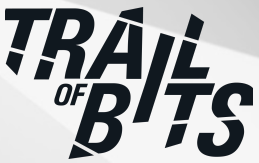

Easy Crypto NZDD

Security Assessment (Summary Report)

August 31, 2023

Prepared for:
Easy Crypto

Prepared by: **Nat Chin**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

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.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Executive Summary

Engagement Overview

Easy Crypto engaged Trail of Bits to review the security of its NZDD token implementation. The USDC-like stablecoin implements an ERC-20 token, with controlled minting and burning, blacklisting limitations, and pausing functionality.

One consultant conducted the review from July 26 to July 28, 2023, for a total of three engineer-days of effort. Our testing efforts focused on best smart contract practices related to the NZDD token implementation. With full access to source code and documentation, we performed static and dynamic testing of the codebase, using automated and manual processes. We did not review any off-chain components or governance multisignatures involved in the token integration.

Observations and Impact

For the most part, the smart contracts follow best practices. They are small and concise, with a focus on permissioned, role-based access.

Two issues discovered during this review pertain to the use of outdated, unused, and vulnerable dependencies. While these dependencies may not affect the codebase directly, deployment scripts and secrets stored in the process environment can potentially be disclosed due to malicious or vulnerable dependencies.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that Easy Crypto take the following steps:

- **Remediate the findings disclosed during this review.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.
- **Update technical and governance specifications to reflect token behavior.** The provided specification identifies functions that do not correspond to ones in the token contract and should be updated to reflect the current codebase.
- **Update vulnerable and outdated dependencies and remove unused ones.** New releases in software are usually created to fix vulnerabilities and should be adopted by projects to ensure dependencies cannot negatively impact the codebase.
- **Document the responsibilities of the per–access control multisig.**

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
Medium	1
Informational	3

CATEGORY BREAKDOWN

Category	*Count*
Data Validation	2
Patching	2

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

Anne Marie Barry, Project Manager
annemarie.barry@trailofbits.com

The following engineer was associated with this project:

Nat Chin, Consultant
natalie.chin@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
July 24, 2023	Pre-project kickoff call
July 28, 2023	Delivery of report draft
July 31, 2023	Delivery of updated report draft
August 1, 2023	Report readout meeting
August 9, 2023	Delivery of summary report
August 31, 2023	Delivery of summary report with Fix Review

Project Goals

The engagement was scoped to provide a security assessment of the Easy Crypto NZDD token. Specifically, we sought to answer the following non-exhaustive list of questions:

- Does the code follow smart contract best practices?
- Is it possible to steal funds?
- Is it possible to bypass access controls set in the contract?
- Are there any denial-of-service attack vectors?
- Can user funds become trapped in the system?
- Are all expected functions the inverse of one another?
- Are there any deviations between the documentation and the protocol implementations?

Project Targets

The engagement involved a review and testing of the following target.

NZDD token

Repository	https://github.com/ec-systems/nzdd
Version	`0ae2793487f15dc1ba25de0681b2fce2cafb318a`
Type	Solidity
Platform	Ethereum

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- **ERC-20 token implementation.** The NZDD token contract is the entry point for users to interact with the NZDD token.
    - We used our static analysis tool, Slither, on the codebase to check for violations of best practices in smart contracts.
    - We reviewed the overridden functions in the contract to ensure that they would not result in unexpected behavior for users interacting with this token, with respect to the expected behavior of ERC-20 tokens.
    - We reviewed the data validation added to the minting and burning functionality in the codebase. We checked the access controls behind only the minter being able to invoke these two functions.
    - We reviewed the initial granting of roles to the minter, admin, and blacklister in the system.
    - We reviewed the `blacklist` and `unblacklist` functions in the system to ensure that they checked and changed state correctly.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. During this project, we were unable to perform comprehensive testing of the following system elements, which may warrant further review:

- Off-chain monitoring components
- Multisignature smart contract wallets used for access controls
- Reassignment or revocation of role-based access controls
- Front-running implications on the codebase

