibility of the reentrancy attack as the contract performs several external calls to other contracts, including arbitrary passed ERC20 token contracts. The issue was successfully fixed by the TrustSwap team.

The second critical issue was related to the arbitrary passed ERC20 token address, to which lockToken.sol performs external calls during functions execution. TrustSwap has assured that this is an intended logic and users of the protocol should be able to pass any arbitrary address. Though the issue still remains, it was verified that thanks to nonReentrant protection, no critical vulnerabilities can occur.

Other issues were connected to the outdated Solidity version, gas optimizations, and the necessity of events. All the issues were successfully fixed by the team except for the outdated Solidity version. According to the team, the version can't be increased due to current architecture of the contracts. It should be mentioned that both the contracts are upgradable.

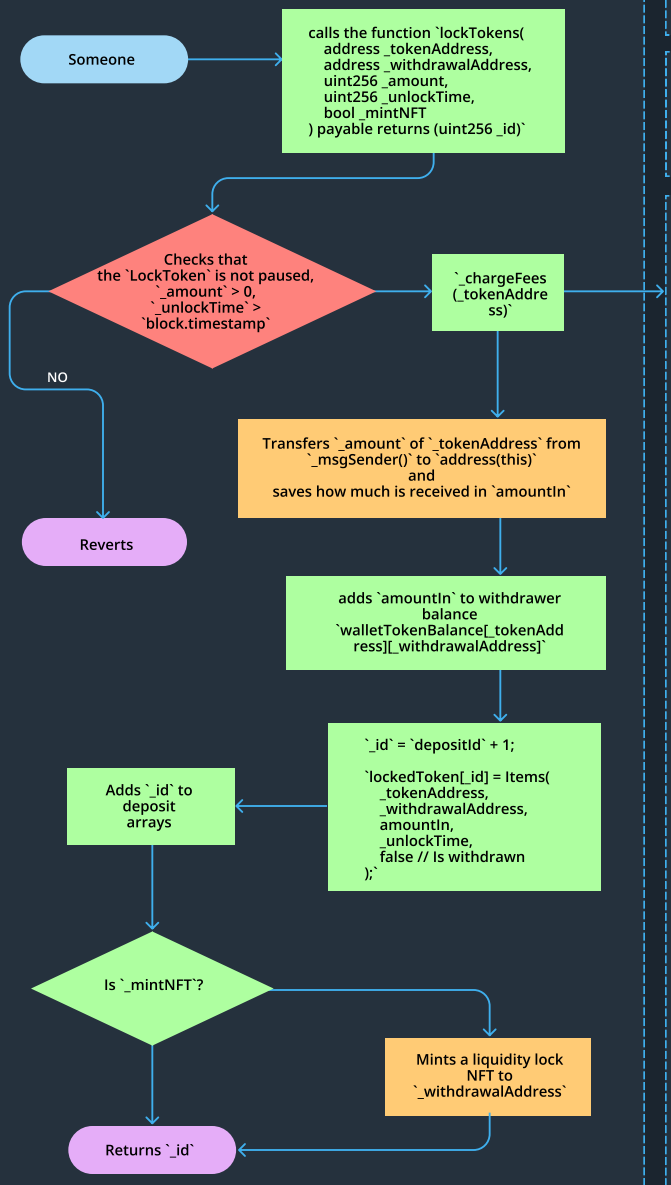
During the second iteration of the audit, a new set of contracts was carefully verified. No critical issues were found within those contracts. The contracts were verified to return correct SVG images for NFTs.

Overall, the security of the contracts is quite high, though they lack documentation. The Zokyo Security team has prepared a set of unit-tests to ensure that the contracts execute as intended.

The total security score is 89. Although the overall security is high, the verified issues such as the outdated Solidity version and the usage of arbitrary ERC20 address still remain, which can lead to unpredictable consequences. Also, the current set of TrustSwap unit test has few failing tests.

lockTokens(
address _tokenAddress,
address _withdrawalAddress,
uint256 _amount,
uint256 _unlockTime,
bool _mintNFT
) public payable whenNotPaused returns (uint256 _id)`

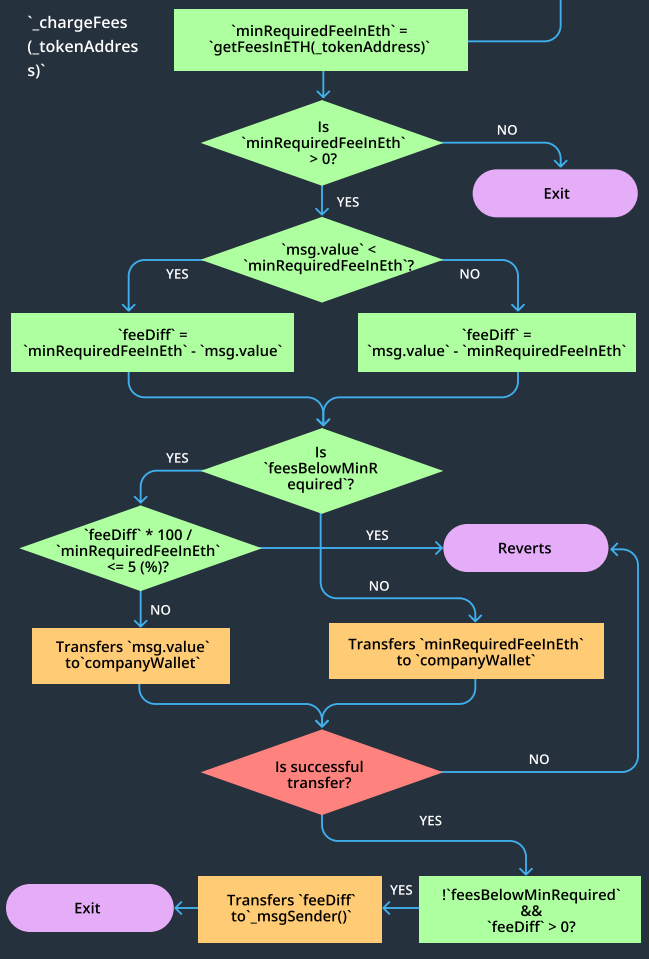
Locks `_tokenAddress` tokens in the amount of `_amount` for withdrawal of them by a `_withdrawalAddress` after an`_unlockTime`, transfers tokens to the `address(this)` from the caller, charges fee, mints a liquidity lock NFT if `_mintNFT` is true for ..., and returns `_id` in the `lockedTokens` mapping.



lockNFTs(
address _tokenAddress,
address _withdrawalAddress,
uint256 _amount,
uint256 _unlockTime,
uint256 _tokenId,
bool _mintNFT
) public payable whenNotPaused returns (uint256 _id)`
Locks a `_tokenId` of a `_tokenAddress` token in the amount of `_amount` for withdrawal of them by a `_withdrawalAddress` after an `_unlockTime`, transfers a`_tokenId` to the `address(this)` from the caller, charges fee, mints a liquidity lock NFT if `_mintNFT` is true for ..., and returns `_id` in the `lockedTokens` mapping. This works similarly to the function `lockTokens(...)`.

