

# Security Audit

## of the MADANA PAX Smart Contract

September 23, 2019

Produced for




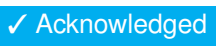
by



CHAINSECURITY

# Table Of Contents

Foreword . . . . .	1
Executive Summary . . . . .	1
Audit Overview . . . . .	2
1. Methodology of the Audit . . . . .	2
2. Scope of the Audit . . . . .	2
3. Depth of the Audit . . . . .	2
4. Terminology . . . . .	3
5. Limitations . . . . .	3
System Overview . . . . .	4
1. Extra Token Features . . . . .	4
2. System Roles . . . . .	4
3. Trust Model . . . . .	4
Best Practices in MADANA's project . . . . .	5
1. Hard Requirements . . . . .	5
2. Best Practices . . . . .	5
3. Smart Contract Test Suite . . . . .	5
Formal Functional Properties . . . . .	6
1. Property syntax and semantics . . . . .	6
2. Properties related to the MADANA contract . . . . .	6
2.1 Checks for immutable and correct token details ✓ Verified . . . . .	6
2.2 Token transfer related properties ✓ Verified . . . . .	7
2.3 Balance and supply related properties ✓ Verified . . . . .	9
2.4 Burn related properties ✓ Verified . . . . .	9
2.5 Owner related properties ✓ Verified . . . . .	10
2.6 Allowance related properties ✓ Verified . . . . .	10

Security Issues . . . . .	11
Trust Issues . . . . .	12
Design Issues . . . . .	13
1. Shadowed_owner   . . . . .	13
Recommendations / Suggestions . . . . .	14
Addendum and general considerations . . . . .	15
1. Forcing ETH into a smart contract . . . . .	15
Disclaimer . . . . .	16

# Foreword

We would first and foremost like to thank MADANA for giving us the opportunity to audit their smart contract. This document outlines our methodology, limitations and results.

– ChainSecurity

## Executive Summary

MADANA engaged CHAINSECURITY to perform a security audit of the MADANA PAX, an Ethereum-based smart contract.

CHAINSECURITY audited the smart contract which is going to be deployed on the public Ethereum chain. Audits of CHAINSECURITY use state-of-the-art tools for detection of generic vulnerabilities and checks of custom functional requirements. A thorough manual code review was performed by CHAINSECURITY's experts to ensure that the highest security standards are met. Additionally, to guarantee that the MADANA PAX contract is secure and functionally correct, CHAINSECURITY verified the contract's code with respect to 24 relevant functional requirements. During the audit CHAINSECURITY found one design issue which has been acknowledged by MADANA. No security or trust issues have been found and CHAINSECURITY considers the code to be secure. The employed coding practices and documentation are up to a high standard.

# Audit Overview

## Methodology of the Audit

CHAINSECURITY's methodology in performing the security audit consisted of four chronologically executed phases:

1. Understanding the existing documentation, purpose and specifications of the smart contract.
2. Executing automated tools to scan for generic security vulnerabilities.
3. Manual analysis covering both functional (best effort based on the provided documentation) and security aspects of the smart contract by one of our CHAINSECURITY experts.
4. Writing the report with the individual vulnerability findings and potential exploits.

## Scope of the Audit

Source code files received	August 9, 2019
Git commit	245bcb334c3d79f9f0d041ec18476e3e20b01896
EVM version	PETERSBURG
Initial Compiler	SOLC compiler, version 0.5.10
Final code update received	September 18, 2019
Final commit	d0a76a04002439d59011da2fb6697d06b937313e

The scope of the audit is limited to the following source code files.

In Scope	File	SHA-256 checksum
	token/ERC20/ERC20MultiTransfer.sol	43a5aa34c43724bef1a4da69960b191438a8576afbb94b21b74be219a7e01ac7
	token/ERC20/ERC20Burnable.sol	ec43420acc852d517a4bfb051210f09300fd2bbc1081f5278967e6efaa00263b
	/MADANA/MADANA.sol	95c124fb5cef1aeef9e23d960ed28c1fdb2bd93b5b2c6bffde3180b92acb353
	utils/Reclaimable.sol	ef509411ca63119ae3054a4a2e4517c59b448f38f46dd807acc31ddb864fc04a

For these files the following categories of issues were considered:

In Scope	Issue Category	Description
	Security Issues	Code vulnerabilities exploitable by malicious transactions
	Trust Issues	Potential issues due to actors with excessive rights to critical functions
	Design Issues	Implementation and design choices that do not conform to best practices

## Depth of the Audit

The scope of the security audit conducted by CHAINSECURITY was restricted to:

- Scanning the contracts listed above for generic security issues using automated systems and manually inspecting the results.
- Manual audit of the contracts listed above for security issues.

