*SECURITY AUDIT OF*

# POCO TOKEN V2
# SMART CONTRACT

**PRIVATE REPORT**

*SEP 07, 2021*

**Verichains Lab**

info@verichains.io

https://www.verichains.io

*Driving Technology > Forward*

# ACRONYMS AND ABBREVIATIONS

| NAME | DESCRIPTION |
|---|---|
| Ethereum | An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications. |
| Ether (ETH) | A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network |
| Binance Chain | Binance Chain is a blockchain software system developed by Binance and its community. |
| Binance Smart Chain (BSC) | Binance Smart Chain (BSC) is a blockchain network built for running smart contract-based applications. BSC runs in parallel with Binance's native Binance Chain (BC), which allows users to get the best of both worlds: the high transaction capacity of BC and the smart contract functionality of BSC. |
| Smart contract | A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract. |
| Solidity | A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform. |
| Solc | A compiler for Solidity. |

## EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Sep 07, 2021. We would like to thank the Poco team for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Poco Token V2 Contract. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

The assessment did not identify any vulnerability issue in Poco Token V2 smart contract code.

Overall, the code reviewed is of good quality, written with the awareness of smart contract development best practices.

# TABLE OF CONTENTS

# 1. MANAGEMENT SUMMARY

### 1.1. About Poco and Poco Token

Poco brings you into the new gaming world. Let's immerse yourself in Pocoland when leading the powerful team with 5 Poco warrants owning different elements, defeat your enemies then collect the huge reward by POCO token.

PocoToken (POCO) is the ERC-20 token of Poco, with an initial total supply of 180 million.

### 1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the Poco Token V2 smart contract. It was conducted on commit *`07b7f1ff926d9c12c9affaf0-fbe24ad0375d9988`* of branch *`master`* from GitHub repository of Poco Protocol.

The main audited contract is the PocoTokenV2 contract. Repository URL of the commit to be audited: https://github.com/pocolab/poco-protocal/blob/07b7f1ff926d9c12c9a-ffaf0fbe24ad0375d9988/contracts/poco/token/PocoTokenV2.sol.

### 1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)

- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in Table 2, depending on their severity level:

| SEVERITY LEVEL | DESCRIPTION |
|---|---|
| **CRITICAL** | A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately. |
| **HIGH** | A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority. |
| **MEDIUM** | A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed. |
| **LOW** | An issue that does not have a significant impact, can be considered as less important. |

*Table 2: Vulnerability severity levels*

## 1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

# 2. AUDIT RESULT

## 2.1. Overview

The Poco Token V2 smart contract is an ERC-20 Token Contract[1], which implements a standard interface for token as defined in EIP-20[2]. This standard provides basic functionality to transfer tokens, as well as allow tokens to be approved so they can be spent by another on-chain third party.

Table 3 lists some properties of the audited Poco Token V2 smart contract (as of the report writing time).

| PROPERTY | VALUE |
|---|---|
| Name | Poco Token |
| Symbol | POCO |
| Decimals | 18 |
| Total Supply | 180,000,000 ($\times 10^{18}$)<br>Note: the number of decimals is 18, so the total representation tokens will be 180,000,000, or 180 million. |

*Table 3: The Poco Token properties*

Besides the standard interface of an ERC-20 token, the Poco Token V2 smart contract also implements some additional functional logics by extending the following contracts:

- Context: provides information about the current execution context, including the sender of the transaction and its data. While these are generally available via msg.sender and msg.data, they should not be accessed in such a direct manner, since when dealing with GSN meta-transactions the account sending and paying for execution may not be the actual sender (as far as an application is concerned).
- Ownable: this contract has a basic access control mechanism, where there is an account (an owner) that can be granted exclusive access to specific functions. By default, the owner account will be the one that deploys the contract. The current owner can renounce or transfer ownership to a new owner.

Overall, the Poco Token V2 smart contract code is referenced from OpenZeppelin's ERC20 token standard. Some new functions are being added to the token contract like *`burnFrom`*, *`setBPAddrss`*, *`setBpEnabled`*. The first one is used to burn tokens (only *`owner`* can call this function), and the last two are used to configure the *`BPContract`*. In addition, if the

[1] https://ethereum.org/en/developers/docs/standards/tokens/erc-20
[2] https://eips.ethereum.org/EIPS/eip-20

*BPContract* function was enabled, the *_transfer* function will call the *BPContract*'s *protect(address sender, address receiver, uint256 amount)* function and check the returned value before continue.