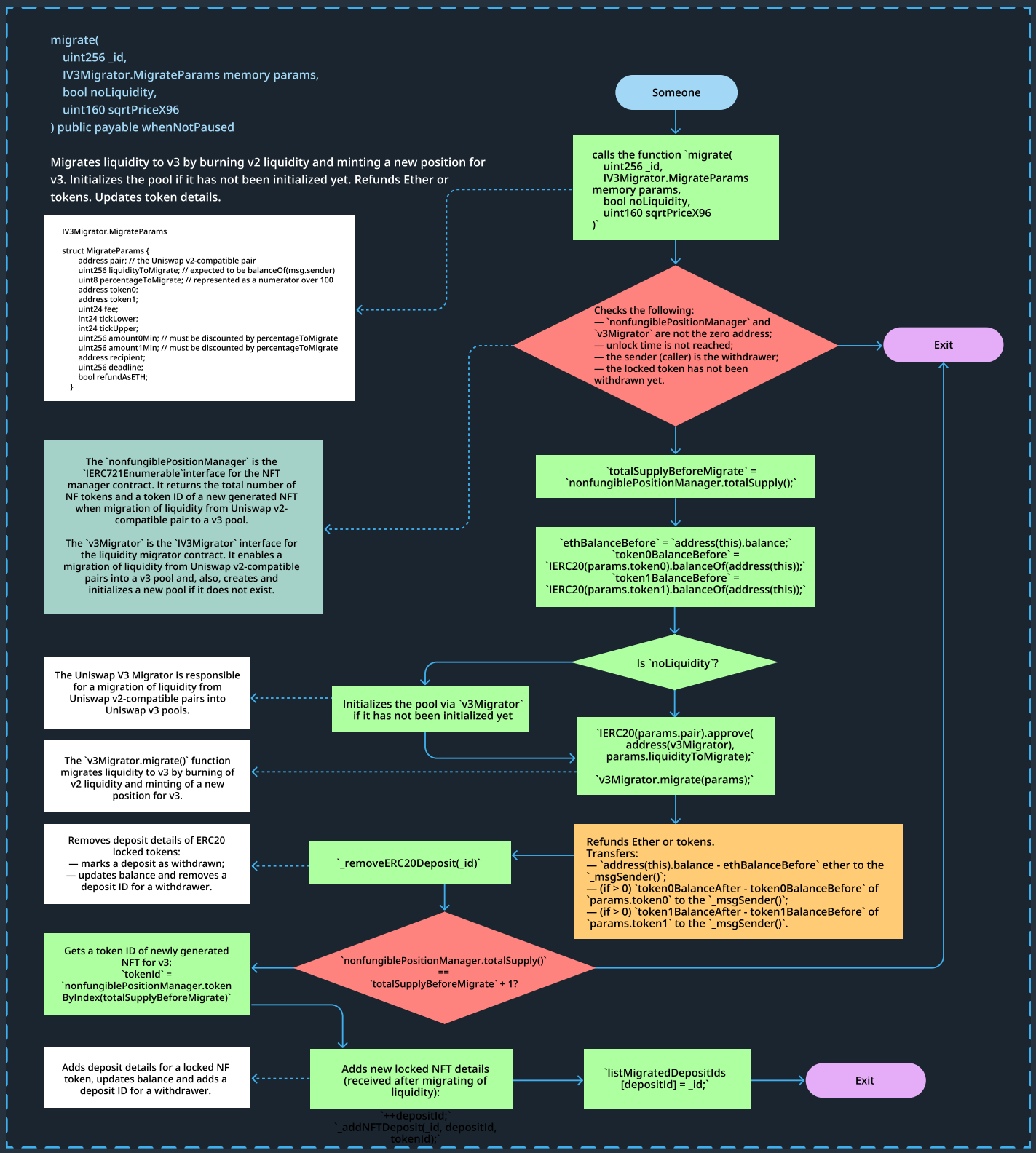
getFeesInETH(_tokenAddress) public view returns (uint256)

Returns fee amount in Ether for `_tokenAddress` using the Uniswap PriceEstimator interface and considering the mapping of free tokens (`listFreeTokens`).



withdrawTokens(uint256 _id) public

Allows `_msgSender()` to withdraw ERC20 tokens or a ERC721 token with the ID `_id` after unlock time. There is the lockTokens(...) and lockNFT(...) functions to lock the tokens for a withdrawer.


migrate(
uint256 _id,
IV3Migrator.MigrateParams memory params,
bool noLiquidity,
uint160 sqrtPriceX96
) public payable whenNotPaused
Migrates liquidity to v3 by burning v2 liquidity and minting a new position for v3. Initializes the pool if it has not been initialized yet. Refunds Ether or tokens. Updates token details.
IV3Migrator.MigrateParams
struct MigrateParams {
address pair; // the Uniswap v2-compatible pair
uint256 liquidityToMigrate; // expected to be balanceOf(msg.sender)
uint8 percentageToMigrate; // represented as a numerator over 100
address token0;
address token1;
uint24 fee;
int24 tickLower;
int24 tickUpper;
uint256 amount0Min; // must be discounted by percentageToMigrate
uint256 amount1Min; // must be discounted by percentageToMigrate
address recipient;
uint256 deadline;
bool refundAsETH;
}
Someone
calls the function `migrate( uint256 _id, IV3Migrator.MigrateParams memory params, bool noLiquidity, uint160 sqrtPriceX96 )`
Checks the following:
— `nonfungiblePositionManager` and `v3Migrator` are not the zero address;
— unlock time is not reached;
— the sender (caller) is the withdrawer;
— the locked token has not been withdrawn yet.
Exit
The `nonfungiblePositionManager` is the `IERC721Enumerable`interface for the NFT manager contract. It returns the total number of NF tokens and a token ID of a new generated NFT when migration of liquidity from Uniswap v2-compatible pair to a v3 pool.
The `v3Migrator` is the `IV3Migrator` interface for the liquidity migrator contract. It enables a migration of liquidity from Uniswap v2-compatible pairs into a v3 pool and, also, creates and initializes a new pool if it does not exist.
`totalSupplyBeforeMigrate` = `nonfungiblePositionManager.totalSupply();`
`ethBalanceBefore` = `address(this).balance;` `token0BalanceBefore` = `IERC20(params.token0).balanceOf(address(this));` `token1BalanceBefore` = `IERC20(params.token1).balanceOf(address(this));`
Is `noLiquidity`?
The Uniswap V3 Migrator is responsible for a migration of liquidity from Uniswap v2-compatible pairs into Uniswap v3 pools.
Initializes the pool via `v3Migrator` if it has not been initialized yet
The `v3Migrator.migrate()` function migrates liquidity to v3 by burning of v2 liquidity and minting of a new position for v3.
`IERC20(params.pair).approve( address(v3Migrator), params.liquidityToMigrate);`
`v3Migrator.migrate(params);`
Removes deposit details of ERC20 locked tokens:
— marks a deposit as withdrawn;
— updates balance and removes a deposit ID for a withdrawer.
`_removeERC20Deposit(_id)`
Refunds Ether or tokens.
Transfers:
— `address(this).balance - ethBalanceBefore` ether to the `_msgSender()`;
— (if > 0) `token0BalanceAfter - token0BalanceBefore` of `params.token0` to the `_msgSender()`;
— (if > 0) `token1BalanceAfter - token1BalanceBefore` of `params.token1` to the `_msgSender()`.
Gets a token ID of newly generated NFT for v3:
`tokenId` = `nonfungiblePositionManager.tokenByIndex(totalSupplyBeforeMigrate)`
`nonfungiblePositionManager.totalSupply()` == `totalSupplyBeforeMigrate` + 1?
Adds deposit details for a locked NF token, updates balance and adds a deposit ID for a withdrawer.
Adds new locked NFT details (received after migrating of liquidity):
`++depositId;` `_addNFTDeposit(_id, depositId, tokenId);`
`listMigratedDepositIds [depositId] = _id;`
Exit

Descriptions of auxiliary functions

`receive() external payable`
returns the event about independent transfer of ether to `address(this).

`extendLockDuration(uint256 _id, uint256 _unlockTime) public`
extends an unlock time for tokens to withdrawal.

`mintNFTforLock(uint256 _id) public whenNotPaused`
can mint a liquidity lock NFT for `_id` by token withdrawer if this was not done at the time of calling function `lockTokens(...)` or `lockNFTs(...)`.

The functions `addTokenToFreeList(address token) external onlyOwner onlyContract(token)` and `removeTokenFromFreeList(address token) external onlyOwner onlyContract(token)` adds/removes a token (`token`) to/from the mapping of free tokens (`listFreeTokens`).

`pause() external onlyOwner` and `unpause() external onlyOwner` for OpenZeppelin pause functionality.

`transferOwnershipNFTContract(address _newOwner) public onlyOwner` calls the OpenZeppelin `transferOwnership(address)` function for the `NFT` contract.

