



Tornado Blast

Interim - Smart Contract Audit Report

DATE: 28 February 2024 | PREPARED BY: Entersoft Pte Ltd

Contents

Revision History & Version Control

| | |
|--|----|
| 1.0 Disclaimer | 4 |
| 2.0 Overview | 5 |
| 2.1 Project Overview | 6 |
| 2.2 Scope | 6 |
| 2.3 Project Summary | 6 |
| 2.4 Audit Summary | 6 |
| 2.5 Security Level References | 7 |
| 2.6 Vulnerability Summary | 7 |
| 4.1 Reliance on block.timestamp in uniV2Buy Function | 10 |
| 4.2 Reliance on block.timestamp in swapRouter01Params Function | 10 |
| 4.3 Use of Unsafe ERC20 Operations in getTokenReadyToBeSold Function | 11 |
| 4.4 Use of Unsafe ERC20 Operations in prepareWethForBuySwap Function | 11 |
| 4.5 Optimization of Inequalities in require() Statements in checkAmountOutMin() function | 12 |
| 4.6 Solidity pragma should be specific, not wide | 12 |
| 5.1 Code Review / Manual Analysis | 13 |
| 5.2 Tools Used for Audit | 13 |

Revision History & Version Control

| Version | Date | Author(s) | Description |
|---------|------------------|----------------------|----------------|
| 1.0 | 28 February 2024 | R. Dixit G. Singh | Interim report |

Entersoft was commissioned by Tornado Blast to perform Smart contract code review on their product. The review was conducted from 27 February 2024 to 28 February 2024, to ensure overall code quality, security, and correctness, and ensure that the code will work as intended.

The report is structured into two main sections:

- Executive Summary which provides a high-level overview of the audit findings.
- Technical Analysis which offers a detailed analysis of the smart contract code.

Please note that the analysis is static and entirely limited to the smart contract code. The information provided in this report should be used to understand the security and quality of the code, as well as its expected behavior.

Scope included:

- BaseBlastBotSwapAndFeesHandler.sol
- HandlerWithAdminEthFees.sol

Standards followed include:

- OWASP (partially, for instance, role validations, input validations, etc.)
- Solidity best security practices

1.0 Disclaimer

This is a limited audit report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to (i) smart contract best coding practices and issues in the framework and algorithms based on white paper, code, the details of which are set out in this report, (Smart Contract audit). To get a full view of our analysis, you must read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us based on what it says or does not say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Entersoft Australia and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (Entersoft) owe no duty of care towards you or any other person, nor does Entersoft make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and Entersoft hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose, and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Entersoft hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Entersoft, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the Smart contract is purely based on the smart contract code shared with us alone.

2.0 Overview

2.1 Project Overview

Entersoft has meticulously audited the smart contract project from 27th February 2024 to 28th February 2024, with a primary focus on Solidity code files integral to blockchain functionality, emphasizing vulnerabilities in associated gas claiming and Burn functionalities. The working of basic functionalities was also tested during the review.

2.2 Scope

The audit scope covers the Tornado Blast smart contracts available in the GitHub private repository. The audit focused on the checklist provided for the smart contract code audit.

1. BaseBlastBotSwapAndFeesHandler.sol
2. HandlerWithAdminEthFees.sol

2.3 Project Summary

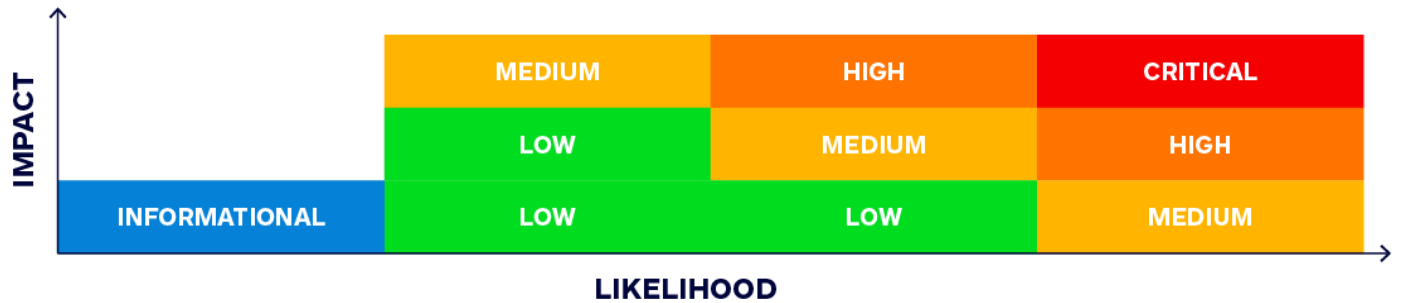
| Name | Verified | Audited | Vulnerabilities |
|---------------|----------|---------|-------------------------|
| Tornado Blast | Yes | Yes | Please review Section 4 |

