TAUNT TOKEN

SMART CONTRACT AUDIT

zokyo

November 15th, 2022 | v. 1.0

Security Audit Score

PASS

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

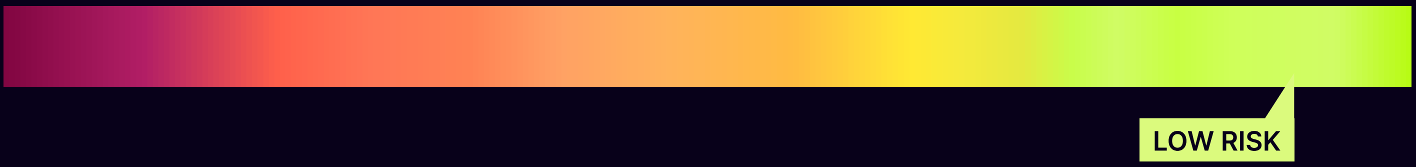
SCORE
91

TECHNICAL SUMMARY

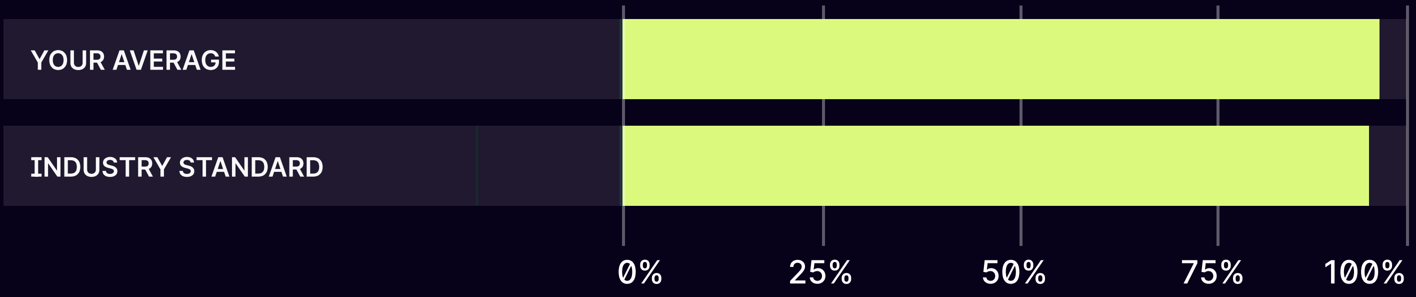
This document outlines the overall security of the Taunt Token smart contracts evaluated by the Zokyo Security team.

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

Contract Status



Testable Code



100% of the code is testable, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Taunt Token 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 Smart contract's source code was delivered as a separate .sol file.
The contract code is written in Solidity.

The code was provided as a separate .sol file sha256:
cdf6f155fd020493f3819ede10eeb20c7f828a46b586ae9b720728984948789b

Last audited version sha256:
ae3fa2736fa39552939c0d2d4e6d8d35e7fbbfac47dffbdf387459198f50232d

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

During the audit, Zokyo Security ensured 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, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most resent vulnerabilities;
- Meets best practices in code readability, etc.

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

01 Due diligence in assessing the overall code quality of the codebase.

02 Cross-comparison with other, similar smart contracts by industry leaders.

03 Testing contract logic against common and uncommon attack vectors.

04 Thorough manual review of the codebase line by line.

Executive Summary

Taunt smart contract represents an ERC20 token. It is fully compatible with the standard as it inherits the OpenZeppelin implementation. The token is expanded with the custom functionality:

- it is an upgradeable contract;
- it has an owner responsible for turning the bot protection mechanism on and off;
- the bot protection mechanism integrated into the transfer function.

In spite of the consultations with the Taunt team who verified the necessity of the upgradeability and bot protection mechanism, it needs to be mentioned that:

- the bot protection mechanism is provided through the 3rd party contract;
- upgradeabilty creates a controlled backdoor in the token contract.

Therefore, the token fails the standard check for the backdoors existence. However, despite the fact that the 3rd party contract is used during the transfer and its code is not available, Zokyo Security agrees with the Taunt team on bot protection security described in the appropriate issue. Thus, the contract passes the 3rd party dependancy check.

Due to the sensitive nature of the support contracts, the team of auditors worked with a single contract out of the repository, so it had no access to the tests and support contracts code. The Taunt team shared the deployment script that allowed to verify the correctness of the contracts deployment. Yet, due to the upgradeable nature of the contract, it is highly recommended for the Taunt team to enable transparent deployment process.

The security of the contract can be evaluated as high. Yet, it is an ERC20 token (which may be used in exchanges, listing, within other protocols, or during DAO votings, etc.), and since it has a controlled backdoor and additional dependancy on the controlled contract, the mark reflects those factors.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For the ease of navigation, document's sections are arranged from the most critical to the least critical. Issues are tagged as "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 further explanation from the Customer or remains disregarded by the Customer. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or unsafe behavior:

● **Critical**

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

● **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

HIGH-1 | VERIFIED

Custom unprotected initializer.