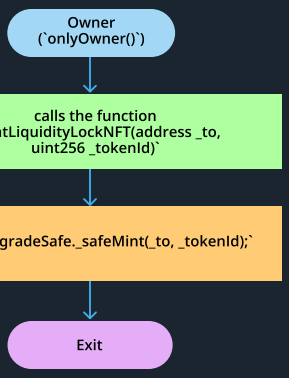
Besides, there are some setters:
— `setFeeParams(
address _priceEstimator,
address _usdTokenAddress,
uint256 _feesInUSD,
address payable _companyWallet
)
external
onlyOwner
onlyContract(_priceEstimator)
onlyContract(_usdTokenAddress)`;
— `setFeesInUSD(uint256 _feesInUSD) external onlyOwner`;
— `setFeesInUSD(uint256 _feesInUSD) external onlyOwner`;
— `setCompanyWallet(address payable _companyWallet) external onlyOwner`;
— `setNonFungiblePositionManager(address _nonfungiblePositionManager)
external
onlyOwner
onlyContract(_nonfungiblePositionManager)`;
— `setV3Migrator(address _v3Migrator)
external
onlyOwner
onlyContract(_v3Migrator)`;
— `setNFTContract(address _nftContractAddress)
public
onlyOwner
onlyContract(_nftContractAddress)`.

Also, there are some getters for the storage variables.

LockNFT.sol is designed for locking of liquidity for the `LockToken` contract.

It is standard OpenZeppelin `ERC721UpgradeSafe` contract with a few features:
— it mints NFTs which are liquidity lock NFTs;
— it transfers locked tokens on the `lockToken` contract (`LockToken.sol`) from a current withdrawer to a new one when transferring of a token.

mintLiquidityLockNFT(address _to, uint256 _tokenId) external onlyOwner



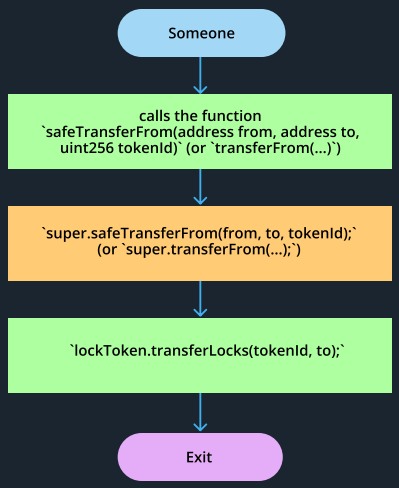
Auxiliary functions

`setLockTokenAddress(address _newLockTokenAddress)
public
onlyOwner
onlyContract(_newLockTokenAddress)`

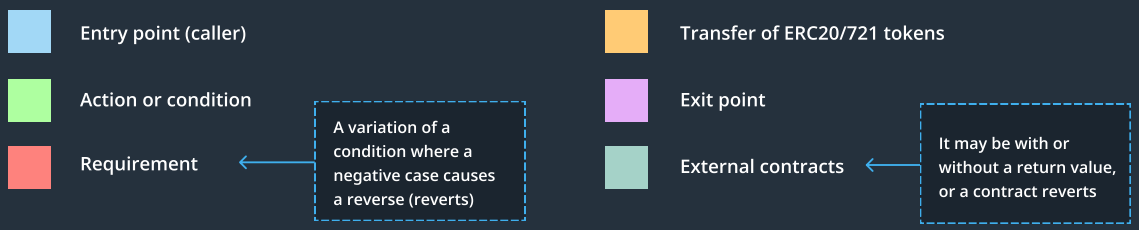
`burn(uint256 tokenId) public onlyOwner`

safeTransferFrom(
address from,
address to,
uint256 tokenId
) public override

(and transferFrom(...))



Color map



STRUCTURE AND ORGANIZATION OF DOCUMENT

For easier navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Customer’s side or is an issue that the Customer disregards as a problem. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that it can lead to a significant loss, funds may be lost or allocated incorrectly.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

CRITICAL | VERIFIED

Use of arbitrary token addresses.

LockToken.sol: functions lockTokens(), lockNFTs(), migrate().
These functions are public, and any token addresses are passed in their parameter lists. This creates a vulnerability for the use of malicious tokens.

Recommendation:
Add token whitelisting to validate any token addresses provided by users.

Post audit:
TrustSwap team verified that this is an intended logic and users should be able to use arbitrary token addresses. Yet, usage of nonReentrant modifier in all functions with token address should be enough to mitigate any critical vulnerabilities.

CRITICAL | RESOLVED

Reentrancy in main functions.

LockToken.sol:

1. withdrawTokens(). External calls of .safeTransferFrom() on line 345, .transfer() on line 364, .burn() on lines 349 and 367 before writing of the nftMinted[_id] state variable with the false value on lines 350 and 368 respectively.
2. migrate(). External calls of .createAndInitializePoolIfNecessary(), v3Migrator.migrate(), .transfer(), and so on before writing of the state variables with _removeERC20Deposit(), and so on.
3. lockNFTs(). External calls of .safeTransferFrom(), .mintLiquidityLockNFT() before writing of a lot of state variables.
4. lockTokens(). Similarly to the function lockNFTs().

Recommendation:
Use the nonReentrant() modifier from the OpenZeppelin ReentrancyGuard contract for these functions.

MEDIUM | VERIFIED

Returned value after the call is not checked.

LockToken.sol: function _chargeFees(), line 762.
Variable “refundSuccess”, returned after the call is not verified to be true. Thus, in case the call is reverted, the whole transaction won’t revert.

Recommendation:
Add validation of “refundSuccess” or verify that the check is not necessary.

From client:
The call is performed for refunding extra ETH to the user. A failure of such a refund should not affect call execution.

LOW | RESOLVED

Extra check.

LockToken.sol: there is an unnecessary requirement of (_tokenId >= 0, “Invalid token id”) on line 174 because this condition is provided by checking for the uint256 type.

Recommendation:
Remove extra check.

INFORMATIONAL | UNRESOLVED

Outdated Solidity version.

Currently the protocol utilizes Solidity version 0.6.2. The general auditor's security checklist recommends to utilize the newest stable version of Solidity. The newest version of Solidity includes the latest optimization for the compiler, the latest bugfixes and features such as built-in overflow and underflow validations.

Recommendation:
Use the latest version of the solc and OpenZeppelin products.

From client:
Due to the contract architecture, the version won't be updated as this can lead to possible storage issues connected to Openzeppelin contracts.

INFORMATIONAL | RESOLVED

Non-informative event.

LockToken.sol: the LogWithdrawal event only reports the destination address and transferred amount but does not carry information about the token address and token ID in the case of an NFT.

Recommendation:
Extend this event and add an event for the NFT case.

INFORMATIONAL | RESOLVED

Unnecessary gas consumption.

LockToken.sol:

- There are unaggregated writings of struct objects to the storage on lines 138—142, 184—189, 677—682.
  Recommendation. Use assignment with a struct object: for lines 138—142 "lockedToken[_id] = Items(_tokenAddress, _withdrawalAddress, amountIn, _unlockTime, false);", similarly for the others.
- There are multiple accesses to storage without a pointer. On lines 135, 181, 276, 280, 304, 308, 685, 701, 716, 730—735, etc.
  Recommendation. Use storage references for these cases.
- The following functions are not called internally but have public visibility: lockTokens(), lockNFTs(), extendLockDuration(), transferLocks(), withdrawTokens(), setNFTContract(), getTotalTokenBalance(), getTokenBalanceByAddress(), getAllDepositIds(), getDepositDetails(), getDepositsByWithdrawalAddress(), migrate(), mintNFTforLock(), transferOwnershipNFTContract().
  Recommendation. They should be declared external.

LockNFT.sol:

- The following functions are not called internally but have public visibility: setLockTokenAddress(), burn().
  Recommendation. They should be declared external.

INFORMATIONAL | RESOLVED

Extra getter.

LockToken.sol: the walletTokenBalance mapping has a public visibility, so there is no need in the function getTokenBalanceByAddress().

Recommendation:
Remove the extra getter.

INFORMATIONAL | RESOLVED

Lack of events.

LockToken.sol: there are no events about writing the state variables for functions extendLockDuration(), setFeeParams(), setFeesInUSD(), setCompanyWallet(), setNonFungiblePositionManager(), setV3Migrator(), setNFTContract(), addTokenToFreeList(), removeTokenFromFreeList().
LockNFT.sol: setLockTokenAddress().
This makes it difficult to track changes of the storage when using the contract.

