BAIL security

BAILSEC.IO

EMAIL : OFFICE@BAILSEC.IO

TWITTER : @BAILSECURITY

TELEGRAM : @HELLOATBAILSEC

FINAL REPORT:

Trustswap LockToken

March 2024

Disclaimer:

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

The content of this assessment is not an investment. The information provided in this report is for general informational purposes only and is not intended as investment, legal, financial, regulatory, or tax advice. The report is based on a limited review of the materials and documentation provided at the time of the audit, and the audit results may not be complete or identify all possible vulnerabilities or issues. The audit is provided on an "as-is," "where-is," and "as-available" basis, and the use of blockchain technology is subject to unknown risks and flaws.

The audit does not constitute an endorsement of any particular project or team, and we make no warranties, expressed or implied, regarding the accuracy, reliability, completeness, or availability of the report, its content, or any associated services or products. We disclaim all warranties, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We assume no responsibility for any product or service advertised or offered by a third party through the report, any open-source or third-party software, code, libraries, materials, or information linked to, called by, referenced by, or accessible through the report, its content, and the related services and products. We will not be liable for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract.

The contract owner is responsible for making their own decisions based on the audit report and should seek additional professional advice if needed. The audit firm or individual assumes no liability for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract. The contract owner agrees to indemnify and hold harmless the audit firm or individual from any and all claims, damages, expenses, or liabilities arising from the use or reliance on the audit report or the smart contract.

By engaging in a smart contract audit, the contract owner acknowledges and agrees to the terms of this disclaimer.

1. Project Details

Important:
Please ensure that the deployed contract matches the source-code of the last commit hash.

Project	TrustSwap LockToken
Website	trustswap.com
Type	ERC20 Token
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/trustswap/teamfinance-contract-tokenlock/blob/f8486570d0a2b1e07fbf903c632473d3f7336f29/contracts/LockToken.sol
Resolution 1	https://github.com/trustswap/teamfinance-contract-locktoken/blob/c4e987e09ab2b165ed1d96c7300d9f39b0a1d34f/contracts/LockToken.sol

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	1	1		
Medium	1	1		
Low	8	7		1
Informational	2			2
Configurational				
Governance	2			2
Quality assurance				
Total	14	9		5

2.1 Detection Definitions

Severity	Description
High	The problem poses a significant threat to the confidentiality of a considerable number of users' sensitive data. It also has the potential to cause severe damage to the client's reputation or result in substantial financial losses for both the client and the affected users.
Medium	While medium level vulnerabilities may not be easy to exploit, they can still have a major impact on the execution of a smart contract. For instance, they may allow public access to critical functions, which could lead to serious consequences.
Low	Poses a very low-level risk to the project or users. Nevertheless the issue should be fixed immediately
Informational	Effects are small and do not post an immediate danger to the project or users
Configurational	Issues which may arise due to different configurational settings
Governance	Governance privileges which can directly result in a loss of funds or other potential undesired behavior
Quality assurance	Aggregated minor issues, ensuring a high quality codebase.

3.Detection

LockToken

The LockToken contract presents a versatile custody solution designed for the secure management of ERC20 and ERC721 tokens. This innovative contract enables users to lock their tokens for a specified duration, offering an added layer of security and utility through its unique features.
Below is a detailed examination of its core functionalities and mechanisms.

Token Locking Mechanism

Token Types: The LockToken contract supports the locking of both ERC20 and ERC721 token standards, accommodating a wide range of digital assets.

Lock Duration: Users have the flexibility to lock their tokens for an arbitrary period, allowing for tailored time-based security measures.

NFT Correspondence: Each locked position has the option to be associated with a unique NFT. This NFT can be minted immediately following the token lock or at a later stage through a designated function for positions initially created without an NFT.

Advanced FunctionalitiesPosition Extension: Both ERC721 and ERC20 locked positions can have their lock period extended, offering users greater control over their digital assets.

ERC20 Position Splitting: Users have the capability to split ERC20 positions, creating an additional locked position. This feature enhances the flexibility and utility of locked ERC20 tokens.

Fees and TransactionsTransaction Fees: The creation or splitting of a lock incurs a transaction fee payable in ETH. This fee is to be provided as msg.value within the transaction.

Fee Structure: Fees can serve as either protocol fees or referral fees based on the input parameter. In the case of referral fees, a portion is allocated to the referrer and

the remainder to the protocol. Specific tokens or addresses can be whitelisted for the fee.

Lock Transferability and Withdrawal

Transfer Options: Locked positions are transferable. If a position is not linked to an NFT, it can be transferred internally within the LockToken contract. Positions associated with an NFT can be transferred via the NFT's simple transfer mechanism.

Withdrawal: Users can withdraw their locked position upon reaching the designated unlock time, ensuring access to their tokens when needed.