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	The NZDD codebase does not contain any instances of adjusted or changed arithmetic. According to system documentation, minter roles are bound by an allowance of tokens. However, this is not implemented in the NZDD contract and was therefore considered out of scope during this review.	**Not Applicable**
Auditing	All relevant functions in the contracts emit events when state changes are made. We recommend documenting a monitoring plan to detect inconspicuous behavior and creating an incident response plan to ensure that the plan of action is clear if a vulnerability disclosure is received in the future.	**Satisfactory**
Authentication / Access Controls	The access controls in the token contract are kept to a minimum. According to governance specifications, all these access controls are N-of-M multisig contracts, whose members are controlled by the admin. The privileges and roles in the system are clearly tested, with special scrutiny on the blacklist functionality. However, the number of expected addresses for these roles remains unclear because the documentation identifies multiple users being able to mint funds in the system and control a single address.	**Further Investigation Required**
Complexity Management	The contracts keep functionality to a minimum and extensively use the OpenZeppelin contract library. All inputs are validated for zero checks and zero arguments. However, checks in certain functions are duplicated in custom errors and OpenZeppelin-provided `require` statements. We recommend further documenting the expectations of all data validation in the system.	**Moderate**

Cryptography and Key Management	The contracts do not use cryptography or key management.	**Not Applicable**
Decentralization	The NZDD stablecoin implementation is not intended to be decentralized and relies heavily on permissioned roles to mint and burn funds, pause the system, blacklist addresses, and provide upgradeability for the system.	**Not Applicable**
Documentation	Documentation on the governance specifications and token specifications were provided but were out of date. This made expected behavior in the system unclear. We recommend ensuring that terminology used in the documentation matches that used in the code and/or specified in NatSpec comments in the code.	**Weak**
Front-Running Resistance	Front-running implications were not investigated during this review.	**Not Considered**
Low-Level Manipulation	The codebase does not use low-level calls or assembly.	**Not Applicable**
Testing and Verification	The contracts in the codebase reach 100% statement and branch coverage. We recommend expanding property-based invariant testing to further check role-based access controls and token transfers. We document invariants that can be tested in appendix E.	**Moderate**

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Use of dependencies with vulnerabilities	Patching	**Medium**
2	Reliance on outdated and unused dependencies	Patching	**Informational**
3	Users can be blacklisted or unblacklisted when system is paused	Data Validation	**Informational**
4	Minting and burning are not exact inverses	Data Validation	**Informational**

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Front-Running Resistance	The system's resistance to front-running attacks
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.

Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. System Level Assumptions

The scope of our three engineer-days of review was limited solely to the NZDD token contract. Below, we identify assumptions that were made during the review. We recommend further efforts be invested in these assumptions and the holding of these invariants.

Role-Based Access Controls

- Minters decide prior to initiating a transaction whether the newly minted amount is within their allowance.
    - The on-chain code does not contain data validation checks on allowance or validate the amounts to be minted.
- Blacklisters should not add the zero address or the burn address to the blacklist.
    - Adding the zero address or equivalent burn address to the blacklist would prevent tokens from being burnt, as transfers to the blacklisted address would fail. However, this would be reversible by unblacklisting the relevant address.
- The token contract supports only a single address for each of the following roles: master admin, default admin, minter, and blacklister.
    - The governance specification and relevant documentation identify scenarios where these roles could be configured on-chain, resulting in multiple minters, which the contracts do not currently support.
- The token contract allows the relevant master admin, default admin, minter, and blacklister addresses to be set only once, at construction.
    - The contract code can be updated to support the granting and revoking of additional addresses for these roles, but that is not a part of the current implementation.
- A minter can burn whatever amount of tokens they hold. The on-chain contracts contain no other logic to limit this amount.
    - If limits exist, they are not included in the NZDD token contract.

D. Deviations in Documentation and Code

Throughout this review, we cross-referenced the documentation and code to identify potential vulnerabilities. We recommend going through each of these deviations and either adjusting the documentation or the code to account for these cases.

Minter

The current implementation of the minter role in the NZDD contract consists of a single minter address, which can be updated once, when the contract is initialized by the admin. This role can increase the balance of NZDD tokens in supply. The description of the minter outlines an existing minter allowance, as well as multiple minters, which are not implemented on the on-chain contracts:

- **Minting**: This function **mint(address _to, uint256 _amount)** allows authorised actors (**minters**) to create new fiat tokens, increasing the total supply in circulation. It is capped at the **minterAllowance** of the caller address. The **masterMinter** configures the addresses authorised as **minters** as well as their **minterAllowance**. masterMinters can also add/remove minters as well as reset minterAllowance (increase or decrease).