## Terminology





For the purpose of this audit, CHAINSECURITY has adopted the following terminology. For security vulnerabilities, we specify the *likelihood*, *impact* and *severity* (inspired by the OWASP risk rating methodology<sup>1</sup>).

**Likelihood** represents the likelihood of a security vulnerability to be encountered or exploited in the wild.










**Impact** specifies the technical and business-related consequences of an exploit.

**Severity** is derived based on the likelihood and the impact calculated previously.





We categorise the findings into four distinct categories, depending on their severities:


-  Low: can be considered less important
-  Medium: should be fixed
-  High: we strongly recommend fixing it before release
-  Critical: needs to be fixed before release

These severities are derived from the likelihood and the impact using the table below, following a standard approach in risk assessment.

LIKELIHOOD	IMPACT		
	High	Medium	Low
High			
Medium			
Low			

During the audit, concerns might arise or tools might flag certain security issues. After carefully inspecting the potential security impact, we assign the following labels:

-  **No Issue**: no security impact
-  **Fixed**: the issue is addressed technically, for example by changing the source code
-  **Addressed**: the issue is mitigated non-technically, for example by improving the user documentation and specification
-  **Acknowledged**: the issue is acknowledged and it is decided to be ignored, for example due to conflicting requirements or other trade-offs in the system

Findings that are labelled as either  **Fixed** or  **Addressed** are resolved and therefore pose no security threat. Their severity is still listed, but just to give the reader a quick overview of what kind of issues were found during the audit.

## Limitations

Security auditing cannot uncover all existing vulnerabilities; even an audit in which no vulnerabilities are found is not a guarantee of a secure smart contract. However, auditing enables discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary.

In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire smart contract application, while some lack protection only in certain areas. This is why we carry out a source code review aimed at determining all locations that need to be fixed. Within the customer-determined timeframe, CHAINSECURITY has performed auditing in order to discover as many vulnerabilities as possible.

<sup>1</sup>[https://www.owasp.org/index.php/OWASP\\_Risk\\_Rating\\_Methodology](https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology)

# System Overview

Token Name & Symbol	MADANA PAX, MDN
Decimals	18 decimals
Tokens issued	100 million
Token Type	ERC20
Token Generation	Pre-Minted, Burnable

Table 1: Facts about the MDN token.

## Extra Token Features

**ERC20MultiTransfer** A token holder can send their tokens to multiple accounts at once using the `multiTransfer` functionality.

## System Roles

In this section we outline the different roles and their permissions and purpose within the system.

**Owner** The owners can reclaim ERC20 tokens accidentally sent to the MDN token smart contract. They can decrease the total supply by burning tokens. The ownership can be transferred to a different account if required.

## Trust Model

Here, we present the trust assumptions for the roles in the system as provided by MADANA PAX. Auditing the enforcement of these assumptions is outside the scope of the audit. Users of MADANA PAX should keep in mind that they have to rely on MADANA PAX to correctly implement and enforce these trust assumptions.

**Owner** The owner is fully trusted. Hence, it must be completely honest at all times and make no accidental mistakes such as typos.

**User** A regular user is untrusted and assumed to be potentially malicious. Regular users are expected to only execute attacks which are economically beneficial for them.

# Best Practices in MADANA's project

CHAINSECURITY is determined to deliver the best results to ensure the security of a project. To enable us to do so, we are listing Hard Requirements which must be fulfilled to allow us to start the audit. Furthermore we are providing a list of proven best practices. Following them will make audits more meaningful by allowing efforts to be focused on subtle and project-specific issues rather than the fulfilment of general guidelines.

## Hard Requirements

These requirements ensure that the MADANA's project can be audited by CHAINSECURITY.

- ✓ **All files and software for the audit have been provided to CHAINSECURITY**  
The project needs to be complete. Code must be frozen and the relevant commit or files must have been sent to CHAINSECURITY. All third party code (like libraries) and third-party software (like the solidity compiler) must be exactly specified or made available. Third party code can be located in a folder separated from client code (and the separation needs to be clear) or included as dependencies. If dependencies are used, the version(s) need to be fixed.
- ✓ The code must compile and the required compiler version must be specified. When using outdated versions with known issues, clear reasons for using these versions are being provided.
- ✓ There are migration/deployment scripts executable by CHAINSECURITY and their use is documented.
- ✓ The code is provided as a Git repository to allow reviewing of future code changes.