## 2.2. Contract codes

The Poco Token V2 smart contract was written in Solidity language[3], with the minimum required version to be *0.8.0*.

The source codes consist of two contracts, two abstract contracts, and two interfaces. Almost all source codes in the Poco Token V2 smart contract are imported from OpenZeppelin's implementation of ERC20-related contracts[4].

### 2.2.1. IERC20 interface

This is the interface of ERC-20 standard as defined in EIP-20. The source code is stored in the *IERC20.sol* file, which is identical to OpenZeppelin's implementation.

### 2.2.2. Context abstract contract

This abstract contract provides information about the current execution context, including the sender of the transaction and its data. The source code is stored in the Context.sol file, which is identical to OpenZeppelin's implementation.

### 2.2.3. IERC20Metadata interface

This is the interface for the optional metadata functions from the ERC-20 standard. The source code is stored in the IERC20Metadata.sol file, which is identical to OpenZeppelin's implementation.

### 2.2.4. Ownable contract

This contract makes the Poco Token V2 Contract ownable. The source code is referenced from OpenZeppelin's implementation.

### 2.2.5. BPContract abstract contract

This abstract contract is used to call the Bot Protection contract before each *_transfer* call.

### 2.2.6. PocoTokenV2 contract

This is the main contract in the Poco Token V2 smart contract, which extends the *IERC20*, *IERC20Metadata*, interfaces and *Context*, *Ownable* contracts. Below is a summary of some important functions in this contract:

[3] https://docs.soliditylang.org/
[4] https://docs.openzeppelin.com/contracts/4.x/erc20

- *constructor()*: specifies the name, symbol and initial total token supply for the Poco token.
- *name()*: returns the token name.
- *getOwner()*: returns the token owner.
- *symbol()*: returns the token symbol.
- *totalSupply():* returns the total supply.
- *balanceOf(address account):* returns the balance of the input account.
- *transfer(address recipient, uint256 amount):* transfers *amount* tokens from the *sender* to *recipient*.
- *allowance(address owner, address spender):* returns the *amount* which *spender* is still allowed to withdraw from *owner*.
- *approve(address spender, uint256 amount):* allows *spender* to withdraw from the caller's account multiple times, up to the *amount*. If this function is called again, it overwrites the current allowance with *amount*.
- *transferFrom(address sender, address recipient, uint256 amount):* transfers *amount* of tokens from address *sender* to address *recipient*.
- *increaseAllowance(address spender, uint256 addedValue):* increases the allowance amount for *spender* by *addedValue*.
- *decreaseAllowance(address spender, uint256 subtractedValue):* decreases the allowance amount for *spender* by *subtractedValue*.
- *burnFrom(uint256 amount):* destroys *amount* tokens from *account*. *amount* is then deducted from the caller's allowance.
- *setBPAddrss(address _bp)*: set the implementation address of the BP contract.
- *setBpEnabled(bool _enabled)*: enable or disable the BP contract protection.

## 2.3. Findings

During the audit process, the audit team did not discover any security vulnerability issue in the Poco Token V2 smart contract. Only some minor issues (such as typos) were found and listed in the following section.

## 2.4. Additional notes and recommendations

### 2.4.1. BPContract function

Since we do not control the logic of the *BPContract*, there is no guarantee that *BPContract* will not contain any security related issues. With the current context, in case the *BPContract* is compromised, there is not yet a way to exploit the PocoTokenV2 contract, but we still note that here as a warning for avoiding any related issue in the future.

By the way, if having any issue, the *BPContract* function can be easily disabled anytime by the contract *owner* using the *setBpEnabled* function.

