Veridise. Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

AsMatch Token

Veridise Inc.

December 21, 2023

► **Prepared For:**

AsMatch
https://www.asmatch.app/

► **Prepared By:**

Jon Stephens
Benjamin Sepanski

► **Contact Us:** contact@veridise.com

► **Version History:**

Dec. 20, 2023 Initial Draft

Contents

1 Executive Summary

On Dec. 18, 2023, AsMatch engaged Veridise to review the security of their AsMatch Token. The review covered the Solidity source code of the token implementation. Veridise conducted the assessment over 2 person-days, with 2 engineers reviewing code over 1 day on commit `0xdf61980`. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing.

Code assessment. The AsMatch Token developers provided the source code of the AsMatch Token contract for review. The token contract is very small as it only inherits from the OpenZeppelin v4.9.5 `ERC20PresetMinterPauser` contract. At the time of writing, no security patches have been released for any of the contracts that are inherited by `ERC20PresetMinterPauser`.

To facilitate the Veridise auditors' understanding of the code, the AsMatch Token developers gave the auditors access to their repository which also contained their deployment script. No external documentation or tests written by developers were provided.

Summary of issues detected. The audit uncovered 1 issue (V-ASM-VUL-001), which was assessed to be of low severity by the Veridise auditors. This low issue corresponds to potential centralization risks.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
AsMatch Token	`0xdf61980`	Solidity	Manta Pacific

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Dec. 18, 2023	Manual & Tools	2	2 person-day

Table 2.3: Vulnerability Summary.

Name	Number	Resolved
Critical-Severity Issues	0	0
High-Severity Issues	0	0
Medium-Severity Issues	0	0
Low-Severity Issues	1	0
Warning-Severity Issues	0	0
Informational-Severity Issues	0	0
TOTAL	1	0

Table 2.4: Category Breakdown.

Name	Number
Centralization	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of AsMatch Token's smart contract. In our audit, we sought to answer the following questions:

- Can the contract be manipulated such that tokens can be stolen from users?
- Can a user manipulate tokens that are owned by another user?
- Are there risks associated with centralization?
- Are any minting functions properly guarded by access controls?
- Does the token adhere to the behaviors defined in the ERC20 specification?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis & testing tools. In particular, we conducted our audit with the aid of the following techniques:

- *Static analysis.* To identify potential common vulnerabilities, we leveraged our custom smart contract analysis tool Vanguard, as well as the open-source tool Slither. These tools are designed to find instances of common smart contract vulnerabilities, such as reentrancy and uninitialized variables.
- *Fuzzing/Property-based Testing.* We also leverage fuzz testing to determine if the protocol may deviate from the expected behavior. To do this, we formalize the desired behavior of the protocol as [V] specifications and then use our fuzzing framework OrCa to determine if a violation of the specification can be found.

Scope. The scope of this audit is limited to the source code of the ASM token contract.

Methodology. Veridise auditors reviewed the reports of previous audits for AsMatch Token, inspected the provided tests, and read the AsMatch Token documentation. They then began a manual audit of the code assisted by both static analyzers and automated testing.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

Vulnerability Report 4

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-ASM-VUL-001	Centralization Risk	Low	Open

4.1 Detailed Description of Issues

4.1.1 V-ASM-VUL-001: Centralization Risk

Severity	Low	**Commit**	df61980
Type	Centralization	**Status**	Open
File(s)	`contracts/AsMatchToken.sol`		
Location(s)	N/A		
Confirmed Fix At			

Similar to many projects, the `ASM` token declares administrator roles that are given special permissions. In particular, it inherits from OpenZeppelin's `ERC20PresentMinterPauser` contract which uses OpenZeppelin's `AccessControl` utility contract to declare minter, pauser and administrator roles. These roles are then given the ability to mint tokens, pause token transfers and modify contract roles.

Impact If a private key were stolen, a hacker would have access to sensitive functionality that could compromise the project. For example, a malicious minter could mint a large number of tokens for themselves, and a malicious pauser could DoS users.

Recommendation As these are all particularly sensitive operations, we would encourage the developers to utilize a decentralized governance or multi-sig contract as opposed to an EOA as an EOA introduces a single point of failure.

Fuzz Testing 5

5.1 Methodology

Our goal was to fuzz test AsMatch Token to assess its compliance with the ERC 20 specification. The fuzzing focused on functional correctness i.e, whether the implementation deviates from the intended behavior. We used OrCa as our fuzzer and wrote invariants–logical formulas that should hold after every transaction. We then encoded those invariants as assertions in [V].

5.2 Properties Fuzzed

Table 5.1 describes the invariants we fuzz-tested. The second column describes the invariant informally in English, and the third shows the total amount of compute time spent fuzzing this property. The last column notes indicates the number of bugs identified while fuzzing the invariant.