Recommendation:
Add events about writing the state variables.

From client:
Events were added where necessary. In order not to increase contract size, some setters won't have an event.

INFORMATIONAL | UNRESOLVED

Unused function parameters.

NFTSVG.sol: generateSVG1(), generateSVG2(), generateSVG3(), lines 53, 72, 85: function parameter "params" is not used.
LockNFT: mintLiquidityLockNFT(), line 39: parameters "_tokenAddress", "_tokenAmount", "_tokenType", "_unlockDate" are not used.
NFTDescriptor.sol: tokenURI(), line 10: variable "nftTokenId" is not used.

Recommendation:
Remove unused parameters.

INFORMATIONAL | UNRESOLVED

Failing tests in the project.

All tests show an error in deploying the LockNFT contract (missing NFTDescriptor implementation). If the implementation is fixed, there is a problem in “Transfer/Withdrawal Lock TestCases” in revert reason (Expected transaction to be reverted with the reason 'Unauthorized to unlock', but it is reverted with the reason 'Unauthorised to unlock'). In “LockToken TestCases for ERC20 Tokens”, check the withdrawal address and owner of lockNFT. In “LockToken TestCases for ERC20 Tokens”, not enough funds to send transaction.

Recommendation:
Add an implementation of NFTDescriptor, fix revert string in the contract, fix the address check, send less or top up account to send tx.

	LockNFT.sol	NFTSVG.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions/Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	NFTDescriptor.sol	LockToken.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions/Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security (During the 1st iteration)

As a part of our work assisting TrustSwap in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Truffle testing framework.

The tests were based on the functionality of the code, as well as a review of the TrustSwap contract requirements for details about issuance amounts and how the system handles these.

LockNFT

- ✓ has a name and a symbol
- ✓ has the right token address and can change the address
- ✓ mints an NFT to the owner address
- ✓ transfers an NFT to the user (59ms)
- ✓ burns an NFT from the user
- ✓ should lock tokens and extend lock duration (113ms)
- ✓ should lock tokens and withdraw (238ms)
- ✓ should lock an NFT and extend lock duration (163ms)
- ✓ should lock a token without minting an NFT and withdraw (111ms)
- ✓ should lock an NFT and withdraw it (164ms)
- ✓ should transfer a lock for an NFT (83ms)
- ✓ should transfer a lock for tokens (55ms)
- ✓ pausable should work
- ✓ sets all params for the contract (96ms)
- ✓ adds and removes free tokens
- ✓ gets balances of tokens
- ✓ gets deposit information and fees in ETH (61ms)
- ✓ should migrate to a new version (240ms)
- ✓ should mint an NFT for lock (61ms)
- ✓ should transfer the ownership on the NFT lock contract
- ✓ should lock tokens with a fee (128ms)

FILE	% STMTS	% BRANCH	% FUNCS
LockToken.sol	99.1	88.19	100
LockNFT.sol	87.33	100	87.5
All files	**91.21**	**94**	**93.75**

Tests written by Zokyo Security (During the 2nd iteration)

LockNFT

- ✓ has a name and a symbol
- ✓ mints an NFT and gets info (3810ms)
- ✓ transfers an NFT and burns it (66ms)
- ✓ sets LockToken address
- ✓ sets NFTDescriptor address

FILE	% STMTS	% BRANCH	% FUNCS
LockNFT.sol	87.5	100	90
NFTDescriptor.sol	100	100	100
NFTSVG.sol	100	100	100
All files	**95.83**	**100**	**96.66**

The Zokyo Security team has carefully checked the whole set of unit tests prepared by the TrustSwap team (listed in the section below) and checked the original coverage of those tests. The audit tests were verified for the correctness, wholesomeness, sufficient coverage, meaning of the scenarios, overall structure, and the correct implementation (despite the covered functionality).

As a result, all the tests are verified, and the coverage is evaluated as conforming to security requirements. Original functionality has necessary coverage, Zokyo auditors confirmed standard and sub-standard functionality in a separate set of exploratory and manual testing scenarios.

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by the TrustSwap team

As a part of our work assisting the TrustSwap team in verifying the correctness of their contract code, we have checked the full set of unit tests prepared by the TrustSwap team.

It needs to be mentioned that the original code has a significant original coverage with testing scenarios provided by the TrustSwap team. All of them were also carefully checked by the team of auditors.

```
LockToken TestCases for ERC20 Tokens
Lock token address -> 0xDc64a140Aa3E981100a9becA4E685f962f0cF6C9 Descriptor deployed
at address -> 0x0165878A594ca255338adfa4d48449f69242Eb8F NFT contract address ->
0x2279B7A0a67DB372996a5FaB50D91eAA73d2eBe6 Transferred Ownership of NFT Contract to
Lock Token SC
Balance before minting Token1 0
Balance after minting Token1 200000000000000000000000000000
✓ Mints ERC20 Tokens of Token1 to specified Address
✓ Should grant approval of Token1 for LockToken SC
✓ Should lock tokens in Lock Token SC for Token1
LockToken TestCases for ERC20 Tokens
Uniswap factory Address -> 0x9A676e781A523b5d0C0e43731313A708CB607508 WETH Address
-> 0x0B306BF915C4d645ff596e518fAf3F9669b97016
Uniswap Router Address -> 0x959922bE3CAee4b8Cd9a407cc3ac1C251C2007B1 UnisSwap
Router WETH Address --> 0x0B306BF915C4d645ff596e518fAf3F9669b97016 ERC20 Token
Address: Token1 -> 0x9A9f2CCfdE556A7E9Ff0848998Aa4a0CFD8863AE ERC20 Token Address:
Token2 -> 0x68B1D87F95878fE05B998F19b66F4baba5De1aed Lock token address ->
0x3Aa5ebB10DC797CAC828524e59A333d0A371443c
NFTSVG Library Deployed to -> 0xc6e7DF5E7b4f2A278906862b61205850344D4e7d
NFTDescriptor address -> 0x59b670e9fA9D0A427751Af201D676719a970857b
NFT contract address -> 0x4ed7c70F96B99c776995fB64377f0d4aB3B0e1C1
Price estimator address -> 0xa85233C63b9Ee964Add6F2cffe00Fd84eb32338f Transferred
Ownership of NFT Contract to Lock Token SC
USDC Stable Coin Address -> 0x09635F643e140090A9A8Dcd712eD6285858ceBef USDC Decimals
--> 6
Balance before minting Token1 0
Balance after minting Token1 200000000000000000000
✓ Mints ERC20 Tokens of Token1 to specified Address
Balance before minting Token2 0
Balance after minting Token2 200000000000000000000
```

