SMART CONTRACT AUDIT REPORT

for

Dtravel TRVL

Prepared By: Yiqun Chen

PeckShield

November 4, 2021

Document Properties

Client	Dtravel
Title	Smart Contract Audit Report
Target	Dtravel TRVL
Version	1.0
Author	Jing Wang
Auditors	Jing Wang, Xuxian Jiang
Reviewed by	Yiqun Chen
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author	Description
1.0	November 4, 2021	Jing Wang	Final Release

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

Given the opportunity to review the design document and related source code of the **Dtravel TRVL** token contract, 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. Our results show that the given version of the smart contract exhibits no ERC20 compliance issues or security concerns. This document outlines our audit results.

1.1 About Dtravel TRVL

`Dtravel` is a decentralized platform for the home-sharing economy that facilitates accommodation discovery, booking, and payments. `Dtravel` users can make payments with both fiat currencies and popular cryptocurrencies, including `TRVL` - the native utility token of the `Dtravel` network. Within the `Dtravel` ecosystem, `TRVL` can be used for payments, incentives and rewards, participation in `Dtravel DAO` governance, and to provide liquidity to decentralized exchanges for rewards. This audit covers the ERC20-compliance validation of the `TRVL` token.

The basic information of Dtravel TRVL is as follows:

Table 1.1: Basic Information of Dtravel TRVL

Item	Description
Name	Dtravel
Type	Ethereum ERC20 Token Contract
Platform	Solidity
Audit Method	Whitebox
Audit Completion Date	November 4, 2021

In the following, we show the Git repository and the commit hash value used in this audit:

- https://github.com/dTravel/trvl-token-issue-contracts.git (95124b3)

And this is the commit ID after all fixes for the issues found in the audit have been checked in:

- https://github.com/dTravel/trvl-token-issue-contracts.git (ec79426)

1.2 About PeckShield

PeckShield Inc. [6] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of 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 OWASP Risk Rating Methodology [5]:

- 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

Impact \ Likelihood	*High*	*Medium*	*Low*
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 List of Check Items

Category	Check Item
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 list of check items and each would be labeled with a severity category. For one check item, if our tool does not identify any issue, the contract is considered safe regarding the check item. For any discovered issue, 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 check 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 findings after analyzing the `Dtravel TRVL` 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.

Severity	# of Findings	
Critical	0	
High	0	
Medium	1	
Low	0	
Informational	2	
Total	2	

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. After that, we examine a few identified issues of varying severities that need to be brought up and paid more attention to. (The findings are categorized in the above table.) Additional information can be found in the next subsection, and the detailed discussions are in Section 4.

2.2 Key Findings

Overall, no ERC20 compliance issue was found, and our detailed checklist can be found in Section 3. Also, there is no critical or high severity issue, although the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 2 informational recommendations.

Table 2.1: Key Dtravel TRVL Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Informational	Redundant State/Code Removal	Coding Practices	Fixed
PVE-002	Informational	Gas Efficient Replacement of memory to callata	Coding Practices	Fixed
PVE-003	Medium	Trust Issue of Admin Keys	Security Features	Confirmed

Besides recommending specific countermeasures to mitigate these issues, we also 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 and Section 4 for elaboration of reported issues.

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). The failure to meet these requirements means 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 exist 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

Item	Description	Status
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 is no ERC20 inconsistency or incompatibility issue found in the audited Dtravel TRVL. 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-

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

Item	Description	Status
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	✓
Approval() event	Is emitted on any successful call to approve()	✓

adopted ERC20 specification. 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/ERC2222), 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 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	✓
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 token holders to burn their own tokens	✓

4 Detailed Results

4.1 Redundant State/Code Removal

- ID: PVE-001
- Severity: Informational
- Likelihood: Low
- Impact: None
- Target: TokenVesting
- Category: Coding Practices [4]
- CWE subcategory: CWE-1041 [1]

Description

In the TRVL token contract, we observe the inclusion of certain unused code or the presence of unnecessary redundancies that can be safely removed. For example, the Ownable contract inherited by TokenVesting is not used throughout the entire TokenVesting contract. For better gas efficiency, we suggest removing the redundant inheritance.

```
contract TokenVesting is Ownable {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    event TokensReleased(address beneficiary, uint256 unreleased);

    // beneficiary of tokens after they are released
    address private immutable _beneficiary;

    // Durations and timestamps are expressed in UNIX time, the same units as block.
        timestamp.
    uint256 private immutable _cliff;
    ...
}
```

Listing 4.1: TokenVesting

Recommendation Remove the Ownable contract.