2.4 Audit Summary

| Delivery Date | Method of Audit | Consultants Engaged |
|-----------------|---|---------------------|
| 28 January 2024 | Static Analysis: The static part of a smart contract audit refers to the process of reviewing the code of a smart contract to identify security vulnerabilities, coding errors, and adherence to best practices without executing the code and without interacting with it in a live environment. The smart contract is reviewed as is. In this we will cover logic vulnerabilities, dependency security flaws and more. | 3 |

2.5 Security Level References

Every vulnerability in this report was assigned a severity level from the following classification table:



2.6 Vulnerability Summary

| ● Critical | ● High | ● Medium | ● Low | ● Informational |
|------------|--------|----------|-------|-----------------|
| 0 | 0 | 0 | 4 | 2 |

3.0 Executive Summary

Entersoft has conducted a comprehensive technical audit of the Tornado Blast project through a comprehensive smart contract audit. The primary objective was to identify potential vulnerabilities and risks within the codebase, ensuring adherence to industry-leading standards while prioritizing security, reliability, and performance. Our focus was on prompt and efficient identification and resolution of vulnerabilities to enhance the overall robustness of the solidity smart contract.

Testing Methodology:

Our testing methodology in Solidity adhered to industry standards and best practices, integrating partially implemented OWASP and NIST SP 800 standards for encryption and signatures.

We have performed a detailed manual analysis, adherence to industry standards, and the use of a comprehensive toolset. Our approach ensured a thorough evaluation of Swap-related functionalities and specific Mint and Burn operations within the designated Solidity code files.

Findings and Security Posture:













Our primary focus was on Access Control Policies, Transaction Signature Validations, Reentrancy, Time Manipulation, Default Visibility, Outdated Compiler Version, Input Validation, Deprecated Solidity Functions, Shadowing State Variables, Presence of Unused Variables, Overflow and Underflow Conditions, Assets Integrity, Errors and Exception.

Importantly, our audit process intentionally avoided reliance solely on automated tools, emphasizing a more in-depth and nuanced approach to security analysis. Conducted from February 27, 2023, to February 28, 2024, our team diligently assessed and validated the security posture of the solidity smart contract, ultimately classifying it as "Secure," reflecting the absence of identified vulnerabilities and the robustness of the codebase against potential threats.

Result

The following table provides an overall summary of the findings and security posture of the smart contract code in scope.

-  **No Security vulnerabilities were identified**
-  **Security vulnerabilities were identified**

| # | Tornado Blast Audit Attack Vectors | Result |
|---------------------------------|------------------------------------|---|
| 1 | Access Control Policies |  |
| 2 | Transaction Signature Validations |  |
| 3 | Default Visibility |  |
| 4 | Address Validation |  |
| 5 | Reentrancy attacks |  |
| 6 | Presence of unused variables |  |
| 7 | Overflow and Underflow Conditions |  |
| 8 | Assets Integrity |  |
| 10 | Program Validation |  |
| 11 | DOS |  |
| 12 | Time Manipulation Vulnerability |  |
| 13 | ERC20 token issues |  |
| Overall Security Posture | | NOT SECURE |

4.0 Technical Analysis

The following results are the efforts of static analysis.

Note: The following values for “Result” mean:

- **PASS** indicates that there is no security risk.
- **FAIL** indicates that there is a security risk that needs to be remediated.
- **Informational** findings should be followed as a best practice, and they are not visible from the solidity smart contract.
- **Not Applicable** means the attack vector is Not applicable or Not available

4.1 Reliance on block.timestamp in uniV2Buy Function

| | |
|-------------------------------|---|
| Result | FAIL |
| Severity | Low |
| Description | The uniV2Buy function within the smart contract utilizes block.timestamp as the deadline parameter when making a transaction via Uniswap V2. This practice introduces a potential security vulnerability due to reliance on block.timestamp, which is susceptible to manipulation by miners. Adversaries can exploit this vulnerability by front-running transactions or manipulating block timestamps to bypass intended transaction deadlines, potentially leading to undesirable outcomes such as failed transactions or unexpected token purchases. |
| Location / Source File | BaseBlastBotSwapAndFeesHandler.sol |
| Observation | Reliance on block.timestamp in uniV2Buy function exposes vulnerability to transaction manipulation, risking failed transactions or unintended token purchases. |
| Remediation | Replace block.timestamp with a more secure and tamper-resistant time source, such as block.number or an external timestamp oracle, to mitigate potential exploits and ensure transaction integrity. |
| Reference | NA |