✓ Mints ERC20 Tokens of Token2 to specified Address
Balance before minting USDT 0
Balance after minting USDT 1000000000
✓ Mints USDT to the specified Address
✓ Grant approval of USDT for Uniswap Router
1 ETH set equal to ~900USDT
✓ Should create a liquidity pool for ETH/USDC in Uniswap
✓ Sets the Fee Params for the Lock Token Contract, Fee set to 10 USD
✓ Should revert on lockTokens call for Token1 as Lock Token SC has not been approved (83ms)
✓ Should grant approval of Token1 for LockToken SC
✓ Should grant approval of Token2 for LockToken SC
✓ Should revert on lockTokens call for Token1 when insufficient ETH sent in txns
✓ Should revert on lockTokens call for Token1 when token amount is 0
✓ Should revert on lockTokens call for Token1 when fee difference is >5%
NFT Minted with ID -> 1
✓ Should lock tokens in Lock Token SC for Token1 (57ms)
NFT Minted with ID -> 2
✓ Should lock tokens in Lock Token SC for Token2 when Exact ETH sent for Fee (65ms)
Pre Locking balance of Token2 Owner -> BigNumber { value: "9998877753058900958701" }
✓ Should lock tokens in Lock Token SC for Token2 when ETH sent > minFeeRequired and should return the Extra ETH
✓ Should transfer correct amount of Locked Tokens of Token2 into the Lock Token SC (47ms)
✓ Should return Null values for getDepositDetails call, for depositId that does not exist
LockToken TestCases for getFeesInETH
Uniswap factory Address -> 0x0E801D84Fa97b50751Dbf25036d067dCf18858bF
WETH Address -> 0x8f86403A4DE0BB5791fa46B8e795C547942fE4Cf
Uniswap Router Address -> 0x9d4454B023096f34B160D6B654540c56A1F81688 UnisSwap Router
WETH Address --> 0x8f86403A4DE0BB5791fa46B8e795C547942fE4Cf ERC20 Token Address: Token1 -> 0x5eb3Bc0a489C5A8288765d2336659EbCA68FCd00 ERC20 Token Address: Token2 -> 0x36C02dA8a0983159322a80FFE9F24b1acfF8B570 Lock token address -> 0x809d550fca64d94Bd9F66E60752A544199cfAC3D NFTDescriptor address -> 0x4c5859f0F772848b2D91F1D83E2Fe57935348029
NFT contract address -> 0x1291Be112d480055DaFd8a610b7d1e203891C274
Price estimator address -> 0x5f3f1dBD7B74C6B46e8c44f98792A1dAf8d69154 Transferred Ownership of NFT Contract to Lock Token SC
USDC Stable Coin Address -> 0x82e01223d51Eb87e16A03E24687EDF0F294da6f1 USDC Decimals --> 6
Balance before minting Token1 0
Balance after minting Token1 100000000000000000000
✓ Mints ERC20 Tokens of Token1 to specified Address
Balance before minting Token2 0
Balance after minting Token2 100000000000000000000
✓ Mints ERC20 Tokens of Token2 to specified Address

✓ Grant ERC20 Token approval for Uniswap router for both Token1 and Token2
Balance before minting USDT 0
Balance after minting USDT 1000000000
✓ Mints USDT to the specified Address
✓ Grant approval of USDT for Uniswap Router
1 ETH set equal to ~900USDT
✓ Should create a liquidity pool for ETH/USDC in Uniswap
✓ Should create a liquidity pool for Token2/ETH in Uniswap
✓ Should create a liquidity pool in Uniswap of Token1/Token2
✓ Should revert when setFeeParams is called and price estimator address is not a contract
✓ Should revert when setFeeParams is called and price estimator address is zero address
✓ Should revert when setFeeParams is called and usdc address is not a contract
✓ Should revert when setFeeParams is called and usdc address is zero address
✓ Should revert when setFeeParams is called and usdc fee 0
✓ Should revert when setFeeParams is called and company wallet is a zero address
✓ Should return Zero fee when getFeesInEth is called and fee params have not been set
✓ Sets the Fee Params for the Lock Token Contract, Fee set to 10 USD
✓ Should Revert when fee is set to 0 USD
✓ Should revert when fee is being set by an address other than the owner
✓ Should revert when company wallet is set to 0 Address
✓ Should revert when company wallet is being set by an address other than the owner
Fees in ETH from Smart Contract--> 0.01112347052280312 Fee in ETH from Manual Calculations --> 0.011111111111111112
✓ Should return correct fees in eth, when Fee is USD 10
✓ Should set the Fee to 25 USD
Fees in ETH from Smart Contract--> 0.0278086763070078
Fee in ETH from Manual Calculations --> 0.027777777777777776
✓ Should return correct fees in eth, when Fee is USD 25
✓ Should revert when addTokenToFreeList is called by an address other than the owner
✓ Should revert when addTokenToFreeList is called and token address is not a contract
✓ Should revert when addTokenToFreeList is called and token address is zero address
✓ Should add Token1 to free token list
✓ Should return Zero when getFeesInEth is called for free token, Token1
getDepositByWithdrawalAddress TestCases for ERC20 Tokens & NFTs
Uniswap factory Address -> 0x0355B7B8cb128fA5692729Ab3AAa199C1753f726
WETH Address -> 0x202CCe504e04bEd6fC0521238dDf04Bc9E8E15aB
Uniswap Router Address -> 0xf4B146FbA71F41E0592668ffbF264F1D186b2Ca8
UnisSwap Router WETH Address --> 0x202CCe504e04bEd6fC0521238dDf04Bc9E8E15aB ERC20 Token Address: Token1 -> 0x172076E0166D1F9Cc711C77Adf8488051744980C ERC20 Token Address: Token2 -> 0x4EE6eCAD1c2Dae9f525404De8555724e3c35d07B
Lock token address -> 0xBEc49fA140aCaA83533fB00A2BB19bDdd0290f25
NFTDescriptor address -> 0xD84379CEae14AA33C123Af12424A37803F885889

NFT contract address -> 0x2B0d36FACD61B71CC05ab8F3D2355ec3631C0dd5
NFT contract address for Users personal NFT's that will be Locked-> 0x46b142DD1E924FAb83eCc3c08e4D46E82f005e0E
Price estimator address -> 0xC9a43158891282A2B1475592D5719c001986Aaec Transferred Ownership of NFT Contract to Lock Token SC
USDC Stable Coin Address -> 0x4C2F7092C2aE51D986bEFEe378e50BD4dB99C901 USDC Decimals --> 6
Balance before minting Token1 0
Balance after minting Token1 200000000000000000000
✓ Mints ERC20 Tokens of Token1 to specified Address
Balance before minting Token2 0
Balance after minting Token2 200000000000000000000
✓ Mints ERC20 Tokens of Token2 to specified Address
Balance before minting USDT 0
Balance after minting USDT 1000000000
✓ Mints USDT to the specified Address
✓ Grant approval of USDT for Uniswap Router
1 ETH set equal to ~900USDT
✓ Should create a liquidity pool for ETH/USDC in Uniswap
✓ Sets the Fee Params for the Lock Token Contract, Fee set to 10 USD
✓ Should grant approval of Token1 for LockToken SC
✓ Should grant approval of Token2 for LockToken SC
NFT Minted with ID -> 1
✓ Should lock tokens in Lock Token SC for Token1 & Should mint NFT, Deposit ID 1 (53ms)
NFT Minted with ID -> 2
✓ Should lock tokens in Lock Token SC for Token2 & not mint NFT, Deposit ID 2 (60ms)
✓ Should mint a personal NFT to the User to be Locked in the LockToken SC
✓ Should grant approval for LockToken SC for User Personal NFT
NFT Minted with ID -> 3
✓ Should lock the users personal NFT in the Lock Token SC & not mint NFT, Deposit ID 3 (55ms)
✓ Should lock Token1 Tokens into LockToken SC & Not Mint NFT, Deposit ID 4 (50ms)
✓ Should mint user personal NFT and lock it in LockToken SC & Not mint Lock NFT, Deposit ID 5 (79ms)
✓ Should mint NFT for ERC20 lock with deposit ID 4, post lockTokens call
✓ Should mint NFT for NFT lock with deposit ID 5, post lockTokens call
✓ Should transfer the lock NFT from addr1 to addr3 (38ms)
LockToken-NFT TestCases
Uniswap factory Address -> 0xCace1b78160AE76398F486c8a18044da0d66d86D
WETH Address -> 0xD5ac451B0c50B9476107823Af206eD814a2e2580
Uniswap Router Address -> 0xF8e31cb472bc70500f08Cd84917E5A1912Ec8397 UnisSwap Router WETH Address --> 0xD5ac451B0c50B9476107823Af206eD814a2e2580 Lock token address -> 0xc0F115A19107322cFBf1cDBC7ea011C19EbDB4F8

