BAIL security

BAILSEC.IO

EMAIL : OFFICE@BAILSEC.IO

TWITTER : @BAILSECURITY

TELEGRAM : @HELLOATBAILSEC

FINAL REPORT:

Trustswap SwapToken

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 SwapToken
Website	trustswap.com
Type	ERC-20 TOKEN
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/trustswap/swap-contracts/blob/74baea17d0b47d3ce7d012c6453aa591c64905af/src/SwapToken.sol
Resolution 1	https://github.com/trustswap/swap-contracts/blob/5956331c709a11191e6e675625135cd89d43b777/src/SwapToken.sol

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High				
Medium				
Low	2	1		1
Informational	4	1		3
Configurational				
Governance	1			1
Quality assurance				
Total	7	2		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

SwapToken

The SwapToken contract is a simple ERC20 contract implementing the ERC20Burnable and ERC20Pausable features which allows users to burn tokens and governance to pause transfers. It serves as an implementation for a proxy contract. Moreover, a blacklist feature was implemented, which then prevents blacklisted addresses from transferring tokens, respectively.

It leverages OpenZeppelin's AccessControl library for the following purposes:

- DEFAULT_ADMIN_ROLE: Can assign roles to other addresses as well as withdraw tokens from the contract (presumably tokens sent there by accident), can blacklist/unblacklist addresses, change the _devWallet and mint tokens.

- PAUSER_ROLE: Can pause and unpause the contract, this will limit the transfer ability.

**It is notable to mention, if the contract is refactored a small nominal fee will apply to the resolution round because a simple diffchecker (https://www.diffchecker.com/) resolution will not be feasible.*

Issue	Governance Privilege: General
Severity	Governance
Description	The mint function allows addresses with the DEFAULT_MINTER_ROLE to mint tokens to any address. This is a highly sensitive governance privilege. Moreover, pause and blacklist can limit the user flexibility.
Recommendations	Consider incorporating a Gnosis Multisignature contract as DEFAULT_ADMIN and ensuring that the Gnosis participants are trusted entities.
Comments / Resolution	Acknowledged.

Issue	_devWallet is private
Severity	Low
Description	Certain variables which might be important for users to inspect should be made public instead of private in an effort to increase transparency. Average users might be unable to directly fetch storage slots using a script.
Recommendations	Consider marking the _devWallet as public.
Comments / Resolution	Resolved.

Issue	Outdated OpenZeppelin version
Severity	Low
Description	The contract incorporates the following dependencies: - Initializable - ContextUpgradeSafe - AccessControlUpgrade - ERC20BurnableUpgradeSafe - ERC20PausableUpgradeSafe These are dependencies for upgradeable contracts, however, they are depreciated as per OpenZeppelin: “ This package has been deprecated Author message: This package has been deprecated and replaced by @openzeppelin/contracts-upgradeable. “ Such deprecated libraries can potentially contain vulnerabilities which are fixed in later versions. A discrepancy between the import files and the inherited files exists, we highly assume that the inherited files are still part of the imported files but we do not have the exact commit for these OZ versions.
Recommendations	Consider switching to the newest OZ versions. Unfortunately, we do not know the exact commit of the used versions as the package.json only points to the following version: "@openzeppelin/contracts-ethereum-package": "^3.0.0" The node_modules folder is most probably only on the local machine of the developer.

	This issue should only be fixed once the contract is completely redeployed, since there is bad impact from the current architecture, we do not see it as necessary to introduce risk of potential storage collision by updating the implementation.
Comments / Resolution	Acknowledged.

Issue	Unstructured codebase
Severity	Informational
Description	The contract's codebase is highly unstructured: - State variables in between functions - Events in between functions - Modifier in between Such an unstructured codebase can confuse third-party inspectors.
Recommendations	Consider refactoring the codebase in a structured manner. This should then moreover be aligned with the correct initializers. This should be done combined with using the newest OZ versions, as the initializer selectors might have changed. Unfortunately, we do not have the exact commit of the used versions as the package.json only points to the following version: *"@openzeppelin/contracts-ethereum-package": "^3.0.0"* *This issue should only be fixed once the contract is completely redeployed, since there is bad impact from the current architecture, we do not see it as necessary to introduce risk of potential storage collision by updating the implementation.*

Comments / Resolution	Acknowledged.

Issue	Unconventional and redundant overriding practice for _approve
Severity	Informational

Description	The _approve function is changed as follows:
	
	function _approve(address owner, address spender, uint256 amount)
	internal
	override(ERC20UpgradeSafe)
	notBlacklisted(owner)
	notBlacklisted(spender)
	{
	super._approve(owner, spender, amount);
	}
	
	First of all, it is very unconventional to apply blacklist functionality on an approval. In fact, in around 100 custom tokens BailSec has audited, such a mechanism was never in place. Secondly, this practice is redundant as the _beforeTokenTransfer is sufficient for blacklisting.

Recommendations	Consider completely removing the _approve override.
Comments / Resolution	Resolved

Issue	Lack of safeTransfer usage for withdrawTokens
Severity	Informational
Description	The contract uses the standard transfer pattern for ERC20 transfers within the withdrawTokens function. This will malfunction for tokens that do not return a boolean on the transfer.
Recommendations	Consider using safeTransfer.
Comments / Resolution	Failed resolution, the transfer was marked as safeTransfer, however: a) The contract does not import OpenZeppelins safeERC20 library b) The require condition is still present, while safeTransfer does NOT return a boolean value. c) The “using” keyword is not implemented Consider simply reverting this change. It seems there was no attempt to compile the contract before the document was provided for the resolution round. In the future please try to compile your contracts first. Resolution 2: TBD

Issue	Redundant gap[] variable
Severity	Informational
Description	The gap[] variable is placed in contracts that are meant to be inherited for the simple reason to prevent storage collisions whenever adding new variables to the contract. This contract incorporates a gap[] as well, however, it is completely redundant as this contract is not meant to be inherited.
Recommendations	Consider removing this variable. *This issue should only be fixed once the contract is completely redeployed, since there is bad impact from the current architecture, we do not see it as necessary to introduce risk of potential storage collision by updating the implementation.*
Comments / Resolution	Acknowledged.