The LockToken contract is engineered to offer a secure and flexible environment for token custody, with a focus on ERC20 and ERC721 tokens. Its design incorporates innovative features such as lock period customization, NFT association, position extension, and splitting capabilities. The contract's fee mechanism and transfer options further enhance its utility and operational efficiency to incentivize usage via referrals.

Issue	Governance: Contract owner has full control over all positions with a corresponding NFT
Severity	Governance
Description	The contract owner can change the NFT address via the setNFTContract function. This can have the following impact: a) Set to a malicious contract, the owner can call the transferLocks function via the NFT contract and transfer positions to a designated address. b) Setting to an incorrect contract can brick withdrawal of such positions as the ownerOf check will not work anymore.
Recommendations	Consider incorporating a Gnosis Multisignature contract as owner and ensuring that the Gnosis participants are trusted entities.
Comments / Resolution	Acknowledged.

Issue	Governance: Certain settings can brick fee-related functions
Severity	Governance
Description	Functions which rely on fee correctness such as lockToken, lockNFT and splitLock can be DoS'ed by incorrect fee-related parameters such as unreasonably high fees, incorrect companyWallet or a flawed priceEstimator.
Recommendations	Consider incorporating a Gnosis Multisignature contract as owner and ensuring that the Gnosis participants are trusted entities.
Comments / Resolution	Acknowledged.

Issue	Advanced exploit allows to DoS positions permanently
Severity	High
Description	In order to remove a position, a loop is executed over all existing position until the desired position is found. This logic is handled within the _removeDepositsForWithdrawalAddress function. We observe the following facts: a) The loop is unbounded b) No large gas consumption is executed This opens the theoretical issue of DoS'ing the loop, however, in order to make such a DoS successful, a large amount of positions would need to be existent. This exploit is inherently limited due to the fee application upon deposits or splits. The problem is that the fee can be circumvented

	by manipulating an UniV2 pair in such a manner that the eth amount for 1e6 USDT returns 1 wei. Using this manipulation a user can create a large amount of locks on behalf of another user, specifically these locks would need to happen BEFORE the victim executes his lock, otherwise the loop would not go over all malicious locks. This can be done either pro-active or via frontrunning. On such a network as Polygon where gas is very cheap, this theoretical attack can become practical. Contrary, on Ethereum no one would attempt such an attack because a large amount of locks need to be created. One could now argue that the victim can withdraw all previous locks in order to decrease the array size. This argument is in itself correct, however, the attacker could deposit a malformed token or NFT that would simply revert on transfers which would therefore prevent withdrawals and the corresponding array size decrease. *This exploit also works by locking tokens for a very long period.
Recommendations	Consider switching to a reliable CL oracle which correctly fetches the ETH price. Moreover, governance should act very carefully when adding exempted tokens or whitelisted addresses, as these could still execute the exploit even with a CL oracle. An additional small validation is necessary to ensure the correctness for such a fundamental change.
Comments / Resolution	Resolved, the PriceEstimator now incorporates an external oracle.

Issue	Malicious user can manipulate UniV2 pool to steal fee from other users
Severity	Medium
Description	In a similar fashion as in the first issue, a referrer could manipulate the ETH price of the UniV2 pool. This would then manipulate the frontend calculation to eventually calculate more msg.value than necessary (this depends on, if the frontend is using the same calculation from the PriceEstimator contract). If the user now agrees for this high msg.value, the referrer will effectively receive more ETH than he should. This issue is only rated as medium because it would require unawareness of the user.
Recommendations	Consider switching to a reliable CL oracle which correctly fetches the ETH price. An additional small validation is necessary to ensure the correctness for such a fundamental change.
Comments / Resolution	Resolved, the PriceEstimator now incorporates an external oracle.

Issue	Violation of checks-effects-interactions pattern
Severity	Low
Description	Throughout the contract there are one or multiple spots which violate the checks-effects-interactions pattern, to ensure a protection against invalid states, all external calls should strictly be implemented after any checks and effects (state variable changes).

	L 165:
	referrer == address(0) ? _chargeFees(_tokenAddress) : _chargeFeesReferral(_tokenAddress, referrer);
	
	These functions are executing external calls.
	
	L 204:
	
	referrer == address(0) ? _chargeFees(_tokenAddress) : _chargeFeesReferral(_tokenAddress, referrer);
	
	These functions are executing external calls.
	
	L 464:
	
	_chargeFees(lockedERC20Address);
	
	This function is executing an external call.

Recommendations	Consider following the checks-effects-interactions pattern. *Since this contract is already live and will be upgraded, we recommend simply adding reentrancy guards to all functions and not manipulate the function logic itself. This should be done once the contract is redeployed in the future.
Comments / Resolution	Resolved.