NFTDescriptor address -> 0xc96304e3c037f81dA488ed9dEa1D8F2a48278a75
NFT contract address -> 0x34B40BA116d5Dec75548a9e9A8f15411461E8c70
NFT contract address for Users personal NFT's that will be Locked->
0xD0141E899a65C95a556fE2B27e5982A6DE7fDD7A
Price estimator address -> 0x07882Ae1ecB7429a84f1D53048d35c4bB2056877 Transferred
Ownership of NFT Contract to Lock Token SC
USDC Stable Coin Address -> 0xfaAddC93baf78e89DCf37bA67943E1bE8F37Bb8c
USDC Decimals --> 6
Balance before minting USDT 0
Balance after minting USDT 1000000000
✓ Mints USDT to the specified Address
✓ Grant approval of USDT for Uniswap Router
1 ETH set equal to ~900USDT
✓ Should create a liquidity pool for ETH/USDC in Uniswap
✓ Sets the Fee Params for the Lock Token Contract, Fee set to 75 USD
✓ Should mint a personal NFT to the User to be Locked in the LockToken SC
✓ Should revert when user personal NFT is attempted to be locked and LockToken SC is not Approved to transfer NFT
✓ Should grant approval for LockToken SC for User Personal NFT
✓ Should revert when lockNFT called for user NFT and amount = 0
✓ Should revert when lockNft called for user NFT and unlockTime is invalid
✓ Should lock the users personal NFT in the Lock Token SC (56ms)
Transfer Lock TestCases for ERC20 Tokens & NFTs
Uniswap factory Address -> 0x3aAde2dCD2Df6a8cAc689EE797591b2913658659
WETH Address -> 0xab16A69A5a8c12C732e0DEFF4BE56A70bb64c926
Uniswap Router Address -> 0xE3011A37A904aB90C8881a99BD1F6E21401f1522 UnisSwap Router
WETH Address --> 0xab16A69A5a8c12C732e0DEFF4BE56A70bb64c926 ERC20 Token Address:
Token1 -> 0x1f10F3Ba7ACB61b2F50B9d6DdCf91a6f787C0E82 ERC20 Token Address: Token2 ->
0x457cCf29090fe5A24c19c1bc95F492168C0EaFdb
Lock token address -> 0x525C7063E7C20997BaaE9bDa922159152D0e8417
NFTSVG Library Deployed to -> 0x38a024C0b412B9d1db8BC398140D00F5Af3093D4 NFTDescriptor address -> 0x5fc748f1FEb28d7b76fa1c6B07D8ba2d5535177c
NFT contract address -> 0xB82008565FdC7e44609fA118A4a681E92581e680
NFT contract address for Users personal NFT's that will be Locked->
0x2a810409872AfC346F9B5b26571Fd6eC42EA4849
Price estimator address -> 0xb9bEECD1A582768711dE1EE7B0A1d582D9d72a6C Transferred
Ownership of NFT Contract to Lock Token SC
USDC Stable Coin Address -> 0xF32D39ff9f6Aa7a7A64d7a4F00a54826Ef791a55
USDC Decimals --> 6
Balance before minting Token1 0
Balance after minting Token1 200000000000000000000
✓ Mints ERC20 Tokens of Token1 to specified Address

Balance before minting Token2 0
Balance after minting Token2 200000000000000000000
✓ Mints ERC20 Tokens of Token2 to specified Address
Balance before minting USDT 0
Balance after minting USDT 1000000000
✓ Mints USDT to the specified Address
✓ Grant approval of USDT for Uniswap Router
1 ETH set equal to ~900USDT
✓ Should create a liquidity pool for ETH/USDC in Uniswap
✓ Sets the Fee Params for the Lock Token Contract, Fee set to 10 USD
✓ Should grant approval of Token1 for LockToken SC
✓ Should grant approval of Token2 for LockToken SC
NFT Minted with ID -> 1
✓ Should lock tokens in Lock Token SC for Token1 & Should mint NFT (54ms)
NFT Minted with ID -> 2
✓ Should lock tokens in Lock Token SC for Token2 & not mint NFT (59ms)
✓ Should mint a personal NFT to the User to be Locked in the LockToken SC
✓ Should grant approval for LockToken SC for User Personal NFT
NFT Minted with ID -> 3
✓ Should lock the users personal NFT in the Lock Token SC & not mint NFT (56ms)
✓ Should withdraw Token1 Tokens previously locked in the lockToken SC with Deposit ID 1
✓ Should withdraw Token2 Tokens previously locked in the lockToken SC with Deposit ID 2
✓ Should withdraw the NFT previously locked in the lockToken SC with Deposit ID 3
✓ Should lock Token1 Tokens into LockToken SC & Not Mint NFT (46ms)
✓ Should mint user personal NFT and lock it in LockToken SC & Not mint Lock NFT (73ms)
✓ Should revert when attempting to mint NFT for deposit ID 1 that already has an NFT
✓ Should revert when attempting to mint lock NFT for personal NFT already withdrawn
✓ Should revert when attempting to mint lock NFT for deposit ID 2 ERC20 Tokens that have already been withdrawn
✓ Should revert when attempting to mint lock NFT from unauthenticated address TOKEN URI

✓ Should mint NFT for ERC20 lock with deposit ID 4, post lockTokens call (3858ms)
✓ Should mint NFT for NFT lock with deposit ID 5, post lockTokens call
Transfer Lock TestCases for ERC20 Tokens & NFTs
Uniswap factory Address -> 0x638A246F0Ec8883eF68280293FFE8Cfbabe61B44
WETH Address -> 0x6C2d83262fF84cBaDb3e416D527403135D757892
Uniswap Router Address -> 0xFD6F7A6a5c21A3f503EBaE7a473639974379c351 UnisSwap Router WETH Address --> 0x6C2d83262fF84cBaDb3e416D527403135D757892 ERC20 Token Address: Token1 -> 0xa6e99A4ED7498b3cdDCBB61a6A607a4925Faa1B7 ERC20 Token Address: Token2 -> 0x5302E909d1e93e30F05B5D6Eea766363D14F9892 Lock token address -> 0x0ed64d01D0B4B655E410EF1441dD677B695639E7 NFTDescriptor address -> 0x4bf010f1b9beDA5450a8dD702ED602A104ff65EE
NFT contract address -> 0x40a42Baf86Fc821f972Ad2aC878729063CeEF403 NFT contract address for Users personal NFT's that will be Locked-> 0x96F3Ce39Ad2BfDCf92C0F6E2C2CAbF83874660Fc
Price estimator address -> 0x986aaa537b8cc170761FDAC6aC4fc7F9d8a20A8C Transferred Ownership of NFT Contract to Lock Token SC
USDC Stable Coin Address -> 0x870526b7973b56163a6997bB7C886F5E4EA53638 USDC Decimals --> 6
Balance before minting Token1 0
Balance after minting Token1 200000000000000000000
✓ Mints ERC20 Tokens of Token1 to specified Address
Balance before minting Token2 0
Balance after minting Token2 200000000000000000000
✓ Mints ERC20 Tokens of Token2 to specified Address
Balance before minting USDT 0
Balance after minting USDT 1000000000
✓ Mints USDT to the specified Address
✓ Grant approval of USDT for Uniswap Router
1 ETH set equal to ~900USDT

