SMART CONTRACT AUDIT REPORT

for

EVRY Token

Prepared By: Yiqun Chen

PeckShield

October 5, 2021

Document Properties

Client	Evrynet Finance
Title	Smart Contract Audit Report
Target	EVRY Token
Version	1.0
Author	Shulin Bie
Auditors	Shulin Bie, Xuxian Jiang
Reviewed by	Yiqun Chen
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	October 5, 2021	Shulin Bie	Final Release
1.0-rc	October 3, 2021	Shulin Bie	Release Candidate

Contact

For more information about this document and its contents, please contact PeckShield Inc.

Name	Yiqun Chen
Phone	+86 183 5897 7782
Email	contact@peckshield.com

Contents

1 Introduction 4

1.1 About EVRY Token 4

1.2 About PeckShield 5

1.3 Methodology 5

1.4 Disclaimer 7

2 Findings 8

2.1 Summary 8

2.2 Key Findings 8

3 ERC20 Compliance Checks 9

4 Conclusion 12

References 13

1 Introduction

Given the opportunity to review the design document and related smart contract source code of the EVRY Token, we outline in the report our systematic method to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistency between smart contract code and the documentation, and provide additional suggestions or recommendations for improvement. This document outlines our audit results.

1.1 About EVRY Token

Evrynet is an intelligent financial services platform that enables developers and businesses to build an unlimited number of centralized/decentralized finance (CeDeFi) applications. It is interoperable with many of the world's leading blockchains. In particular, the protocol token EVRY Token is widely used in the Evrynet ecosystem and is designed to enable a variety of critical functions across the network.

The basic information of EVRY Token is as follows:

Table 1.1: Basic Information of EVRY Token

Item	Description
Target	EVRY Token
Type	Smart Contract
Language	Solidity
Audit Method	Whitebox
Latest Audit Report	October 5, 2021

In the following, we show the Git repositories of reviewed files and the commit hash values used in this audit.

- https://gitlab.com/Evrynet/evry_generation/token_generation.git (d50af32)

1.2 About PeckShield

PeckShield Inc. [2] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of the current blockchain ecosystem by offering top-notch, industry-leading services and products (including the service of smart contract auditing). We are reachable at Telegram (https://t.me/peckshield), Twitter (http://twitter.com/peckshield), or Email (contact@peckshield.com).

1.3 Methodology

To standardize the evaluation, we define the following terminology based on the OWASP Risk Rating Methodology [1]:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk;

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

Table 1.2: Vulnerability Severity Classification

		High	*Medium*	*Low*
Impact	*High*	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		Likelihood		

We perform the audit according to the following procedures:

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.

- ERC20 Compliance Checks: We then manually check whether the implementation logic of the audited smart contract(s) follows the standard ERC20 specification and other best practices.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

Table 1.3: The Full Audit Checklist

Category	Checklist Items
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead of Transfer
	Costly Loop
	(Unsafe) Use of Untrusted Libraries
	(Unsafe) Use of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
	Approve / TransferFrom Race Condition
ERC20 Compliance Checks	Compliance Checks (Section 3)
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

To evaluate the risk, we go through a checklist of items, with each being labeled with a corresponding severity category. For each checklist item, if our tool does not identify any issue, the contract is considered safe regarding that item. For any discovered issues, we might further deploy contracts on our private testnet and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of checklist items is shown in Table 1.3.

1.4 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as investment advice.

2 Findings

2.1 Summary

Here is a summary of our analysis after examining the EVRY Token contract. During the first phase of our audit, we study the smart contract source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool. We further manually review business logics, examine system operations, and place ERC20-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Moreover, we explicitly evaluate whether the given contracts follow the standard ERC20 specification and other known best practices, and validate its compatibility with other similar ERC20 tokens and current DeFi protocols. The detailed ERC20 compliance checks are reported in Section 3. Overall, the token contract is well-implemented with the known best practices properly applied. And it is ready for deployment without any issue that needs to be addressed.

2.2 Key Findings

Overall, no ERC20 compliance issues were found, and our detailed checklist can be found in Section 3. From another perspective, we would like to emphasize that it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms need to kick in at the very moment when the contracts are being deployed in mainnet. Please refer to Section 3 for our detailed compliance checks.

3 | ERC20 Compliance Checks

The ERC20 specification defines a list of API functions (and relevant events) that each token contract is expected to implement (and emit). Any failure to meet these requirements would mean that the token contract cannot be considered to be ERC20-compliant. Naturally, as the first step of our audit, we examine the list of API functions defined by the ERC20 specification and validate whether there exists any inconsistency or incompatibility in the implementation or the inherent business logic of the audited contract(s).