The Veridise auditors devoted a total of 8 compute-hours to fuzzing this protocol, identifying a total of 0 bugs.

Table 5.1: Invariants Fuzzed.

Specification	Invariant	Hours Fuzzed	Bugs Found
V-ASM-SPEC-001	ERC20.01: transfer should revert if a user atte...	8	0
V-ASM-SPEC-002	ERC20.02: Funds should be successfully transfer	8	0
V-ASM-SPEC-003	ERC20.03: static totalSupply	8	0
V-ASM-SPEC-004	ERC20.04: allowance/balances	8	0
V-ASM-SPEC-005	ERC20.05: funds transfer	8	0
V-ASM-SPEC-006	ERC20.06: no extra modfications	8	0
V-ASM-SPEC-007	ERC20.07: approve makes appropriate state chai	8	0
V-ASM-SPEC-008	ERC20.08: increaseAllowance correctness	8	0
V-ASM-SPEC-009	ERC20.09: decreaseAllowance correctness	8	0
V-ASM-SPEC-010	ERC20.10: burn reverts when expected	8	0
V-ASM-SPEC-011	ERC20.11: burn reduces balance	8	0
V-ASM-SPEC-012	ERC20.12: burn modifies expected state	8	0
V-ASM-SPEC-013	ERC20.13: burnFrom burns correct amount	8	0
V-ASM-SPEC-014	ERC20.14: burnFrom reduces total supply	8	0
V-ASM-SPEC-015	ERC20.15: no unintended side effects in burnFro	8	0
V-ASM-SPEC-016	ERC20.16: mint increases totalSupply/balance	8	0
V-ASM-SPEC-017	ERC20.17: mint has no bad side effects	8	0

5.3 Detailed Description of Fuzzed Specifications

5.3.1 V-ASM-SPEC-001: ERC20.01: transfer should revert if a user attempts to send more funds than they have

Minutes Fuzzed	60	Bugs Found	0

Specification

Scope ERC20 tokens.

Natural Language `transfer` should revert if a user attempts to send more funds than they have.

Formal This specification is part of the Veridise [V] specification library.

```
vars: ASM t
inv:  reverted(t.transfer(to, amt), amt > t.balanceOf(sender))
```

5.3.2 V-ASM-SPEC-002: ERC20.02: Funds should be successfully transferred from sender to to as long as sender != to

Minutes Fuzzed	60	Bugs Found	0

Specification

Scope ERC20 tokens.

Natural Language Funds should be successfully transferred from sender to to as long as `sender` $\neq$ `to`.

Formal This specification is part of the Veridise [V] specification library.

```
vars: ASM t
inv:  finished(t.transfer(to, amt),
  to != sender |=>
    t.balanceOf(sender) = old(t.balanceOf(sender)) - amt &&
    t.balanceOf(to) = old(t.balanceOf(to)) + amt
)
```

5.3.3 V-ASM-SPEC-003: ERC20.03: transfer should not modify totalSupply, allowances, or balances other than sender and to

Minutes Fuzzed	60	Bugs Found	0

Specification

Scope ERC20 tokens.

Natural Language `transfer` should not modify `totalSupply`, allowances, or balances other than `sender` and `to`.

Formal This specification is part of the Veridise [V] specification library.

```
vars: ASM t, address o1, address o2, address o3
inv:  finished(t.transfer(to, amt),
  o1 != sender && o1 != to |=>
    t.totalSupply() = old(t.totalSupply()) &&
    t.balanceOf(o1) = old(t.balanceOf(o1)) &&
    t.allowance(o2, o3) = old(t.allowance(o2, o3))
)
```

5.3.4 V-ASM-SPEC-004: ERC20.04: transferFrom should enforce allowance and user balance

Minutes Fuzzed	60	Bugs Found	0

Specification

Scope ERC20 tokens.

Natural Language `transferFrom` should enforce allowance and user balance.

`transferFrom` should revert when the amount requested is greater than what the spender owns or beyond the recipient's allowance.

Formal This specification is part of the Veridise [V] specification library.

```
vars: ASM t
inv:  reverted(t.transferFrom(from, to, amt),
  amt > t.balanceOf(from) || (from != sender && amt > t.allowance(from, sender))
)
```

5.3.5 V-ASM-SPEC-005: ERC20.05: As long as from != to, funds should be transferred from from to to and sender should have their allowance credited

Minutes Fuzzed	60	Bugs Found	0

Specification

Scope ERC20 tokens.

Natural Language As long as `from` ≠ `to`, funds should be transferred from `from` to `to` and `sender` should have their allowance credited.

