Seamoon Protocol 2

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Token Minter and Vesting Wallet
Timeline	2023-06-05 through 2023-06-08
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	• dm2c/token-contracts #6477dfd

Documentation quality	Medium
Test quality	High
Total Findings	6 Fixed: 1 Acknowledged: 5
High severity findings ⓘ	0
Medium severity findings ⓘ	0
Low severity findings ⓘ	2 Acknowledged: 2

Auditors	- Faycal Lalidji Senior Auditing Engineer - Jennifer Wu Auditing Engineer - Roman Rohleder Senior Auditing Engineer - Ruben Koch Auditing Engineer

Undetermined severity findings	1 Fixed: 1
Informational findings	3 Acknowledged: 3

Summary of Findings

Initial Audit

Through reviewing the code, we found **7 potential issues**. We recommend carefully re-considering the logic to ensure the safety of the users.

Fix Review

All highlighted issues have been solely acknowledged, with the exception of SEA-6 ("Unclear Re-Implementation of RestrictedVestingWallet._vestingSchedule()") which has been fixed by adding additional code documentation.

ID	DESCRIPTION	SEVERITY	STATUS
SEA-1	Critical Role Transfer Not Following Two-Step Pattern	● Low	Acknowledged
SEA-2	Privileged Roles and Ownership	● Low	Acknowledged
SEA-3	Missing Input Validation	● Informational	Acknowledged
SEA-4	Use of Unsafe Cast Operations	● Informational	Acknowledged

ID	DESCRIPTION	SEVERITY	STATUS
SEA-5	Unused Eth Vesting Functionality	● Informational	Acknowledged
SEA-6	Unclear Re-Implementation of `RestrictedVestingWallet._vestingSchedule()`	● Undetermined	Fixed

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

> **Disclaimer**
>
> Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities

- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

Files Included

Repo: https://github.com/dm2c/token-contracts(7ad1f6435bbb962328155eaf65e03be28c0fb9ff) Files:

- contracts/Minter.sol

- contracts/RestrictedVestingWallet.sol

Files Excluded

Repo: https://github.com/dm2c/token-contracts(7ad1f6435bbb962328155eaf65e03be28c0fb9ff) Files:

- contracts/DM2E.sol
- contracts/DM2P.sol

Findings

SEA-1
Critical Role Transfer Not Following Two-Step Pattern

● Low ⓘ Acknowledged

> **Update**
>
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> > In our case, even if the ownership of the contract is moved to an unintended address, it is recoverable by the re-deployment of the contract by the operator. With the 2-Step Pattern you suggested, we would have to handle the complexity of the process and additional gas fees, so we decide to keep the simple `transferOwnership()` function.

File(s) affected: `Minter.sol`

Description: The `owner` role can be transferred to another address simply by calling the `Minter.transferOwnership()` function. This function immediately transfers a high-level privilege to new addresses in a single transaction, which can be risky from a security perspective, as providing a faulty address may lock out that role from future calls.

A more secure pattern for such privilege transfers is to require the new pending addresses to issue an `acceptAdmin()` function call before finalizing the transfer.

Recommendation: Ensure that before transferring an authority, the new account calls the `acceptAdmin()` method to accept the role. Consider using the OpenZeppelin's Ownable2Step library

SEA-2 Privileged Roles and Ownership

● Low ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> We will not modify the contract, but we will include a note in the token distribution documentation for end users.

File(s) affected: `Minter.sol`

Description: Certain contracts have state variables, e.g. `owner` , which provide certain addresses with privileged roles. Such roles may pose a risk to end-users.

The `Minter.sol` contract contains the following privileged roles:

1. An owner (`_owner` , `onlyOwner()` modifier), as initialized during the constructor execution to `_msgSender()` :

    1. Renounce the role (**and thereby prevent any future calls to the following listed functions!**) by calling `renounceOwnership()` (or `transferOwnership()` to an uncontrolled address).
    2. Transfer ownership to an arbitrary address by calling `transferOwnership()` .
    3. Mint the predefined ERC20 token up to the according minting schedule for the given beneficiary and create a restricted vesting wallet (`RestrictedVestingWallet.sol`) for him by calling `mint()` .

    A compromised owner could:

    1. **Deny the service to this contract and thereby to the ability to mint any new tokens by calling** `rennounceOwnership()` **,** `transferOwnership()` **to an uncontrolled address or simply by not calling** `mint()` .
    2. **(Re-)Mint the currently mintable amount according to the defined minting schedule to oneself**.

Recommendation: Clarify the impact of these privileged actions to the end-users via publicly facing documentation.

SEA-3 Missing Input Validation

● Informational ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> It is expected that the value of each duration can take 0. On the other hand, in relation to SEA-4, it would be useful to validate that these values do not exceed type(uint64).max. However, as with the reason for SEA-4, we have determined that this is avoidable in operation.

File(s) affected: `Minter.sol`

Related Issue(s): SWC-123

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error. In `Minter.constructor()`, `_mintingDuration`, `_lockingDuration`, and `_vestingDuration` are not validated.