## Best Practices

Although these requirements are not as important as the previous ones, they still help to make the audit more valuable.

- ✓ There are no compiler warnings, or warnings are documented.
- ✓ Code duplication is minimal, or justified and documented.
- ✓ The output of the build process (including possible flattened files) is not committed to the Git repository.
- ✓ The project only contains audit-related files, or, if this is not possible, a meaningful distinction is made between modules that have to be audited and modules that CHAINSECURITY should assume are correct and out-of-scope.
- ✓ There is no dead code.
- ✓ The code is well-documented.
- ✓ The high-level specification is thorough and enables a quick understanding of the project without any need to look at the code.
- ✓ Both the code documentation and the high-level specification are up-to-date with respect to the code version CHAINSECURITY audits.
- ✓ Functions are grouped together according to either the Solidity guidelines<sup>2</sup>, or to their functionality.

## Smart Contract Test Suite

In this section, CHAINSECURITY comments on the smart contract test suite of MADANA PAX. While the test suite is not a component of the audit, a good test suite is likely to result in better code.

The tests are easy to run for CHAINSECURITY, using the documentation provided by the client. They validate the migration and initialization of the contract. The tests seem extensive and appear to cover the intended use cases. Failure cases are also covered, which is good. However, the tests do not contain proper assertion to check for the correct error message making sure that errors were handled as expected, the test just ensure that the call threw.

<sup>2</sup><https://solidity.readthedocs.io/en/v0.4.24/style-guide.html#order-of-functions>



# Formal Functional Properties

CHAINSECURITY investigated selected functional requirements. Below, we list the considered requirements and indicate whether they have been successfully verified (marked with label **✓ Verified**) or not (marked with label **✗ Does not hold**).

## Property syntax and semantics

The formalization of the properties uses the syntax of Solidity extended with temporal operators (such as **always** and **prev**) and additional logical connectives (such as implication, written with  $\implies$ ). For example, **always** quantifies over all contract states and **prev** refers to the previous state (before a transaction has executed).

The expression `FUNCTION == MADANA.transfer(address,uint256)` evaluates to `true` if the current transaction is a call to function `transfer(address,uint256)`. The expression `MADANA.transfer(address,uint256)[0]` returns the first argument of the function `MADANA.transfer(address,uint256)`. For brevity, we name arguments in function declarations and refer to them by their name. For example, given a function with named arguments `MADANA.transfer(address to, uint256 tokens)`, we write `to` to refer to its first argument.

The properties are interpreted over a sequence of contract states, induced by executing a given sequence of transactions. A property is *verified* if it holds for any possible sequences of contract states.

## Properties related to the MADANA contract

In the formalization of the properties below, the symbols `X` and `Y` represents arbitrary values (such as `0x123`).

### 2.1 Checks for immutable and correct token details **✓ Verified**

Some of the properties might seem trivial but are nonetheless crucial for institutions like exchanges.

#### Immutable decimals **✓ Verified**

The decimals specified in the MADANA PAX must be immutable.

```
always( prev(MADANA._decimals) == MADANA._decimals);
```

#### Correct decimals **✓ Verified**

The decimals of MADANA PAX always equal two.

```
always( MADANA._decimals == 18);
```

We note that this property implies the *Immutable decimals* property given above.

#### Immutable name **✓ Verified**

The name specified in the MADANA PAX must be immutable.

```
always( prev(MADANA._name) == MADANA._name);
```

#### Correct name **✓ Verified**

The name of MADANA PAX always equals "MADANA PAX".

```
always( MADANA._name == "MADANA PAX");
```

We note that this property implies the *Immutable name* property given above.

#### Immutable symbol **✓ Verified**

The symbol specified in the MADANA PAX must be immutable.

```
always( prev(MADANA._symbol) == MADANA._symbol);
```

#### Correct symbol **✓ Verified**

The symbol of MADANA PAX always equals "MDN".

```
always( MADANA._symbol == "MDN");
```

We note that this property implies the *Immutable symbol* property given above.

## 2.2 Token transfer related properties ✓ Verified

### Compliant transfer function ✓ Verified

The transfer function must comply to the ERC20 standard.

- The balance of the transaction sender is decreased by the provided amount of tokens.
- The balance of the receiver to is increased by the provided amount of tokens.

```
always(  
  (FUNCTION == MADANA.transfer(address to, uint256 tokens)) ==>  
    (  
      (  
        (MADANA._balances[msg.sender] == (prev(MADANA._balances)[msg.sender]  
          - tokens)) &&  
        (MADANA._balances[to] == (prev(MADANA._balances)[to] + tokens))  
      ) ||  
      (msg.sender == to)  
    )  
);
```

The condition (`msg.sender == to`) indicates that the balances must not increase/decrease if the transaction sender equals the receiver.