Formal This specification is part of the Veridise [V] specification library.

```
vars: ASM t
inv:  finished(t.transferFrom(from, to, amt),
  from != to |=>
    t.balanceOf(from) = old(t.balanceOf(from)) - amt &&
    t.balanceOf(to) = old(t.balanceOf(to)) + amt &&
    ( (from != sender && old(t.allowance(from, sender)) != MAX_UINT256) ==>
      t.allowance(from, sender) = old(t.allowance(from, sender)) - amt )
)
```

5.3.6 V-ASM-SPEC-006: ERC20.06: transferFrom should not modify totalSupply, other allowances, or balances

Minutes Fuzzed	60	Bugs Found	0

Specification

Scope ERC20 tokens.

Natural Language `transferFrom` should not modify `totalSupply`, other allowances, or balances.

Formal This specification is part of the Veridise [V] specification library.

```
vars: ASM t, address o1, address o2, address o3, address o4
inv:  finished(t.transferFrom(from, to, amt),
  o1 != from && o1 != to && o2 != sender && o3 != from |=>
    t.balanceOf(o1) = old(t.balanceOf(o1)) &&
    t.allowance(from, o2) = old(t.allowance(from, o2)) &&
    t.allowance(o3, o4) = old(t.allowance(o3, o4)) &&
    t.totalSupply() = old(t.totalSupply())
)
```

5.3.7 V-ASM-SPEC-007: ERC20.07: approve makes appropriate state changes

Minutes Fuzzed	60	Bugs Found	0

Specification

Scope ERC20 tokens.

Natural Language `approve` makes appropriate state changes.

`approve` should never finish in a state where the allowance of the `spender` is not equal to the given `amount`. `totalSupply`, other allowances and balances should not be modified.

Formal This specification is part of the Veridise [V] specification library.

```
vars: ASM t, address o1, address o2, address o3
inv:  finished(t.approve(spender, amt),
  o2 != sender && o1 != spender |=>
    t.allowance(sender, spender) = amt &&
    t.allowance(sender, o1) = old(t.allowance(sender, o1)) &&
    t.allowance(o2, o3) = old(t.allowance(o2, o3)) &&
    t.balanceOf(o3) = old(t.balanceOf(o3)) &&
    t.totalSupply() = old(t.totalSupply())
)
```

5.3.8 V-ASM-SPEC-008: ERC20.08: increaseAllowances makes appropriate state changes

Minutes Fuzzed	60	Bugs Found	0

Specification

Scope ERC20 tokens.

Natural Language `increaseAllowances` makes appropriate state changes.

The function should increase a user's allowance by the indicated `amount`. `totalSupply`, other allowances, and balances should not be modified.

Formal This specification is part of the Veridise [V] specification library.

```
vars: ASM t, address o1, address o2, address o3
inv:  finished(t.increaseAllowance(spender, amt),
  o2 != sender && o1 != spender |=>
    t.allowance(sender, spender) = old(t.allowance(sender, spender)) + amt &&
    t.allowance(sender, o1) = old(t.allowance(sender, o1)) &&
    t.allowance(o2, o3) = old(t.allowance(o2, o3)) &&
    t.balanceOf(o3) = old(t.balanceOf(o3)) &&
    t.totalSupply() = old(t.totalSupply())
)
```

5.3.9 V-ASM-SPEC-009: ERC20.09: decreaseAllowance makes appropriate state changes

Minutes Fuzzed	60	Bugs Found	0

Specification

Scope ERC20 tokens.

Natural Language `decreaseAllowance` makes appropriate state changes.

The function should decrease a user's allowance by the indicated amount. `totalSupply`, other allowances, and balances should not be modified.

Formal This specification is part of the Veridise [V] specification library.

```
vars: ASM t, address o1, address o2, address o3
inv:  finished(t.decreaseAllowance(spender, amt),
  o2 != sender && o1 != spender |=>
    t.allowance(sender, spender) = old(t.allowance(sender, spender)) - amt &&
    t.allowance(sender, o1) = old(t.allowance(sender, o1)) &&
    t.allowance(o2, o3) = old(t.allowance(o2, o3)) &&
    t.balanceOf(o3) = old(t.balanceOf(o3)) &&
    t.totalSupply() = old(t.totalSupply())
)
```

5.3.10 V-ASM-SPEC-010: ERC20.10: burn will revert if a user attempts to burn more than they own

Minutes Fuzzed	60	Bugs Found	0

Specification

Scope ERC20 tokens.

Natural Language `burn` will revert if a user attempts to burn more than they own.

Formal This specification is part of the Veridise [V] specification library.

```
vars: ASM t
inv:  reverted(t.burn(amt), amt > t.balanceOf(sender))
```

5.3.11 V-ASM-SPEC-011: ERC20.11: burn will reduce the sender balance and the total supply by the indicated amount

Minutes Fuzzed	60	Bugs Found	0

Specification

Scope ERC20 tokens.

Natural Language `burn` will reduce a user's balance and the total supply by the indicated amount.