✓ Should create a liquidity pool for ETH/USDC in Uniswap
✓ Sets the Fee Params for the Lock Token Contract, Fee set to 10 USD
✓ Should grant approval of Token1 for LockToken SC
✓ Should grant approval of Token2 for LockToken SC
✓ Should lock tokens in Lock Token SC for Token1 & not mint NFT (48ms)
✓ Should lock tokens in Lock Token SC for Token2 when Exact ETH sent for Fee & not Mint NFT (59ms)
✓ Should mint a personal NFT to the User to be Locked in the LockToken SC
✓ Should grant approval for LockToken SC for User Personal NFT
✓ Should lock the users personal NFT in the Lock Token SC & not mint NFT (54ms)
✓ Should revert when transfer called for user personal NFT with depositId 3 from unauthenticated address
✓ Should transfer Locks for user personal NFTLock with depositId 3 to new receiver address (41ms)
NFT Minted with ID--> 4
✓ Should lock tokens in Lock Token SC for Token1 & mint NFT (50ms)
NFT Minted with ID--> 5
✓ Should lock tokens in Lock Token SC for Token2 when Exact ETH sent for Fee & Mint NFT (64ms)
✓ Should revert when transferLock called for deposit ID 4 from authenticated address
✓ Should revert when transferLock called for deposit ID 5 from authenticated address
✓ Should revert when transferLock called for deposit ID 4 with lock NFT Minted
✓ Should revert when transferLock called for deposit ID 5 with lock NFT Minted
✓ Should transfer lock NFT for Deposit Id 4 from addr1 to addr3 (39ms)
✓ Should transfer lock NFT for Deposit Id 5 from addr1 to addr3 (42ms)
✓ Should mint user personal NFT & Lock it in LockToken SC, Mint Lock NFT, Deposit ID 6 (101ms)
✓ Should revert when transferLock called for deposit ID 6 from authenticated address
✓ Should revert when transferLock called for deposit ID 6 with lock NFT Minted
✓ Should transfer lock NFT for Deposit Id 6 from addr1 to addr3 (43ms)
LockToken-UniSwap TestCases for Uniswap Liquidity/Pool Tokens
Uniswap factory Address -> 0xddE78e6202518FF4936b5302cC2891ec180E8bFf
WETH Address -> 0xB06c856C8eaBd1d8321b687E188204C1018BC4E5
Uniswap Router Address -> 0xaB7B4c595d3cE8C85e16DA86630f2fc223B05057
UnisSwap Router WETH Address --> 0xB06c856C8eaBd1d8321b687E188204C1018BC4E5 ERC20 Token Address: Token1 -> 0xAD523115cd35a8d4E60B3C0953E0E0ac10418309 ERC20 Token Address: Token2 -> 0x045857BDEAE7C1c7252d611eB24eB55564198b4C Lock token address -> 0x2b5A4e5493d4a54E717057B127cf0C000C876f9B
NFTDescriptor address -> 0x413b1AfCa96a3df5A686d8BFBF93d30688a7f7D9
NFT contract address -> 0x02df3a3F960393F5B349E40A599FEda91a7cc1A7
Price estimator address -> 0x821f3361D454cc98b7555221A06Be563a7E2E0A6 Transferred Ownership of NFT Contract to Lock Token SC
USDC Stable Coin Address -> 0x71089Ba41e478702e1904692385Be3972B2cBf9e
USDC Decimals --> 6
Balance before minting Token1 0
Balance after minting Token1 200000000000000000000

✓ Mints ERC20 Tokens of Token1 to specified Address
Balance before minting Token2 0
Balance after minting Token2 200000000000000000000
✓ Mints ERC20 Tokens of Token2 to specified Address
✓ Grant ERC20 Token approval for Uniswap router for both Token1 and Token2
Balance before minting USDT 0
Balance after minting USDT 1000000000
✓ Mints USDT to the specified Address
✓ Grant approval of USDT for Uniswap Router
1 ETH set equal to ~900USDT
✓ Should create a liquidity pool for ETH/USDC in Uniswap
✓ Sets the Fee Params for the Lock Token Contract, Fee set to 75 USD
✓ Should create a liquidity pool in Uniswap of Token1/Token2 and ensure Liquidity/Pool tokens are Minted (38ms)
✓ Should grant approval for Lock Token of Liquidity/Pool Tokens
NFT Minted with ID -> 1
✓ Should Lock Tokens in LockToken SC for Liquidity/Pool Tokens (47ms)
✓ Should Increase Liquidity Within Token1/Token2 Uniswap Pool and Mint Additional Liquidity/Pool Tokens
✓ Should grant approval for Lock Token of Additional Liquidity/Pool Tokens that were Minted
NFT Minted with ID -> 2
✓ Should Lock Tokens in LockToken SC for Additional Liquidity/Pool Tokens that were Minted (48ms)
Withdrawal Lock TestCases for ERC20 Tokens & NFTs
Uniswap factory Address -> 0xE8addD62feD354203d079926a8e563BC1A7FE81e
WETH Address -> 0xe039608E695D21aB11675EBBA00261A0e750526c
Uniswap Router Address -> 0x071586BA1b380B00B793Cc336fe01106B0BFbE6D UnisSwap Router
WETH Address --> 0xe039608E695D21aB11675EBBA00261A0e750526c ERC20 Token Address: Token1 > 0xe70f935c32dA4dB13e7876795f1e175465e6458e ERC20 Token Address: Token2 -> 0x3C15538ED063e688c8DF3d571Cb7a0062d2fB18D Lock token address -> 0xccf1769D8713099172642EB55DDFFC0c5A444FE9 NFTDescriptor address -> 0x3904b8f5b0F49cD206b7d5AABeE5D1F37eE15D8d
NFT contract address -> 0x2Dd78Fd9B8F40659Af32eF98555B8b31bC97A351
NFT contract address for Users personal NFT's that will be Locked-> 0x56fC17a65ccFEC6B7ad0aDe9BD9416CB365B9BE8
Price estimator address -> 0x2625760C4A8e8101801D3a48eE64B2bEA42f1E96 Transferred Ownership of NFT Contract to Lock Token SC
USDC Stable Coin Address -> 0x139e1D41943ee15dDe4DF876f9d0E7F85e26660A
USDC Decimals --> 6
Balance before minting Token1 0
Balance after minting Token1 200000000000000000000
✓ Mints ERC20 Tokens of Token1 to specified Address
Balance before minting Token2 0

Balance after minting Token2 200000000000000000000
✓ Mints ERC20 Tokens of Token2 to specified Address
Balance before minting USDT 0
Balance after minting USDT 1000000000
✓ Mints USDT to the specified Address
✓ Grant approval of USDT for Uniswap Router
1 ETH set equal to ~900USDT
✓ Should create a liquidity pool for ETH/USDC in Uniswap
✓ Sets the Fee Params for the Lock Token Contract, Fee set to 10 USD
✓ Should grant approval of Token1 for LockToken SC
✓ Should grant approval of Token2 for LockToken SC
NFT Minted with ID -> 1
✓ Should lock tokens in Lock Token SC for Token1 & mint lock NFT (49ms)
NFT Minted with ID -> 2
✓ Should lock tokens in Lock Token SC for Token2 when Exact ETH sent for Fee & mint lock NFT (60ms
✓ Should mint a personal NFT to the User to be Locked in the LockToken SC
✓ Should grant approval for LockToken SC for User Personal NFT
✓ Should lock the users personal NFT with depositID 3 in the Lock Token SC & not mint NFT (54ms)
✓ Should revert with Unlock Time message when withdrawal attempted on Token1 with Deposit ID 1 when Unlock time not reached
✓ Should revert with Unlock Time message when withdrawal attempted on user NFT with Deposit ID 3 when Unlock time not reached
✓ Should revert when unauthorized address attempts to withdraw Token1 Lock with Deposit ID 1
✓ Should revert when unauthorized address attempts to withdraw NFT Lock with Deposit ID 3
✓ Should withdraw Token1 Tokens previously locked in the lockToken SC with Deposit ID 1 (38ms)
✓ Should revert when locked Token1 tokens with deposit ID 1 attempted to be withdrawn again
✓ Should withdraw Token2 Tokens previously locked in the lockToken SC with Deposit ID 2
✓ Should withdraw the NFT previously locked in the lockToken SC with Deposit ID 3
✓ Should revert when NFT locked with deposit ID 3 attempted to be withdrawn again
✓ Should lock Token1 Tokens into LockToken SC with deposit ID 4 & Mint NFT (49ms)
✓ Should revert when lockTokens with deposit ID 4 withdrawn from address not owning the lock NFT
✓ Should revert when lockTokens with deposit ID 4 withdrawn & unlock time not reached & caller owning the lock NFT
✓ Should withdraw Token1 locked tokens with deposit ID 4 & caller owns lock NFT (38ms)
✓ Should mint user personal NFT, and locked it in LockToken SC & mint lock NFT, deposit ID 5 (81ms)
✓ Should revert when locked user NFT withdrawn from address not owning lock NFT, deposit ID 5
✓ Should revert when locked user NFT with deposit ID 5 withdrawn & unlock time not reached & caller owning the lock NFT
✓ Should withdraw locked user NFT with deposit ID 5 & caller owns lock NFT (39ms)
✓ Should lock Token1 Tokens into LockToken SC with deposit ID 6 & Mint NFT to addr1 (49ms)
✓ Should transfer the lock NFT of deposit ID 6 from addr1 to addr2 (45ms)
✓ Should revert when lockTokens with deposit ID 6 withdrawn from address not owning the lock NFT