Table 3.1: Basic `View-Only` Functions Defined in The ERC20 Specification

Check Item	Description	Pass
name()	Is declared as a public view function	✓
	Returns a string, for example "Tether USD"	✓
symbol()	Is declared as a public view function	✓
	Returns the symbol by which the token contract should be known, for example "USDT". It is usually 3 or 4 characters in length	✓
decimals()	Is declared as a public view function	✓
	Returns decimals, which refers to how divisible a token can be, from 0 (not at all divisible) to 18 (pretty much continuous) and even higher if required	✓
totalSupply()	Is declared as a public view function	✓
	Returns the number of total supplied tokens, including the total minted tokens (minus the total burned tokens) ever since the deployment	✓
balanceOf()	Is declared as a public view function	✓
	Anyone can query any address' balance, as all data on the blockchain is public	✓
allowance()	Is declared as a public view function	✓
	Returns the amount which the spender is still allowed to withdraw from the owner	✓

Our analysis shows that there are no ERC20 inconsistency or incompatibility issues found in the audited EVRY Token. In the surrounding two tables, we outline the respective list of basic `view-only` functions (Table 3.1) and key `state-changing` functions (Table 3.2) according to the widely-adopted

ERC20 specification.

Table 3.2: Key `State-Changing` Functions Defined in The ERC20 Specification

Check Item	Description	Pass
transfer()	Is declared as a public function	✓
	Returns a boolean value which accurately reflects the token transfer status	✓
	Reverts if the caller does not have enough tokens to spend	✓
	Allows zero amount transfers	✓
	Emits Transfer() event when tokens are transferred successfully (include 0 amount transfers)	✓
	Reverts while transferring to zero address	✓
transferFrom()	Is declared as a public function	✓
	Returns a boolean value which accurately reflects the token transfer status	✓
	Reverts if the spender does not have enough token allowances to spend	✓
	Updates the spender's token allowances when tokens are transferred successfully	✓
	Reverts if the from address does not have enough tokens to spend	✓
	Allows zero amount transfers	✓
	Emits Transfer() event when tokens are transferred successfully (include 0 amount transfers)	✓
	Reverts while transferring from zero address	✓
	Reverts while transferring to zero address	✓
approve()	Is declared as a public function	✓
	Returns a boolean value which accurately reflects the token approval status	✓
	Emits Approval() event when tokens are approved successfully	✓
	Reverts while approving to zero address	✓
Transfer() event	Is emitted when tokens are transferred, including zero value transfers	✓
	Is emitted with the from address set to $address(0x0)$ when new tokens are generated	✓
Approve() event	Is emitted on any successful call to approve()	✓

In addition, we perform a further examination on certain features that are permitted by the ERC20 specification or even further extended in follow-up refinements and enhancements (e.g., ERC777), but not required for implementation. These features are generally helpful, but may also impact or bring certain incompatibility with current DeFi protocols. Therefore, we consider it is important to highlight them as well. This list is shown in Table 3.3.

Table 3.3: Additional `Opt-in` Features Examined in Our Audit

Feature	Description	Opt-in
Deflationary	Part of the tokens are burned or transferred as a fee while on transfer()/transferFrom() calls	—
Rebasing	The balanceOf() function returns a re-based balance instead of the actual stored amount of tokens owned by the specific address	—
Pausable	The token contract allows the owner or privileged users to pause the token transfers and other operations	—
Blacklistable	The token contract allows the owner or privileged users to blacklist a specific address such that token transfers and other operations related to that address are prohibited	—
Mintable	The token contract allows the owner or privileged users to mint tokens to a specific address	—
Burnable	The token contract allows the owner or privileged users to burn tokens of a specific address	✓
Hookable	The token contract allows the sender/recipient to be notified while sending/receiving tokens	—
Permittable	The token contract allows for unambiguous expression of an intended spender with the specified allowance in an off-chain manner (e.g., a permit() call to properly set up the allowance with a signature).	—

4 Conclusion

In this audit, we have examined the `EVRY Token` design and implementation. We have accordingly checked all aspects related to the ERC20 standard compatibility and other known ERC20 pitfalls/vulnerabilities, and no issues were found in these areas. We have also proceeded to examine other areas such as coding practices and business logic. Overall, we believe there are no issues that need to bring up and the current implementation is ready for deployment. Meanwhile, as disclaimed in Section 1.4, we appreciate any constructive feedbacks or suggestions about our findings, procedures, audit scope, etc.

References

[1] OWASP. OWASP Risk Rating Methodology. https://www.owasp.org/www-community/OWASP_Risk_Rating_Methodology.

[2] PeckShield. PeckShield Inc. https://www.peckshield.com.