### Compliant transferFrom function ✓ Verified

The transferFrom function must comply to the ERC20 standard.

- The balance of the `from` account is decreased by the provided amount of tokens.
- The balance of the receiving address to is increased by the provided amount of tokens.
- The allowance provided to the transaction sender was greater than the provided amount of tokens.
- The allowance provided to the transaction sender is decreased by the provided amount of tokens.

```
always(  
  (FUNCTION == MADANA.transferFrom(address from, address to, uint256 tokens))  
    ==>  
    (  
      (from == to) ||  
      (  
        (MADANA._balances[from] == (prev(MADANA._balances)[from] - tokens)) &&  
        (MADANA._balances[to] == (prev(MADANA._balances)[to] + tokens))  
      )  
    )  
);
```

```
always(  
  (FUNCTION == MADANA.transferFrom(address from, address to, uint256 tokens))  
    ==>  
    (  
      (prev(MADANA._allowances)[from][msg.sender] >= tokens) &&  
      (  
        MADANA._allowances[from][msg.sender] ==  
          (prev(MADANA._allowances)[from][msg.sender] - tokens)  
      )  
    )  
);
```

### Allowance increase safety ✓ Verified

The allowances provided to Y by X may be increased only by X.

```

always(
  (prev(MADANA._allowances[X][Y]) < MADANA._allowances[X][Y])
  ==> (msg.sender == X)
);

```

The above property states that if the allowance provided to a user Y by user X increases, then the transaction was made by user X.

#### Allowance decrease safety ✓ Verified

Only the issuer or the aquirer of an allowance may decrease the allowed amount.

```

always(
  (prev(MADANA._allowances[X][Y]) > MADANA._allowances[X][Y])
  ==> ((msg.sender == X) || (msg.sender == Y))
);

```

#### Balance decrease safety ✓ Verified

The balance of a user X may decrease only by:

- Transactions made by user X;
- Transactions to function transferFrom(address from, address to, uint256 tokens) where from equals X and the allowance provided to the transaction sender by X exceeds the provided amount of tokens.

```

always(
  (prev(MADANA._balances[X]) > MADANA._balances[X]) ==>
  (
    (msg.sender == X) ||
    (
      (FUNCTION == MADANA.transferFrom(address from, address to, uint256
        tokens)) &&
      (from == X) &&
      (prev(MADANA._allowances)[X][msg.sender] >= tokens)
    )
  )
);

```

#### No overflow in transfer ✓ Verified

Function transfer must not decrease the balance of the token receiver.

```

always(
  (FUNCTION == MADANA.transfer(address to, uint256 tokens)) ==>
  (MADANA._balances[to] >= prev(MADANA._balances)[to])
);

```

#### No underflow in transfer ✓ Verified

Function transfer must not increase the balance of the transaction sender.

```

always(
  (FUNCTION == MonartMonartToken.transfer(address, uint256)) ==>
  (MADANA._balances[msg.sender] <= prev(MADANA._balances)[msg.sender])
);

```

#### No overflow in transferFrom for balances ✓ Verified

Function transferFrom must not decrease the balance of the receiving address to.

```

always(
  (FUNCTION == MADANA.transferFrom(address from, address to, uint256
    tokens)) ==>
  (MADANA._balances[to] >= prev(MADANA._balances)[to])
);

```

### No underflow in transferFrom for balances ✓ Verified

Function transferFrom must not increase the balance of the **from** account.

```
always(
  (FUNCTION == MADANA.transferFrom(address from, address to, uint256
    tokens)) ==>
    (MADANA._balances[from] <= prev(MADANA._balances)[from])
);
```

### No underflow in transferFrom for allowance ✓ Verified

Function transferFrom must not increase the allowed amount of the transaction sender.

```
always(
  (FUNCTION == MADANA.transferFrom(address from, address to, uint256
    tokens)) ==>
    (
      prev(MADANA._allowances)[from][msg.sender] >=
        MADANA._allowances[from][msg.sender]
    )
);
```

## 2.3 Balance and supply related properties ✓ Verified

### Correct balances ✓ Verified

The sum of balances must always equal the total supply of tokens.

```
always(SUM(MADANA._balances) == MADANA._totalSupply);
```

### Only mint and burn change total supply ✓ Verified

Total supply can be changed only by function mint or burn.

```
always(
  (MADANA._totalSupply != prev(MADANA._totalSupply)) ==>
    (FUNCTION == MADANA.burn(uint256))
);
```

