KENSHI

Kenshi Token
Security Assessment
November 28th, 2020

Project Details

Kenshi is a profit sharing token on the Binance Smart Chain network. It implements the BEP-20 token standard and features a friction-less reflection mechanism.

Name	Kenshi	**Blockchain**	BSC
Total Supply	10 trillion	**Symbol**	₭

Summary

We did not find any security vulnerabilities in the code of the smart contract. The code is fully tested and has a 100% unit test coverage. SafeMath is utilized to prevent overflows and underflows. Profit sharing functionality is trust-based and is handled by the contract owner.

Audit Details

The smart contract was statically analyzed and manually reviewed by 3 of our developers. Kenshi was also tasked to assess the team responsible for the “Kenshi Token” and verify the identity of its members.

The analysis is done on commit hash 77255...87bd3 on file contracts/Kenshi.sol located at https://github.com/kenshi-token/contracts. The contract was not deployed at the time of analysis.

Users of the token are recommended to check whether the deployed contract is verified and matches the above commit hash.

Security Vulnerabilities

We have tested and scanned the smart contract using multiple security tools, as well as analyzing the source code of the smart contract manually.

We could not find any security vulnerabilities using the automated security scanning tools. The manual analysis yield a similar result.

Mythril

Scanning the smart contract with Mythril shows no errors or warnings.

Slither

Scanning with Slither displays the following “informational” messages:

Dangerous comparisons

In function `_getTaxPercentage`, block time-stamp is used for comparison.

Kenshi.sol

```
874 /**
875 * @dev calculate tax percentage for `sender` based on purchase times.
876 */
877 function _getTaxPercentage(address sender) private view returns (uint8) {
878     if (_isExcluded(sender)) {
879         return _baseTax;
880     }
881     uint256 daysPassed = block.timestamp.sub(_purchaseTimes[sender]).div(
882         86400
883     );
884     if (daysPassed ⩾ 30) {
885         return _baseTax;
886     }
887     return _baseTax + _earlySaleFines[daysPassed];
888 }
```

Kenshi analysis concludes that the above usage of block time-stamp does not make the contract vulnerable to time-based attacks. The block time-stamp is not used for generating a pseudo-random number, cannot be abused and is used to check for a time-span much bigger than the block time of the blockchain network.

Incorrect Version of Solidity

The `Kenshi.sol` contract file necessitates a version too recent to be trusted.

Kenshi analysis concludes the correct choice was made in using the latest version of the solidity compiler, as indicated in the Solidity documentation:

> *When deploying contracts, you should use the latest released version of Solidity.*

Missing Inheritance

BEP20Token should inherit from IPresale.

Kenshi analysis concludes that the BEP20Token should not inherit from IPresale, as this interface is used to communicate with a “Presale” contract deployed elsewhere.

Manual Analysis

No security issues found while doing a manual review of the contract code.

Compiler Errors

The latest version of the Solidity compiler (v0.8.10 at the time of writing) is used for developing the token. Compiling the contract code with this version does not show any errors or compiler warnings.

Code Quality

The smart contract code follows the best practices in contract development. We did not find any misspellings in the contract code. All functions are commented and well-documented. Prettier is used for formatting the smart contract code. All linting and formatting tests pass.

We didn't find any unused code or variables. All function signatures are correct. There are no inconsistencies in the variable naming convention or the style of the code.

Special Addresses

The smart contract features a few specially treated addresses in its source code:

#	Address	Purpose
1	`0×dead`	Burn address
2	`Owner`	Owner
3	`Unset*`	DEX address
4	`Unset*`	DEX router address
5	`Unset*`	Treasury address

* Set after deployment.

DEX address, treasury address and the burn address are not included in the “circulating” supply, effectively excluding them from any rewards or reflections.

Addresses 2-5 are excluded from rewards, as well as being excluded from a max balance that is in place for other addresses.

Super-powers

The contract is in a locked state on deployment, rejecting any type of token trades. Only the owner is able to unlock the trades. However, once unlocked, the owner is not able to lock the contract again.

The owner is able to change the DEX address, the DEX router address and the treasury address at any time. The owner is able to set a tax on all transactions at anytime, with a maximum value of 15%. The owner is able to decide how much of the said tax goes to the treasury and how much of it is reflected.

The owner is able to change the “burn threshold” variable, effectively changing the maximum amount of tokens that can be burnt (set to 50% on deployment). The owner is able to recover BEP-20 tokens trapped in (or sent by mistake to) the contract address.

Team

Three of the team members have been identified and verified by the Kenshi team. The three said members have +30 years of cumulative experience in software development and engineering.

The identity of the founder is revealed to the public. Kenshi has verified the said person’s profile, experience and expertise and they appear to be the one they claim to be.

Disclaimer

Kenshi always takes a neutral stand-point when it comes to smart contract audits; we never recommend buying or avoiding a token or cryptocurrency in any of our audits.

Our reports are merely informational and should not be considered the absolute source of truth or trust.

You should always do your own research, examine a project from several points of view, and you should never make a decision solely based on an audit or report.