Burning

The current implementation of the burner role in the contract allows the address to be updated only once, when initialized. This role is allowed to burn only its own tokens. While the burning of tokens by a blacklisted minter is prevented, the NZDD contract does not have the concept of paused addresses:

- **Burning**: Similar to minting, the **burn(uint256 _amount)** method enables authorised actors (**minters**) to remove fiat tokens from circulation, effectively decreasing the total supply. Each minter can only **burn** its own tokens. Minter blacklisted or paused addresses cannot burn.

Pauser

The current implementation of the pauser role in this contract allows the address to be updated once, when initialized. The pauser does not have a separate role—rather, the `pause` and `unpause` functions are protected by the `DEFAULT_ADMIN_ROLE`. As a result, the owner cannot update the contract pauser address, as shown below:

- **Pausing**: The pausing function **pause()** allows authorised actors (**pauser**) to temporarily halt certain functions within the contract, such as all transfers, minting, burning, and adding minter, in case of emergency or security concerns. An **unpause()** method can be called to reset contract functions. Only the **owner** can update the contract **pauser** address.

E. Property-Based System Invariants

We recommend Easy Crypto implement property-based testing using Echidna. To start, we identified system properties that can be tested on the NZDD token contract:

Upgrading

- ☐ Only the `masterAdmin` can update the contract and implementation.
- ☐ The `initialize` function can be called only once.

Pausing/Unpausing

- ☐ The `DEFAULT_ADMIN_ROLE` can pause the system.
- ☐ The `DEFAULT_ADMIN_ROLE` can unpause the system.

Blacklisting

- ☐ Once blacklisted, an address can be unblacklisted only by calling the `unblacklist` function.
- ☐ Only the `NZDD_BLACKLISTER_ROLE` can blacklist an address.
- ☐ Only the `NZDD_BLACKLISTER_ROLE` can unblacklist an address.
- ☐ Blacklisting an already blacklisted address should revert.
- ☐ Unblacklisting an unblacklisted address should revert.
- ☐ Blacklisting an address when NZDD is paused should revert.
- ☐ Unblacklisting an address when NZDD is paused should revert.

Minting

- ☐ Only an address with the `NZDD_MINTER_ROLE` can mint funds.
- ☐ Minting funds to `address(0)` should revert.
- ☐ Minting zero tokens should revert.
- ☐ Minting when the system is paused should revert.
- ☐ Minting funds when NZDD is paused should revert.

Burning

- ☐ Only an address with the `NZDD_MINTER_ROLE` can burn funds.
- ☐ The `NZDD_MINTER_ROLE` can burn only their own token balance.
- ☐ Burning funds when NZDD is paused should revert.

Transferring

- ☐ Transferring should revert when the system is paused.
- ☐ Sending from a blacklisted address should revert.
- ☐ Sending to a blacklisted address should revert.
- ☐ Transferring between non-blacklisted addresses should succeed.
- ☐ Transferring between addresses when NZDD is paused should revert.

F. Code Quality Recommendations

In this appendix, we identify non-security–related code improvements or code changes we observed during our review.

- **Reconsider Solidity compiler optimizations being enabled.** We recommend further analyzing the compiler optimization flag and determining whether optimizations are needed for this code to be deployed and run. The Solidity optimizer occasionally has bug reports raised due to the complexity of the code and should be avoided, if possible.

- **Clarify the use of custom errors over reverts.** The codebase currently reimplements conditions that are already checked in `require` statements. For example, the `mint` function check against a nonzero address is already done in a higher level `_mint` function but is done again through custom errors in the NZDD token contract. This redundant code may result in additional gas costs.

- **The `NZDD_MINTER_ROLE` comment in the code should specify that addresses with this role can mint and burn funds.** The current notice tag on the hash mentions only minting funds.

G. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

On August 30, 2023, Trail of Bits reviewed the fixes and mitigations implemented by the Easy Crypto team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the four issues described in this report, Easy Crypto has resolved three issues and has not resolved the remaining issue. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Use of dependencies with vulnerabilities	Resolved
2	Reliance on outdated and unused dependencies	Resolved
3	Users can be blacklisted or unblacklisted when system is paused	Unresolved
4	Minting and burning are not exact inverses	Resolved

Detailed Fix Review Results

TOB-NZDD-1: Use of dependencies with vulnerabilities
Resolved in PR #51. Vulnerable dependencies related to the in-scope smart contracts were removed or updated. The output of `yarn audit` still yields additional results due to dependencies in other areas of the codebase that were not considered in scope. We recommend investing further efforts to pruning and clearing vulnerable dependencies in these other areas of the repository. Additionally, consider maintaining separate project dependencies for different components so that a single vulnerability does not affect all pieces of the code.