Formal This specification is part of the Veridise [V] specification library.

```
vars: ASM t
inv:  finished(t.burn(amt),
  t.balanceOf(sender) = old(t.balanceOf(sender)) - amt &&
  t.totalSupply() = old(t.totalSupply()) - amt
)
```

5.3.12 V-ASM-SPEC-012: ERC20.12: burn will not modify the balance of a user or any allowances

Minutes Fuzzed	60	Bugs Found	0

Specification

Scope ERC20 tokens.

Natural Language `burn` will not modify a user's balance or any allowances.

Formal This specification is part of the Veridise [V] specification library.

```
vars: ASM t, address o1, address o2, address o3
inv:  finished(t.burn(amt),
  o1 != sender |=>
    t.balanceOf(o1) = old(t.balanceOf(o1)) &&
    t.allowance(o2, o3) = old(t.allowance(o2, o3))
)
```

5.3.13 V-ASM-SPEC-013: ERC20.13: burnFrom will revert if a user attempts to burn more than they own or more than their allowance

Minutes Fuzzed	60	Bugs Found	0

Specification

Scope ERC20 tokens.

Natural Language `burnFrom` will revert if a user attempts to burn more than they own or more than their allowance.

Formal This specification is part of the Veridise [V] specification library.

```
vars: ASM t
inv:  reverted(t.burnFrom(from, amt),
  amt > t.balanceOf(from) || (from != sender && amt > t.allowance(from, sender))
)
```

5.3.14 V-ASM-SPEC-014: ERC20.14: burnFrom will reduce the total supply and the balance of from by the indicated amount

Minutes Fuzzed	60	Bugs Found	0

Specification

Scope ERC20 tokens.

Natural Language `burnFrom` will reduce the total supply and the balance of from by the indicated amount.

Formal This specification is part of the Veridise [V] specification library.

```
vars: ASM t
inv:  finished(t.burnFrom(from, amt),
  t.balanceOf(from) = old(t.balanceOf(from)) - amt &&
  t.totalSupply() = old(t.totalSupply()) - amt &&
  ( (from != sender || old(t.allowance(from, sender)) != MAX_UINT256) ==>
    t.allowance(from, sender) = old(t.allowance(from, sender)) - amt )
)
```

5.3.15 V-ASM-SPEC-015: ERC20.15: burnFrom will not modify another user balance or other allowances

Minutes Fuzzed	60	Bugs Found	0

Specification

Scope ERC20 tokens.

Natural Language `burnFrom` will not modify another user's balance or other allowances.

Formal This specification is part of the Veridise [V] specification library.

```
vars: ASM t, address o1, address o2, address o3
inv:  finished(t.burnFrom(from, amt),
  o1 != from && o2 != sender |=>
    t.balanceOf(o1) = old(t.balanceOf(o1)) &&
    t.allowance(from, o2) = old(t.allowance(from, o2)) &&
    t.allowance(o1, o3) = old(t.allowance(o1, o3))
)
```

5.3.16 V-ASM-SPEC-016: ERC20.16: mint will increase totalSupply and user balance by the indicated amount

Minutes Fuzzed	60	Bugs Found	0

Specification

Scope ERC20 tokens.

Natural Language `mint` will increase `totalSupply` and a user's balance by the indicated amount.

Formal This specification is part of the Veridise [V] specification library.

```
vars: ASM t
inv:  finished(t.mint(acc, amt),
  t.balanceOf(acc) = old(t.balanceOf(acc)) + amt &&
  t.totalSupply() = old(t.totalSupply()) + amt
)
```

5.3.17 V-ASM-SPEC-017: ERC20.17: mint will not modify the balance of another user or any allowances

Minutes Fuzzed	60	Bugs Found	0

Specification

Scope ERC20 tokens.

Natural Language `mint` will not modify another user's balance or any allowances.

Formal This specification is part of the Veridise [V] specification library.

```
vars: ASM t, address o1, address o2, address o3
inv:  finished(t.mint(acc, amt),
  o1 != acc |=>
    t.balanceOf(o1) = old(t.balanceOf(o1)) &&
    t.allowance(o2, o3) = old(t.allowance(o2, o3))
)
```

Glossary

ERC 20 The famous Ethereum fungible token standard. See `https://eips.ethereum.org/EIPS/eip-20` to learn more. 9

OpenZeppelin A security company which provides many standard implementations of common contract specifications. See `https://www.openzeppelin.com`. 1

smart contract A self-executing contract with the terms directly written into code. Hosted on a blockchain, it automatically enforces and executes the terms of an agreement between buyer and seller. Smart contracts are transparent, tamper-proof, and eliminate the need for intermediaries, making transactions more efficient and secure.. 27

Solidity The standard high-level language used to develop smart contracts on the Ethereum blockchain. See `https://docs.soliditylang.org/en/v0.8.19/` to learn more. 1