Recommendation: We recommend adding the relevant checks.

SEA-4 Use of Unsafe Cast Operations

● Informational ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> We do not expect to use a range of values for each duration that exceeds type(uint64).max, and even if an incorrect value is set, it will not be activated unless the minter is granted MINTER_ROLE by the token contract owner, so we have determined that this can be avoided in operation.

File(s) affected: `Minter.sol`

Description: The following functions make use of unsafe cast operations (`uint64()`), exposing them to truncation, when reaching values higher than `type(uint64).max` (`18,446,744,073,709,551,615` or as an interpreted timestamp `07/21/2554 23:34:33.709`):

1. `Minter.sol#L104` : `uint64(block.timestamp + lockingDuration),` .
2. `Minter.sol#L105` : `uint64(vestingDuration)` .

Recommendation: We recommend using a safe cast library and replacing the unsafe cast operations with the corresponding safe variant.

SEA-5 Unused Eth Vesting Functionality

● Informational ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> We are considering the operation of bridging DM2P and DM2E tokens to multi chains, possibly implemented as native tokens on a Layer 2 private chain. It would also be less confusing to match the behavior with the inherited VestingWallet. Therefore, no modifications were made.

File(s) affected: `RestrictedVestingWallet.sol`

Description: Contract `RestrictedVestingWallet.sol` inherits from OpenZeppelins `VestingWallet` and extends it by overriding the vesting release functions to be only callable by the beneficiary. This includes functions:

- `release()` : Transfer any available and vested Eth to the beneficiary.
- `release(address token)` : Transfer any available and vested ERC-20 `token` to the beneficiary.

According to the provided specification/documentation, only the `ERC–20 vesting functionality is planned to be used (for tokens` DM2E and DM2P`).

Recommendation: Consider removing the unused Eth vesting functionality to save on deployment gas and reduce complexity by overriding the relevant functions with an empty body.

SEA-6

Unclear Re-Implementation of `RestrictedVestingWallet._vestingSchedule()`

● Undetermined ⓘ Fixed

> **Update**
>
> Marked as "Fixed" by the client. Addressed in: `419a62f0a381e6b16e7d94cbdf76ba2d9b21ca61` . The client provided the following explanation:
>
> > The purpose of this reimplementation is to prevent zero-division errors when the duration is zero. We have included that comment in the code. Also, `VestingWallet.end()` was not used because it is the latest specification with no fixed version.

File(s) affected: `RestrictedVestingWallet.sol`

Description: It is unclear why `RestrictedVestingWallet._vestingSchedule()` has been reimplemented, considering that the function `VestingWallet.end()` , which is used in `VestingWallet._vestingSchedule()` , already returns the value `start()`

`+ duration()` . The reimplemented function seems to replicate the exact value that is already provided by the existing function except for the change in the greater than or equal operator.

Recommendation: To clarify this issue, it is important to clearly indicate whether this reimplemented version is intentional and aligned with the specifications. If it deviates from the specifications, it should be fixed accordingly.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Code Documentation

1. Missing or incorrect NatSpec comments:
    1. `Minter.mintableAmount()` : Missing NatSpec comment for the return value.
    2. `RestrictedVestingWallet.constructor()` : Missing NatSpec comments for parameters `beneficiaryAddress` , `startTimestamp and durationSeconds` .
    3. `RestrictedVestingWallet._vestingSchedule()` : Missing NatSpec comments for parameters `totalAllocation` , `timestamp` and the return value.

Adherence to Best Practices

1. To facilitate logging it is recommended to index address parameters within events. Therefore the `indexed` keyword should be added to the (other) address parameters in
    1. `Minter.Mint()` ,
2. As a best practice, if state variables are only initialized in the constructor, they should be declared as `immutable` . This modification can lead to improved execution efficiency and reduced gas costs, as `immutable` variables are directly embedded in the contract's bytecode, thus lowering gas usage during interactions. Consider changing the following state variables to immutable:
    1. `Minter.vestingDuration`
    2. `Minter.lockingDuration`
    3. `Minter.mintStart`
    4. `Minter.capAmount`
3. Consider using custom errors in place of `require` statements for gas savings.

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

- `fc8...d8d ./contracts/RestrictedVestingWallet.sol`
- `bd9...4b7 ./contracts/Minter.sol`

Tests

- `08f...350 ./tests/RestrictedVestingWallet.test.ts`
- `105...049 ./tests/Minter.test.ts`

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.9.3

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

Slither did not report any significant findings.