Status The issue has been addressed by the following commit: 45dbe03.

4.2 Gas Efficient Replacement of memory to calldata

- ID: PVE-002
- Severity: Informational
- Likelihood: Low
- Impact: Low
- Target: TRVL
- Category: Coding Practices [4]
- CWE subcategory: CWE-287 [2]

Description

In Solidity, the `external` functions can read array arguments directly from `calldata`. However, it comes to our attention that the `external` `TRVL::mintBatch()` function uses the array arguments as the `memory` type. This will make the `Solidity` `compiler` to copy the array arguments into memory before they can be used. Note that the memory allocation can be expensive, whereas reading from `calldata` is not. So we recommend changing the `memory` type to the `calldata` type for array arguments in `TRVL::mintBatch()`.

```
    function mintBatch(address[] memory to, uint256[] memory amount) external {
        require(
          to.length == amount.length,
          "TRVL: mintBatch inputs do not have same length"
        );
        for (uint256 i = 0; i < to.length; i++) {
          mint(to[i], amount[i]);
        }
    }
```

Listing 4.2: TRVL::mintBatch()

Recommendation Change the `memory` type to `calldata` type.

Status The issue has been addressed by the following commit: `45dbe03`.

4.3 Trust Issue of Admin Keys

- ID: PVE-003
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: `TokenVesting`
- Category: Security Features [3]
- CWE subcategory: CWE-287 [2]

Description

In the `TokenVesting` protocol, there is a special administrative account, i.e., `owner`. This `owner` account plays a critical role in governing and regulating the system-wide operations (e.g. token revoking). Our analysis shows that the privileged account needs to be scrutinized. In the following, we examine the privileged account and their related privileged accesses in current contracts.

To elaborate, we show below the `revokeTokenVesting()` function in the `TokenVesting` contract. This function allows the `owner` to withdraw all funds from the contract and send to a specific address.

```
function revokeTokenVesting(address ownerWallet) external onlyOwner {
  require(ownerWallet != address(0), "TokenVesting: invalid ownerWallet");

  _revoked = true;

  uint256 currentBalance = _token.balanceOf(address(this));
  if (currentBalance > 0) {
    _token.safeTransfer(ownerWallet, currentBalance);
  }

  emit TokensRevoked(_beneficiary, ownerWallet, currentBalance);
}
```

Listing 4.3: `AcetAdaptor::revokeTokenVesting()`

Note that it could be worrisome if the privileged `owner` account is a plain EOA account. A revised multi-sig account could greatly alleviate this concern, though it is still far from perfect. Specifically, a better approach is to eliminate the administration key concern by transferring the role to a community-governed DAO.

Recommendation Promptly transfer the privileged account to the intended `DAO`-like governance contract. All changed to privileged operations may need to be mediated with necessary timelocks. Eventually, activate the normal on-chain community-based governance life-cycle and ensure the intended trustless nature and high-quality distributed governance.

Status This issue has been confirmed.

5 | Conclusion

In this security audit, we have examined the design and implementation of the `Dtravel TRVL` token contract. During our audit, we first checked all respects related to the compatibility of the ERC20 specification and other known ERC20 pitfalls/vulnerabilities. We then proceeded to examine other areas such as coding practices and business logics. Overall, although no critical or high level vulnerabilities were discovered, we identified two issues that were promptly confirmed and addressed by the team. In the meantime, as disclaimed in Section 1.4, we appreciate any constructive feedbacks or suggestions about our findings, procedures, audit scope, etc.

References

[1] MITRE. CWE-1041: Use of Redundant Code. https://cwe.mitre.org/data/definitions/1041.html.

[2] MITRE. CWE-287: Improper Authentication. https://cwe.mitre.org/data/definitions/287.html.

[3] MITRE. CWE CATEGORY: 7PK - Security Features. https://cwe.mitre.org/data/definitions/254.html.

[4] MITRE. CWE CATEGORY: Bad Coding Practices. https://cwe.mitre.org/data/definitions/1006.html.

[5] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

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