Issue	Rounding issue in fee slippage calculation can be abused by users
Severity	Low
Description	The fee slippage calculation applies whenever the amount is smaller than the fee and aims to allow users to provide up to 5% less fee for their transaction. This mechanism was incorporated to counter any negative effects if the transaction is stuck for some time in the mempool. The arithmetic operation is as follows: (fee - amount) * 100 / fee real world values applied could be: (1e18 - 0.95e18) * 100 / 1e18 = 5 The calculation in itself is correct and works for its purpose, the problem is just that solidity rounds down. Consider this scenario: (1e18 - 0.941e18) * 100 / 1e18 = 5.9 -> 0 Therefore, users can still pass this calculation by abusing the rounding while providing in fact less than 95%.
Recommendations	Consider increasing the multiplier to BPS denomination.
Comments / Resolution	Resolved.

Issue	Fee calculation during getFeesInEth is using calculation for fee on raw amount
Severity	Low
Description	The fee calculation within the getFeesInEth function is on the raw amount instead of the total amount. This will result in slightly less fee than anticipated. To understand the flaw, consider a simple PoC: (Note that this PoC is without decimals to simplify it) amountIn = 100.3 this includes the 0.3% swap fee. Our goal is to deduct this swap fee. (total amount) The current calculation is as follows: amount * fee / divisor (raw calculation) 100.3 * 3 / 1000 = 0.3009 -> 100.3 - 0.3009 = 99.9991 As we can see, instead of 100 we have 99.9991 The problem here is that the fee is calculated based on the raw amount, while it should be calculated on the total amount: amount * fee / (fee+100) (total amount calculation) In our example: 100.3 * 3 / (0.3+100) = 3

Recommendations	The fix for this issue would be to use the correct calculation based on the amount including the fee. However, since a larger problem is the usage of the UniV2 pool itself, we recommend simply switching to a CL oracle. An additional small validation is necessary to ensure the correctness for such a fundamental change.
Comments / Resolution	Resolved.

Issue	Lack of reentrancy guard for extendLockDuration, transferLocks and mintNFTforLock
Severity	Low
Description	The aforementioned functions do not have a reentrancy guard. While the first two are not executing any external calls, they can still be called whenever one of the functions with a guard is executing an external call.
Recommendations	Consider implementing the reentrancy guard towards these functions.
Comments / Resolution	Resolved.

Issue	transferLocks allows transfer to self
Severity	Low
Description	The aforementioned function allows the _receiverAddress to be the same address as the current withdrawalAddress. While we were not able to detect any harm which can be done, we always aim to follow the following philosophy: “Most of the time, exploits happen due to arbitrary user inputs or users invoking functions which are not meant to be invoked by users, one can argue that a large user flexibility is a great seed for exploits. Therefore, at BailSec, we are of the opinion that codebases should never provide more user flexibility than necessary during the normal business logic.”
Recommendations	Consider if there is any specific use-case of invoking this function with the same _receiverAddress as the current withdrawalAddress (also in relation to this function being callable via the NFT flow). If there is no such use-case, consider preventing such a call explicitly.
Comments / Resolution	Resolved.

Issue	Hardcoded 0.3% fee can differ from real swap fee
Severity	Low
Description	Depending on which router is used (the router is changeable within the PriceEstimator), the 0.3% fee becomes incorrect. Many projects are using a fee which is lower than 0.3%.
Recommendations	Consider switching to the Chainlink oracle approach. An additional small validation is necessary to ensure the correctness for such a fundamental change.
Comments / Resolution	Resolved.

Issue	Users can create new positions with zero-value using splitLock
Severity	Low
Description	The splitLock function lacks a _splitAmount validation. This value can be set to zero, effectively allowing the caller to create locks with zero-amounts.
Recommendations	Consider validating _splitAmount accordingly.
Comments / Resolution	Resolved.

Issue	Fee transfer can DoS the following functions: lockToken, lockNFT and splitLock
Severity	Low
Description	During all three aforementioned functions, ETH is transferred to one or more addresses via a simple call. If one address has no fallback function or intentionally reverts upon ETH receipt, this will DoS these functions.
Recommendations	Consider wrapping ETH to WETH and transfer it out via WETH in such a scenario when success is false.
Comments / Resolution	Acknowledged.

Issue	Referrer parameter can be abused to sybil referral
Severity	Informational
Description	Currently, the referrer parameter is simply provided with the function, we assume that the frontend handles this based on a certain logic. The problem is, users can also provide their secondary wallets as referrals. Such an issue is present with all referral systems but contrary to this system, other referral systems usually cache the referrer for a user upon the first interaction.
Recommendations	Consider if it makes sense to re-write the referral system in such a manner that it is not changeable. A fix for this issue is not necessarily expected since with a high likelihood it is a design choice.
Comments / Resolution	Acknowledged.

Issue	_removeDepositsForWithdrawalAddress does not revert if ID was not found
Severity	Informational
Description	The aforementioned functions loops over all IDs a user owns. If the to be removed ID was found, it is removed from the array. The problem is, if the ID is not found, it will not revert.
Recommendations	Consider reverting in the scenario where the ID is not existent in the array.
Comments / Resolution	Acknowledged.