TOB-NZDD-2: Reliance on outdated and unused dependencies
Resolved in PR #51. The unused packages (Gnosis Safe, Gnosis deployments, and OpenZeppelin contracts) were removed from the `package.json`. While the `@openzeppelin/contracts^4.8.1` dependency has been removed from the codebase, the `yarn.lock` file still contains remnants of an older version (3.4.1) of OpenZeppelin contracts, which is why the `yarn audit` command continues to raise issues related to the dependency.

```
"@openzeppelin/contracts@3.4.1-solc-0.7-2":
  version "3.4.1-solc-0.7-2"
  resolved
"https://registry.yarnpkg.com/@openzeppelin/contracts/-/contracts-4.7.0.tgz#3092d70e
a60e3d1835466266b1d68ad47035a2d5"
  integrity
sha512-52Qb+A1DdOss8QvJrijYYPSf32GUg2pGaG/yCxtaA3cu4jduouTdg4XZSMLW9op54m1jH7J8hoajh
HKOPsoJFw==
```

Figure G.1: `@openzeppelin/contracts@3.4.1` version in `yarn.lock`

This is due to code in the `uniswap-widget`, which was not considered in scope for this review.

```
└─┬ ec-uniswap-widget@1.0.0 -> ./apps/uniswap-widget
  └─┬ @uniswap/widgets@2.22.0
    ├─┬ @uniswap/router-sdk@1.6.0
    │ └─┬ @uniswap/swap-router-contracts@1.1.0
    │   ├── @openzeppelin/contracts@3.4.1-solc-0.7-2
    │   └─┬ @uniswap/v3-periphery@1.3.0
    │     └── @openzeppelin/contracts@3.4.1-solc-0.7-2 deduped
    ├─┬ @uniswap/smart-order-router@2.10.2
    │ └─┬ @uniswap/swap-router-contracts@1.3.0
    │   ├── @openzeppelin/contracts@3.4.2-solc-0.7
    │   └─┬ @uniswap/v3-periphery@1.4.1
    │     └── @openzeppelin/contracts@3.4.2-solc-0.7 deduped
    ├─┬ @uniswap/universal-router-sdk@1.5.7
    │ └─┬ @uniswap/universal-router@1.4.3
    │   └── @openzeppelin/contracts@4.7.0
```

```
└─┬ @uniswap/v3-sdk@3.10.0
  ├─┬ @uniswap/v3-periphery@1.4.3
  │ └── @openzeppelin/contracts@3.4.2-solc-0.7
  └─┬ @uniswap/v3-staker@1.0.0
    └── @openzeppelin/contracts@3.4.1-solc-0.7-2 deduped
```

Figure G.2: Output of `npm ls @openzeppelin/contracts`

TOB-NZDD-3: User can be blacklisted/unblacklisted when system is paused
Unresolved. The client states:

> *We have concluded that the exploit scenario listed [...] is not really an exploit or damaging behaviour to the system. Pragmatically, we think that in this scenario - Bob would always know Alice has paused transfers. Pausing transfers on an ERC-20 contract (especially one of large value) would be a large event - and any Easy Crypto/NZDD actor executing these write methods would be in the know about whether or not the contract has been paused.*
>
> *To also confirm - the USDC Contract that this contracts behaviour is largely modelled after - has both pausable and blacklist functionality - and they do not stop blacklisting when the contract is paused.*
>
> *Due to these reasons - no code changes have been made in relation to this issue.*

We recommend documenting this expected behavior in system documentation and in the codebase to ensure clarity in what functions should be activated or deactivated in case the system is paused.

TOB-NZDD-4: Minting and burning are not exact inverses
Resolved in PR #51. A zero check was added to the `burn` function, which throws an `EmptyParameter` error if the `_amount` is zero.

H. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Further Investigation Required	The status of the issue requires additional effort to determine whether it has been resolved.
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.