Test Suite Results

```
testing for DM2E
  Deployment
    ✔ Should assign the total supply of tokens to the owner
  Transactions
    ✔ Should transfer tokens between accounts (94ms)
    ✔ Should fail if sender doesn’t have enough tokens (105ms)
    ✔ Should update balances after transfers (70ms)
  Mint
    ✔ Should mint initial supplies correctly
    ✔ Should allow admin to mint
    ✔ Should fail to mint when users other than admin signs
  pause
    ✔ Should allow admin to paused and unpaused (115ms)
    ✔ Should fail when pause by non-admin
    ✔ Should fail when unpause by non-admin
  burn
    ✔ Should allow burn by admin (42ms)
    ✔ Should fail when burn by non-admin
    ✔ Should allow burnFrom by admin (93ms)
    ✔ Should fail when burnFrom by non-admin (42ms)
    ✔ Should fail when exceeds the approve
  AccessControl
    ✔ Should grant initial DEFAULT_ADMIN_ROLE correctly
    ✔ Should allow admin to grant role (85ms)
    ✔ Should fail when grant role by non-admin
    ✔ Should allow admin to revoke role (65ms)
    ✔ Should allow admin to revoke role (101ms)
    ✔ Should fail when revokeRole DEFAULT_ADMIN_ROLE by last admin
    ✔ Should fail when renounceRole DEFAULT_ADMIN_ROLE by last admin
    ✔ Should fail when revokeRole MINTER_ROLE by last admin
    ✔ Should fail when renounceRole MINTER_ROLE by last admin

testing for DM2P
  Deployment
    ✔ Should assign the total supply of tokens to the owner
  Transactions
    ✔ Should transfer tokens between accounts (62ms)
    ✔ Should fail if sender doesn’t have enough tokens
    ✔ Should update balances after transfers (52ms)
  Mint
    ✔ Should mint initial supplies correctly
    ✔ Shoud set cap correctly
    ✔ Should allow admin to mint
    ✔ Should fail to mint when users other than admin signs
    ✔ Should fail when exceeds the cap
  pause
    ✔ Should allow admin to paused and unpaused (94ms)
    ✔ Should fail when pause by non-admin
    ✔ Should fail when unpause by non-admin
  burn
    ✔ Should allow burn by admin (40ms)
    ✔ Should fail when burn by non-admin
    ✔ Should allow burnFrom by admin (78ms)
    ✔ Should fail when burnFrom by non-admin (47ms)
    ✔ Should fail when exceeds the approve (42ms)
  AccessControl
    ✔ Should grant initial DEFAULT_ADMIN_ROLE correctly
    ✔ Should allow admin to grant role (74ms)
    ✔ Should fail when grant role by non-admin
    ✔ Should allow admin to revoke role (71ms)
```

```
      ✔ Should allow admin to revoke role (98ms)
      ✔ Should fail when revokeRole DEFAULT_ADMIN_ROLE by last admin
      ✔ Should fail when renounceRole DEFAULT_ADMIN_ROLE by last admin
      ✔ Should fail when revokeRole MINTER_ROLE by last admin
      ✔ Should fail when renounceRole MINTER_ROLE by last admin

  testing for Minter
    ✔ scenario (692ms)
    constructor
      success
        ✔ set token address
        ✔ set capAmount state
        ✔ set mintStart state
        ✔ set mintingDuration state
        ✔ set lockingDuration state
        ✔ set vestingDuration state
        - gas cost
        ✔ even if vestingDuration is zero, it does not throw an error (68ms)
      errors
        ✔ if token address is zero, it throws an error
        ✔ if capAmount is zero, it throws an error
        ✔ if mintStart is zero, it throws an error
    mint
      success
        ✔ emits Mint event
        ✔ deploy new vesting wallet
        ✔ mint tokens to vesting wallet
        ✔ beneficiary of new vesting wallet is equal to given arg
        ✔ start timestamp of new vesting wallet is equal to current time + locking duration
        ✔ if time is after mintStart, it can mint tokens (143ms)
        ✔ if minting duration is passed, all token is mintable (111ms)
      error
        ✔ only owner can mint
        ✔ if mintStart is future, it throws an error (104ms)
        ✔ if beneficiary is zero, it throws an error
        ✔ if amount is zero, it throws an error
        ✔ if minting amount is greater than mintable amount, it throws an error on first minting
        ✔ if minting amount is greater than minted + mintable amount, it throws an error (60ms)
        ✔ if minted amount is equal to capAmount, it throws an error (53ms)

  VestingWallet
    release for ETH
      ✔ only beneficiary can withdraw
      ✔ non beneficiary can not withdraw
    withdraw
      ✔ only beneficiary can withdraw (103ms)
      ✔ beneficiary can withdraw according to vestingSchedule (152ms)
      ✔ beneficiary cannot withdraw before start() (117ms)
      ✔ beneficiary can withdraw after start() + duration() (120ms)
      ✔ duration is zero (112ms)


  82 passing (25s)
```

Code Coverage

Quantstamp usually recommends developers increase the branch coverage to `90%` and above before a project goes live, in order to avoid hidden functional bugs that might not be easy to spot during the development phase. For branch code coverage, the current targeted files by the audit achieve a high score.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	100	100	100	
DM2E.sol	100	100	100	100	
DM2P.sol	100	100	100	100	
Minter.sol	100	100	100	100	
RestrictedVestingWallet.sol	100	100	100	100	
All files	100	100	100	100	

Changelog

- 2023-06-08 - Initial report
- 2023-07-04 - Fix review (5d3e01b)
- 2023-07-10 - Code coverage update (5d3e01b)

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that your access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

Quantstamp