4.2 Reliance on block.timestamp in swapRouter01Params Function

| | |
|-------------------------------|---|
| Result | FAIL |
| Severity | Low |
| Description | The uniV2Buy function within the smart contract utilizes block.timestamp as the deadline parameter when making a transaction via Uniswap V2. This practice introduces a potential security vulnerability due to reliance on block.timestamp, which is susceptible to manipulation by miners. Adversaries can exploit this vulnerability by front-running transactions or manipulating block timestamps to bypass intended transaction deadlines, potentially leading to undesirable outcomes such as failed transactions or unexpected token purchases. |
| Location / Source File | BaseBlastBotSwapAndFeesHandler.sol |
| Observation | Reliance on block.timestamp in uniV2Buy function exposes vulnerability to transaction manipulation, risking failed transactions or unintended token purchases. |
| Remediation | Replace block.timestamp with a more secure and tamper-resistant time source, such as block.number or an external timestamp oracle, to mitigate potential exploits and ensure transaction integrity. |
| Reference | NA |

4.3 Use of Unsafe ERC20 Operations in getTokenReadyToBeSold Function

| | |
|-------------------------------|--|
| Result | FAIL |
| Severity | Low |
| Description | The smart contract BaseBlastBotSwapAndFeesHandler.sol at line 143 employs unsafe ERC20 operations, specifically the approve function, without incorporating additional safety measures. Direct usage of ERC20 functions without proper safety checks may result in unexpected behavior, such as unreliable return values. It is recommended to utilize established libraries like OpenZeppelin's SafeERC20 to mitigate these risks and ensure secure interactions with ERC20 tokens. |
| Location / Source File | BaseBlastBotSwapAndFeesHandler.sol |
| Observation | The use of unprotected ERC20 operations, like the approve function, in BaseBlastBotSwapAndFeesHandler.sol at line 143 raises concerns regarding potential vulnerabilities and unreliable return values due to lack of safety measures. |
| Remediation | Implement best practices for interacting with ERC20 tokens by utilizing reliable libraries like OpenZeppelin's SafeERC20. This ensures that token operations are conducted securely, with appropriate checks and safeguards in place to mitigate potential risks associated with ERC20 functions. |
| Reference | NA |

4.4 Use of Unsafe ERC20 Operations in prepareWethForBuySwap Function

| | |
|-------------------------------|--|
| Result | FAIL |
| Severity | Low |
| Description | The smart contract BaseBlastBotSwapAndFeesHandler.sol at line 143 employs unsafe ERC20 operations, specifically the approve function, without incorporating additional safety measures. Direct usage of ERC20 functions without proper safety checks may result in unexpected behavior, such as unreliable return values. It is recommended to utilize established libraries like OpenZeppelin's SafeERC20 to mitigate these risks and ensure secure interactions with ERC20 tokens. |
| Location / Source File | BaseBlastBotSwapAndFeesHandler.sol |
| Observation | The use of unprotected ERC20 operations, like the approve function, in BaseBlastBotSwapAndFeesHandler.sol at line 143 raises concerns regarding potential vulnerabilities and unreliable return values due to lack of safety measures. |
| Remediation | Implement best practices for interacting with ERC20 tokens by utilizing reliable libraries like OpenZeppelin's SafeERC20. This ensures that token operations are conducted securely, with appropriate checks and safeguards in place to mitigate potential risks associated with ERC20 functions. |
| Reference | NA |

4.5 Optimization of Inequalities in require() Statements in checkAmountOutMin() function

| Result | Informational |
|------------------------|---|
| Description | The smart contract contains a require() statement utilizing a non-strict inequality (>=) to compare amountOut against amountOutMin. In Solidity, non-strict inequalities are typically more expensive in terms of gas consumption compared to strict inequalities (>, <) due to additional bytecode operations required for evaluation. This inefficient usage of inequalities inside require() statements can lead to unnecessary gas costs, impacting the overall efficiency of contract execution. |
| Location / Source File | BaseBlastBotSwapAndFeesHandler.sol |
| Observation | The contract employs a non-strict inequality (>=) in the require() statement, which may lead to unnecessary gas costs due to additional bytecode operations. |
| Remediation | Optimize gas efficiency by utilizing strict inequalities (>, <) instead of non-strict inequalities (>=, <=) inside require() statements wherever possible. In the context of the provided code snippet, consider modifying the checkAmountOutMin function to use a strict inequality (>) if applicable to the contract's logic without compromising security requirements. |
| Reference | NA |