## 2.4 Burn related properties ✓ Verified

### Only owner can burn ✓ Verified

Only owner can change the supply of MADANA.

```
always(
  (MADANA._totalSupply != prev(MADANA._totalSupply)) ==>
    (msg.sender == prev(MADANA._owner))
);
```

### Burn decreases total supply and account balance correctly ✓ Verified

Function burn decrease the \_totalSupply and balance of msg.sender by provided amount.

```
always(
  (FUNCTION == MADANA.burn(uint256 amount)) ==>
    (
      (MADANA._totalSupply == (prev(MADANA._totalSupply) - amount)) &&
      (MADANA._balances[msg.sender] == (prev(MADANA._balances)[msg.sender] -
        amount))
    )
);
```

## 2.5 Owner related properties ✓ Verified

### Only owner can change owner ✓ Verified

Only owner can change owner

```
always(  
  (prev(MADANA._owner) != MADANA._owner) ==>  
  (msg.sender == prev(MADANA._owner) )  
);
```

### Constant owner ✗ Does not hold

It is expected that the owner can be changed and that this property does not hold, which signifies that the owner can be changed.

```
always( MADANA._owner == 0x445cB41f5CD66b00c9BAc837f7A2c5db4132099C );
```

## 2.6 Allowance related properties ✓ Verified

### Compliant approve function ✓ Verified

Function approve must set the allowed amount for the spender with the msg.sender as an issuer.

```
always(  
  (FUNCTION == MADANA.approve(address spender, uint256 amount)) ==>  
  (  
    (MADANA._allowances[msg.sender][spender] == amount)  
  )  
);
```

# Security Issues

This section relates to our investigation into security issues. It is meant to highlight times when we found specific issues, but also mentions what vulnerability classes do not appear, if relevant.

CHAINSECURITY has no concerns to raise in this category of the report.

## Trust Issues

This section reports functionality that is not enforced by the smart contract and hence correctness relies on additional trust assumptions.

CHAINSECURITY has no concerns to raise in this category of the report.

# Design Issues

This section lists general recommendations about the design and style of MADANA's project. These recommendations highlight possible ways for MADANA to improve the code further.

Shadowed `_owner`  

Contract MADANA declares the following constant:

```
address private constant _owner = 0x445cB41f5CD66b00c9BAc837f7A2c5db4132099C;
```

However, a variable with the same name `_owner` is already declared and used in the parent `Ownable` contract. Inside the contract MADANA the constant shadows the variable and `_owner` represents the constant. In the parent contract `_owner` represents the mutable variable.

In the current code, the owner is after deployment only accessed by the methods declared in the `Ownable` contract and the smart contract behaves as intended. It is possible that a user who is reading the source code of the MADANA smart contracts gets the impression that the owner is constant, while it actually can be changed.

**Acknowledged:** MADANA is aware of the possibility that reading the code might imply a constant owner. MADANA will clarify that the owner can be changed in appropriate places in their documentation. Given that the public variable `owner`, which is read out by Etherscan and used by third-party smart contracts, is returning the correct owner, the issue is unlikely to affect ordinary users.



## Recommendations / Suggestions

CHAINSECURITY has no concerns to raise in this category of the report.

# Addendum and general considerations

Blockchains and especially the Ethereum Blockchain might often behave differently from common software. There are many pitfalls which apply to all smart contracts on the Ethereum blockchain.

CHAINSECURITY mentions general issues in this section which are relevant for MADANA's code, but do not require a fix. Additionally, CHAINSECURITY mentions information in this section, to clarify or support the information in the security report. This section, therefore, serves as a reminder to create awareness for MADANA and potential users.

## Forcing ETH into a smart contract

Regular ETH transfers to smart contracts can be blocked by those smart contracts. On the high-level this happens if the according solidity function is not marked as **payable**. However, on the EVM levels there exist different techniques to transfer ETH in unblockable ways, e.g. through **selfdestruct** in another contracts. Therefore, many contracts might theoretically observe "locked ETH", meaning that ETH cannot leave the smart contract any more. In most of these cases, it provides no advantage to the attacker and is therefore not classified as an issue.

# Disclaimer

UPON REQUEST BY MADANA, CHAINSECURITY LTD. AGREES MAKING THIS AUDIT REPORT PUBLIC. THE CONTENT OF THIS AUDIT REPORT IS PROVIDED "AS IS", WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND, AND CHAINSECURITY LTD. DISCLAIMS ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT. COPYRIGHT OF THIS REPORT REMAINS WITH CHAINSECURITY LTD..