✓ Should revert when lockTokens with deposit ID 6 withdrawn & unlock time not reached & caller owr the lock NFT
✓ Should withdraw Token2 locked tokens with deposit ID 6 & caller owns lock NFT
✓ Should mint user personal NFT, and lock it in LockToken SC & mint lock NFT to addr1, deposit ID 7 (79ms)
✓ Should transfer the lock NFT of deposit ID 7 from addr1 to addr2 (46ms)
✓ Should revert when user NFT with deposit ID 7 withdrawn from address not owning the lock NFT
✓ Should revert when user NFT with deposit ID 7 withdrawn & unlock time not reached & caller ownin the lock NFT
✓ Should withdraw locked user NFT with deposit ID 7 & caller owns lock NFT (39ms)
✓ Should lock Token2 Tokens into LockToken SC with deposit ID 8 & Mint NFT to addr1 (48ms)
✓ Should transfer lock NFT with Deposit ID 8, from addr1 to addr2, addr2 to addr3 and addr3 back to addr1 (207ms)
✓ Should mint user personal NFT, and lock it in LockToken SC & mint lock NFT to addr1, deposit ID 9 (91ms)
✓ Should transfer lock NFT with Deposit ID 9, from addr1 to addr2, addr2 to addr3 and addr3 back to addr1 (76ms)
✓ Should return proper values for getDepositsByWithdrawalAddress for addr1, addr2, addr3
✓ Should return proper values for getDepositsByWithdrawalAddress for addr2, once all lock NFTs transferred to it (61ms)
NFT Liquidity Lock TestCases
Uniswap factory Address -> 0x114e375B6FCC6d6fCb68c7A1d407E652C54F25FB
WETH Address -> 0xcD0048A5628B37B8f743cC2FeA18817A29e97270
Uniswap Router Address -> 0x976C214741b4657bd99DFD38a5c0E3ac5C99D903 UnisSwap Router
WETH Address --> 0xcD0048A5628B37B8f743cC2FeA18817A29e97270 ERC20 Token Address:
Token1 -> 0x8bEe2037448F096900Fd9affc427d38aE6CC0350 ERC20 Token Address: Token2 ->
0x942ED2fa862887Dc698682cc6a86355324F0f01e Lock token address ->
0x8D81A3DCd17030cD5F23Ac7370e4Efb10D2b3cA4 NFTDescriptor address ->
0xcC4c41415fc68B2fBf70102742A83cDe435e0Ca7
NFT contract address -> 0xa722bdA6968F50778B973Ae2701e90200C564B49
NFT contract address for Users personal NFT's that will be Locked->
0xc7cDb7A2E5dDa1B7A0E792Fe1ef08ED20A6F56D4
Price estimator address -> 0x967AB65ef14c58bD4DcfFeaAA1ADb40a022140E5 Transferred
Ownership of NFT Contract to Lock Token SC
USDC Stable Coin Address -> 0x8e264821AFa98DD104eEcfcfa7FD9f8D8B320adA
USDC Decimals --> 6
Balance before minting Token1 0
Balance after minting Token1 200000000000000000000
✓ Mints ERC20 Tokens of Token1 to specified Address
Balance before minting Token2 0
Balance after minting Token2 200000000000000000000
✓ Mints ERC20 Tokens of Token2 to specified Address

```
Balance before minting USDT 0
Balance after minting USDT 1000000000
✓ Mints USDT to the specified Address
✓ Grant approval of USDT for Uniswap Router
1 ETH set equal to ~900USDT
✓ Should create a liquidity pool for ETH/USDC in Uniswap
✓ Sets the Fee Params for the Lock Token Contract, Fee set to 10 USD
✓ Should grant approval of Token1 for LockToken SC
✓ Should grant approval of Token2 for LockToken SC
NFT Minted with ID -> 1
✓ Should lock tokens in Lock Token SC for Token1 & mint lock NFT (54ms)
✓ Should mint userNFT
BigNumber { value: "120559" }
✓ Should transfer userNFT and give us the estimated gas, ~120454
BigNumber { value: "172302" }
✓ Should transfer lock NFT and give us the estimated gas, ~221806
189 passing (14s)
9 failing
    LockToken TestCases for ERC20 Tokens
    Should lock tokens in Lock Token SC for Token1:
    AssertionError: expected '0xf39Fd6e51aad88F6F4ce6aB8827279cffFb...' to equal
    '0x3C44CdDdB6a900fa2b585dd299e03d12FA4...'
✓   + expected - actual
    -0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
    +0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC
    at Context.<anonymous> (test/deploy.js:176:41)
    LockToken TestCases for ERC20 Tokens
    Should lock tokens in Lock Token SC for Token2 when ETH sent > minFeeRequired and
    should return the Extra ETH:
    InvalidInputError: sender doesn't have enough funds to send tx. The max upfront cost is:
    1100010652951000000000000 and the sender's account only has: 9998877753058900958701
    at TxPool._validateTransaction (node_modules/hardhat/src/internal/hardhat-network/provider/
    TxPool.ts:447:13)
✓   at TxPool.addTransaction (node_modules/hardhat/src/internal/hardhat-network/provider/
    TxPool.ts:112:5)
    at HardhatNode._addPendingTransaction (node_modules/hardhat/src/internal/hardhat-network/
    provider/node.ts:1587:5)
    at HardhatNode._mineTransaction
    (node_modules/hardhat/src/internal/hardhat-network/provider/node.ts:1595:5)
    at EthModule._sendTransactionAndReturnHash (node_modules/hardhat/src/internal/hardhat-
    network/provider/modules/eth.ts:1504:18)
    at HardhatNetworkProvider.request
```

(node_modules/hardhat/src/internal/hardhat-network/provider/provider.ts:118:18)
at EthersProviderWrapper.send (node_modules/@nomiclabs/hardhat-ethers/src/internal/ethers-provider-wrapper.ts:13:20)
Transfer Lock TestCases for ERC20 Tokens & NFTs
✓ Should revert when transfer called for user personal NFT with depositId 3 from unauthenticated address:
AssertionError: Expected transaction to be reverted with reason 'Unauthorized to transfer', but it reverted with reason 'Unauthorised to transfer'
Withdrawal Lock TestCases for ERC20 Tokens & NFTs
✓ Should revert when unauthorized address attempts to withdraw Token1 Lock with Deposit ID 1:
AssertionError: Expected transaction to be reverted with reason 'Unauthorized to unlock', but it reverted with reason 'Unauthorised to unlock'
Withdrawal Lock TestCases for ERC20 Tokens & NFTs
✓ Should revert when unauthorized address attempts to withdraw NFT Lock with Deposit ID 3:
AssertionError: Expected transaction to be reverted with reason 'Unauthorized to unlock', but it reverted with reason 'Unauthorised to unlock'
Withdrawal Lock TestCases for ERC20 Tokens & NFTs
✓ Should revert when lockTokens with deposit ID 4 withdrawn from address not owning the lock NFT:
AssertionError: Expected transaction to be reverted with reason 'Unauthorized to unlock', but it reverted with reason 'Unauthorised to unlock'
Withdrawal Lock TestCases for ERC20 Tokens & NFTs
✓ Should revert when locked user NFT withdrawn from address not owning lock NFT, deposit ID 5:
AssertionError: Expected transaction to be reverted with reason 'Unauthorized to unlock', but it reverted with reason 'Unauthorised to unlock'
Withdrawal Lock TestCases for ERC20 Tokens & NFTs
✓ Should revert when lockTokens with deposit ID 6 withdrawn from address not owning the lock NFT:
AssertionError: Expected transaction to be reverted with reason 'Unauthorized to unlock', but it reverted with reason 'Unauthorised to unlock'
Withdrawal Lock TestCases for ERC20 Tokens & NFTs
✓ Should revert when user NFT with deposit ID 7 withdrawn from address not owning the lock NFT:
AssertionError: Expected transaction to be reverted with reason 'Unauthorized to unlock', but it reverted with reason 'Unauthorised to unlock'

FILE	% STMTS	% BRANCH	% FUNCS
LockNFT.sol	75	50	70
NFTDescriptor.sol	100	100	100
NFTSVG.sol	100	100	100
All files	**91.66**	**83.33**	**90**

We are grateful for the opportunity to work with the TrustSwap team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the TrustSwap team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.