4.6 Solidity pragma should be specific, not wide

| Result | Informational |
|------------------------|---|
| Description | Consider using a specific version of Solidity in your contracts instead of a wide version. For example, instead of pragma solidity ^0.8.0;, use pragma solidity 0.8.0; |
| Location / Source File | BaseBlastBotSwapAndFeesHandler.sol |
| Observation | Contracts should use a specific version of Solidity in pragma directives to ensure deterministic compilation and minimize risks associated with unintended behavior or vulnerabilities stemming from compiler upgrades. |
| Remediation | Specify an exact version of Solidity in the pragma directive to ensure deterministic compilation and minimize the risk of unintended behavior or vulnerabilities resulting from compiler upgrades. |
| Reference | NA |

5.0 Auditing Approach and Methodologies Applied

The solidity smart contract was audited in a comprehensive approach to ensure the highest level of security and reliability. Careful attention was given to the following key areas to ensure the overall quality of code:

- **Code quality and structure:** We conducted a detailed review of the codebase to identify any potential issues related to code structure, readability, and maintainability. This included analyzing the overall architecture of the solidity smart contract and reviewing the code to ensure it follows best practices and coding standards.
- **Security vulnerabilities:** Our team used manual techniques to identify any potential security vulnerabilities that could be exploited by attackers. This involved a thorough analysis of the code to identify any potential weaknesses, such as buffer overflows, injection vulnerabilities, Signatures, and deprecated functions.
- **Documentation and comments:** Our team reviewed the code documentation and comments to ensure they accurately describe the code's intended behavior and logic. This helps developers to better understand the codebase and make modifications without introducing new issues.
- **Compliance with best practices:** We checked that the code follows best practices and coding standards that are recommended by the solidity community and industry experts. This ensures that the solidity smart contract is secure, reliable, and efficient.

Our audit team followed OWASP and Ethereum(Solidity) community security guidelines for this audit. As a result, we were able to identify potential issues and provide recommendations to improve the smart contract security and performance.

Throughout the audit of the smart contracts, our team placed great emphasis on ensuring the overall quality of the code and the use of industry best practices. We meticulously reviewed the codebase to ensure that it was thoroughly documented and that all comments and logic aligned with the intended behavior. Our approach to the audit was comprehensive, methodical, and aimed at ensuring that the smart contract was secure, reliable, and optimized for performance.

5.1 Code Review / Manual Analysis

Our team conducted a manual analysis of the Solidity smart contracts to identify new vulnerabilities or to verify vulnerabilities found during static and manual analysis. We carefully analyzed every line of code and made sure that all instructions provided during the onboarding phase were followed. Through our manual analysis, we were able to identify potential vulnerabilities that may have been missed by automated tools and ensure that the smart contract was secure and reliable.

5.2 Tools Used for Audit

In the course of our audit, we leveraged a suite of tools to bolster the security and performance of our program. While our team drew on their expertise and industry best practices, we also integrated various tools into our development environment. Noteworthy among them are Adreyn, Solidity Scan. This holistic approach ensures a thorough analysis, uncovering potential issues that automated tools alone might overlook. Entersoft takes pride in utilizing these tools, which significantly contribute to the quality, security, and maintainability of our codebase.

6.0 Limitations on Disclosure and Use of this Report

This report contains information concerning potential details of Tornado Blast Project and methods for exploiting them. Entersoft recommends that special precautions be taken to protect the confidentiality of both this document and the information contained herein. Security Assessment is an uncertain process, based on past experiences, currently available information, and known threats. All information security systems, which by their nature are dependent on human beings, are vulnerable to some degree. Therefore, while Entersoft considers the major security vulnerabilities of the analyzed systems to have been identified, there can be no assurance that any exercise of this nature will identify all possible vulnerabilities or propose exhaustive and operationally viable recommendations to mitigate those exposures. In addition, the analysis set forth herein is based on the technologies and known threats as of the date of this report. As technologies and risks change over time, the vulnerabilities associated with the operation of the Tornado Blast solidity smart contract described in this report, as well as the actions necessary to reduce the exposure to such vulnerabilities will also change. Entersoft makes no undertaking to supplement or update this report based on changed circumstances or facts of which Entersoft becomes aware after the date hereof, absent a specific written agreement to perform the supplemental or updated analysis. This report may recommend that Entersoft use certain software or hardware products manufactured or maintained by other vendors. Entersoft bases these recommendations upon its prior experience with the capabilities of those products. Nonetheless, Entersoft does not and cannot warrant that a particular product will work as advertised by the vendor, nor that it will operate in the manner intended. This report was prepared by Entersoft for the exclusive benefit of Tornado Blast and is proprietary information. The Non-Disclosure Agreement (NDA) in effect between Entersoft and Tornado Blast governs the disclosure of this report to all other parties including product vendors and suppliers.