### 2.4.2. Typo in setBPAddrss function

There is a typo in function name at the line 409 in file ***PocoTokenV2.sol***

```
function setBPAddrss(address _bp) external onlyOwner
```

### 2.4.3. Misleading comments

There are some misleading comments from line 86 to 90 in the ***PocoTokenV2.sol*** file.

```
/**
* @dev Implementation of the {IERC20} interface.
* functions have been added to mitigate the well-known issues around setting
* allowances. See {IERC20-approve}.
*/
```

## 3. VERSION HISTORY

| VERSION | DATE | STATUS/CHANGES | CREATED BY |
|---|---|---|---|
| 1.0 | Sep 07, 2021 | Initial private report | Verichains Lab |

# APPENDIX A: FUNCTION CALL GRAPH

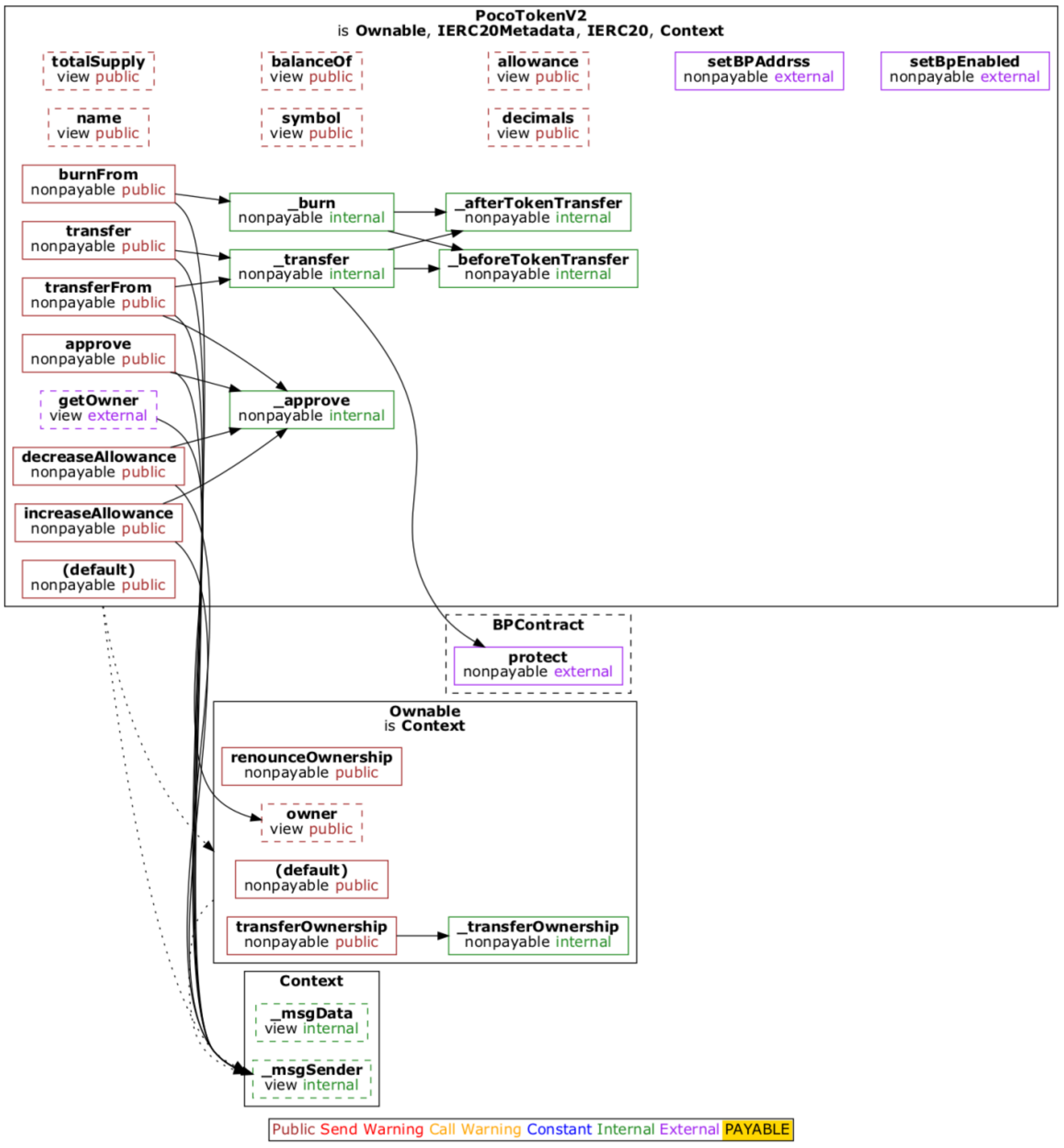

*Figure 1: The function call graph of PocoTokenV2 smart contract*