Being upgrabeable, the token utilizes the init() custom initializer and not the recommended, more common initialize()) with no restrictions for admins-only calls. This fact and the absence of deployment scripts makes it impossible to verify if the contract's proxy will be deployed and initialized properly. Thus, it is open to a set of possible exploits.

Recommendation:
Provide deployment scripts/procedures so that the team of auditors can verify the correctness of Proxy deployment and contract initialization.

Post-audit:
The Taunt team has provided the deployment script with the correct way of utilizing the custom initializer.

LOW-1 | FIXED

Incorrect Solidity version.

Currently, the contract uses Solidity ^0.8.1, which violates the standard security checklist since it is set as a float version, not fixed, and it is not the latest stable version (which is 0.8.17). Using any other Solidity version except for the latest stable one may lead to unexpected behavior since the latest version contains new features, updates to the optimizer, and security fixes. Also, since there is no deployment scripts provided, it is impossible to verify which version will be used during the deployment.
The issue is marked as low since it appears in thestandard security checklist and best practices.

Recommendation:
Use the latest stable Solidity version (0.8.17)

INFO-1 | VERIFIED

3rd-party contract used.

TauntToken.sol: function _transfer()
The function uses a 3rd party contract. It is named “bot protection”, though,
since it is out of the scope and its code is absent, it cannot be validated, thus it
leaves place for unexpected behavior and backdoors.
Also, since the address is not set during the initialization and is set later by the
admin, it creates a dangerous backdoor for the system.
Besides, there is no validation against zero address, which allows setting parameters for the transfer to fail every time.
The issue is marked as informational as it requires verification from the Taunt team as for the 3rd party contract. Yet, if no verification is provided the code
of the 3rd-party contract), the issue will influence the rating since unknown logic
may completely block the transfer.

Recommendation:
1) Provide the address of the deployed contract or the contract code in order to verify the 3rd-party contract.
2) Add validation to the _transfer() function for the case when the BP contract is not set (it equals 0 address).
3) The better way to ensure the security of the contract is to have the 3rd party set in the constructor. Thus, provide BP variable initialization in the constructor. If it is impossible (since contract may be created later), provide code of that contract.

From the client:
The Taunt team has verified that BP is the bot protection contract. Its code is not verified to make it as hard as possible for attackers to subvert the protection and attack the listing.
Since unverified production code is a sensitive matter, the Taunt team took precautions:
1. The BP code can only be controlled by the internal team that is in charge of every deployment.
2. The client that uses the BP code can always turn it on and off.
3. The Taunt team always instructs its clients to include a global flag that disables the BP permanently so that the code cannot be executed shortly after a successful launch. Even if the team does not disable the BP, the Taunt team code automatically disables itself after a couple of days. Therefore, investors only need to make sure that the feature was disabled once by the project managers after the launch.

INFO-2 | VERIFIED

Upgradeable token.

TauntToken.sol
The contract is upgradeable. Usually, such implementation has no impact on security (though it creates a controllable backdoor). But it is highly recommended to avoid using upgradeable contracts for the token since it influences the trust to the token from exchanges and other 3rd parties that may use the token. The issue is marked as informational since it is mostly connected to the implementation, but it needs to be mentioned in the report.

Recommendation:
Use non-upgradeable version of the contract OR verify that the Taunt team will stay with the upgradeable solution.

Post-audit:
The Taun team has verified the upgradeability of the token. Nevertheless, the team of auditors needs to mention that it creates a controllable backdoor in the token logic, which may be crucial for the token usage in exchanges, listings, and DeFi protocols.

INFO-3 | UNRESOLVED

_beforTokenTransfer() may be used.

TauntToken.sol
In order to decrease the code size and use recommended best practices, it is advised to use the _beforeTokenTransfer() function for additional checks (e.g., for the bot protection) instead of overloading the whole _transfer() method.
The issue is marked as informational as it refers to the best practices and does not influence the security of the project.

Recommendation:
Consider using the _beforTokenTransfer() method instead of _transfer() overloading.

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

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

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

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

```
Contract: TauntToken
  Initialization
    ✓ Should has correct name
    ✓ Should has correct name, symbol, minted token
    ✓ Should has correct minted token
  Standart ERC20 functions
    ✓ Admin should transfer tokens
    ✓ Admin should not transfer tokens from user1 to user2 if hasn`t allowance (48ms)
    ✓ Admin should transfer tokens from user1 to user2 if has allowance (52ms)
  Special TauntToken functions
    ✓ User (not owner) should not set forever bot protection
    ✓ Admin (owner) should set forever bot protection
    ✓ Admin (owner) should not set forever bot protection if it setted
    ✓ User (not owner) should not set bot protection
    ✓ Admin (owner) should set bot protection
    ✓ User (not owner) should not set bot protection contract
    ✓ Admin (owner) should set bot protection contract
    ✓ Admin (owner) should not set bot protection contract if it setted
    ✓ Overrided ._transfer() if bot protection enabled

15 passing (5s)
```

FILE	% STMTS	% BRANCH	% FUNCS
TauntToken.sol	100	100	100
All files	**100**	**100**	**100**

We are grateful for the opportunity to work with the Taunt Token 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 Taunt Token team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo