# SMART CONTRACT AUDIT REPORT

for

# ChainSwap MATTER/ASAP



Prepared By: Yiqun Chen

PeckShield

July 22, 2021

## Document Properties

| | |
|---|---|
| Client | ChainSwap |
| Title | Smart Contract Audit Report |
| Target | MATTER/ASAP |
| Version | 1.0-rc |
| Author | Jing Wang |
| Auditors | Jing Wang, Xuxian Jiang |
| Reviewed by | Yiqun Chen |
| Approved by | Xuxian Jiang |
| Classification | Confidential |

## Version Info

| Version | Date | Author | Description |
|---|---|---|---|
| 1.0-rc | July 22, 2021 | Jing Wang | 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

Given the opportunity to review the design document and related source code of the **MATTER/ASAP** 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 can be further improved due to the presence of certain issues related to ERC20-compliance, security, or performance. This document outlines our audit results.

## 1.1 About MATTER/ASAP

**MATTER/ASAP** is an ERC20-compliant token contract developed using the excellent smart contract base from `OpenZeppelin`. The main features or customizations of `MATTER/ASAP` include the full ERC20 compatibility and the initial supply of 100 million tokens to different parties.

The basic information of MATTER/ASAP is as follows:

Table 1.1: Basic Information of MATTER/ASAP

| Item | Description |
|---:|---|
| Issuer | ChainSwap |
| Website | https://chainswap.com/ |
| Type | Ethereum ERC20 Token Contract |
| Platform | Solidity |
| Audit Method | Whitebox |
| Audit Completion Date | July 22, 2021 |

In the following, we show the list of reviewed contracts used in this audit:

- https://ropsten.etherscan.io/address/0x9b99cca871be05119b2012fd4474731dd653febe#code
- https://ropsten.etherscan.io/address/0x08d59467e8fbee7575ed0905bba03903654cfbfd#code

- https://ropsten.etherscan.io/address/0x85f159b637344620d8f70254742850b5be0a98a0#code

And here is the list of new contracts that have been deployed after fixing issues reported here:

- https://ropsten.etherscan.io/address/0x5a402ae3d29a7f89c5943085e72786adc84c8f37#code
- https://ropsten.etherscan.io/address/0xf8149fbbeac622c471297fb43956ac5882d6650c#code
- https://ropsten.etherscan.io/address/0x9c88ffe3779204154e33e4812a5a6507f37b6fea#code

## 1.2 About PeckShield

PeckShield Inc. [8] 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 [7]:

- 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.

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 1.2: Vulnerability Severity Classification

| Impact \ Likelihood | High | Medium | Low |
|---|---|---|---|
| *High* | Critical | High | Medium |
| *Medium* | High | Medium | Low |
| *Low* | Medium | Low | Low |

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.

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 |

# 2 Findings

## 2.1 Summary

Here is a summary of our findings after analyzing the `MATTER/ASAP` 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 | 2 | ■ ■ |
| Low | 0 | |
| Informational | 2 | ■ ■ |
| Total | 4 | |

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. However, the smart contract implementation can be improved because of the existence of 2 medium-severity vulnerabilities and 2 informational recommendations.

Table 2.1: Key MATTER/ASAP Audit Findings

| ID | Severity | Title | Category | Status |
|---|---|---|---|---|
| PVE-001 | Informational | Excessive Contract Inheritance | Coding Practices | Fixed |
| PVE-002 | Medium | Trust Issue Of Admin Roles | Security Features | Confirmed |
| PVE-003 | Informational | Redundant State/Code Removal | Coding Practices | Fixed |
| PVE-004 | Medium | Improved Validation Of Function Arguments | Coding Practices | Fixed |

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 MATTER/ASAP. 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 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 | — |

# 4 Detailed Results

## 4.1 Excessive Contract Inheritance

- ID: PVE-001
- Severity: Informational
- Likelihood: None
- Impact: None
- Target: ERC20UpgradeSafe
- Category: Coding Practices [6]
- CWE subcategory: CWE-563 [4]

### Description

In current implementation, we observe that both ERC20UpgradeSafe and ContextUpgradeSafe contracts inherit from the Initializable contract. In addition, ERC20UpgradeSafe inherits from ContextUpgradeSafe , which indicates the inheritance of Initializable in ERC20UpgradeSafe is redundant. Note that any excessive inheritance likely introduces unnecessarily convoluted dependency and makes it harder to reason or infer derived function implementations. With that, we may consider to remove the redundant inheritance of Initializable in ERC20UpgradeSafe.

```
contract ERC20UpgradeSafe is Initializable, ContextUpgradeSafe, IERC20 {
    using SafeMath for uint256;
    using Address for address;
    ...
}
```

Listing 4.1: ERC20UpgradeSafe

```
contract ContextUpgradeSafe is Initializable {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    ...
}
```

Listing 4.2: ContextUpgradeSafe

**Recommendation** Remove the redundant inheritance of Initializable in ERC20UpgradeSafe.

**Status** This issue has been fixed by following the above suggestion.

## 4.2 Trust Issue Of Admin Roles

- ID: PVE-002
- Severity: Medium
- Likelihood: Low
- Impact: High
- Target: `BaseAdminUpgradeabilityProxy`, `__AdminUpgradeabilityProductProxy__`
- Category: Security Features [5]
- CWE subcategory: CWE-287 [3]

### Description

In the `__AdminUpgradeabilityProductProxy__` contract, there is a privileged `admin` account (assigned in the `constructor`) that plays a critical role in governing and regulating the related upgrade operations.

To elaborate, we show below the `__AdminUpgradeabilityProductProxy_init__()` function in the `InitializableProductProxy` contract and the `upgradeTo()` function in the `BaseAdminUpgradeabilityProxy` contract. These functions are guarded with `ifAdmin` and allows the `admin` to change the implementation of the proxy to any contract.

```
function __AdminUpgradeabilityProductProxy_init__(address logic, address factory, bytes32 name, bytes memory data) external ifAdmin {
  __AdminUpgradeabilityProductProxy_init(logic, factory, name, data);
}
```

Listing 4.3: `InitializableProductProxy::__AdminUpgradeabilityProductProxy_init__()`

```
function upgradeTo(address newImplementation) external ifAdmin {
  _upgradeTo(newImplementation);
}
```

Listing 4.4: `BaseAdminUpgradeabilityProxy::upgradeTo()`

We understand the need of the privileged functions for contract upgrade, but at the same time the extra power to the admin roles may also be a counter-party risk to the contract users. It is worrisome if the privileged `owner` account is a plain EOA account. Note that a 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 by the teams. And the team clarifies that they will transfer the privileged account to the DAO-like governance contract after the protocol is stabilized.

## 4.3 Redundant State/Code Removal

- ID: PVE-003
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: ERC20UpgradeSafe
- Category: Coding Practices [6]
- CWE subcategory: CWE-563 [4]

### Description

The protocol makes good use of a few library functions from OpenZeppelin. One example is the widely-used SafeMath library. However, we observe the inclusion of certain unused code or the presence of unnecessary redundancies that can be safely removed. To elaborate, we show the Address library from the Include.sol.

```
/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // According to EIP-1052, 0x0 is the value returned for not-yet created
            accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is
            returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
```

```
            bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
            // solhint-disable-next-line no-inline-assembly
            assembly { codehash := extcodehash(account) }
            return (codehash != accountHash && codehash != 0x0);
        }

        /**
         * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
         * `recipient`, forwarding all available gas and reverting on errors.
         *
         * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
         * of certain opcodes, possibly making contracts go over the 2300 gas limit
         * imposed by `transfer`, making them unable to receive funds via
         * `transfer`. {sendValue} removes this limitation.
         *
         * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
         *
         * IMPORTANT: because control is transferred to `recipient`, care must be
         * taken to not create reentrancy vulnerabilities. Consider using
         * {ReentrancyGuard} or the
         * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-pattern[checks-effects-interactions pattern].
         */
        function sendValue(address payable recipient, uint256 amount) internal {
            require(address(this).balance >= amount, "Address: insufficient balance");

            // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
            (bool success, ) = recipient.call{ value: amount }("");
            require(success, "Address: unable to send value, recipient may have reverted");
        }
    }
```

Listing 4.5: `Include.sol`

This particular `Address` library provides two handy routines: `isContract()` and `sendValue()`. The first one determines whether the provided account is a contract or not while the second one provides a replacement of Solidity's `transfer` by forwarding all available gas and reverting on errors (due to introduced gas cost change of certain opcodes in EIP1884 [1]). However, both these two routines are not used anymore in current code base and can therefore be safely removed.

**Recommendation** Remove the `Address` library in current code base and delete its reference in `ERC20UpgradeSafe`.

**Status** This issue has been fixed by following the above suggestion.

## 4.4 Improved Validation Of Function Arguments

- ID: PVE-004
- Severity: Medium
- Likelihood: Low
- Impact: High
- Target: `ASAPING`
- Category: Coding Practices [6]
- CWE subcategory: CWE-1041 [2]

### Description

In the `ASAPING` contract, the `startUnlock()` is a privileged function that is used by the governor of the contract to start the token-unlocking process. To elaborate, we show below the related code snippet of the contract.

```
function startUnlock(address _token, address recipient, uint _firstTime, uint
    _firstRatio, uint _begin, uint _end) external governance {
    token = _token;
  _setupDecimals(ERC20UpgradeSafe(token).decimals());
    _mint(recipient, IERC20(token).balanceOf(address(this)).sub(_totalSupply));

    firstTime = _firstTime;
    firstRatio = _firstRatio;
    begin = _begin;
    end = _end;
}
```

Listing 4.6: `ASAPING::startUnlock()`

It comes to our attention that the `startUnlock()` function has the inherent assumption that the value of the given `_token` would not be changed among multiple calls of this function. However, this is not enforced inside the `startUnlock()` function. The underlying token's type could be changed by the governor while the minted token's type would be kept the same all the time. And this may introduce unexpected loss.

Additionally, we notice there is no guarantee that the value of `_begin` is smaller than the value of `_end`. Also, the value of `firstRatio` may be larger than $1e18$, which could introduce unexpected result. To mitigate, we suggest to properly validate the values of `_begin`, `_end` and `firstRatio` before using them.

**Recommendation** Improve the validation of function arguments and do not allow the change of token's type in `startUnlock()`.

**Status** This issue has been fixed by following the above suggestion.

# 5 | Conclusion

In this security audit, we have examined the design and implementation of the MATTER/ASAP 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] The Ethereum Foundation. EIP-1884: Repricing for Trie-Size-Dependent Opcodes. https://eips.ethereum.org/EIPS/eip-1884.

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

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

[4] MITRE. CWE-563: Assignment to Variable without Use. https://cwe.mitre.org/data/definitions/563.html.

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

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

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

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