Status Security Analysis Report and Formal Verification Properties

x

10.10.2023

Table of Contents

Summary

This document describes the specification and verification of the new **Status SNT Optimism Bridge and MiniMeToken implementation** using the Certora Prover and manual code review findings. The work was undertaken from **28th August 2023** to **10th October 2023**. The latest commits that was reviewed manually and run through the Certora Prover are 6d9d4f5487 and 74f3fd88a5.

The following contracts list is included in the **scope**:

optimism-bridge-snt Repo:

`contracts/optimism/IOptimismMintableERC20.sol`
`contracts/optimism/OptimismMintableMiniMeToken.sol`
`contracts/optimism/Semver.sol`
`contracts/SNTOptimismController.sol`

minime Repo:

`contracts/ApproveAndCallFallBack.sol`
`contracts/Controlled.sol`
`contracts/MiniMeBase.sol`
`contracts/MiniMeToken.sol`
`contracts/Nonces.sol`
`contracts/TokenController.sol`

The contracts are written in Solidity ^0.8.0.

The Certora Prover demonstrated that the implementation of the Solidity contracts above is correct with respect to the formal rules written by the Certora team. In addition, the team performed a manual audit of all Solidity contracts. During the verification process and the manual audit, the Certora Prover discovered bugs in the Solidity contracts code, as listed below.

Summary of findings

The table below summarizes the issues discovered during the audit, categorized by severity.

Severity	Total discovered	Total fixed	Total acknowledged
High	2	2	2
Medium	2	2	2
Low	2	2	2
Informational	7	5	5
Centralization	1	0	1
Total (High, Medium, Low)	**14**	**11**	**14**

Disclaimer

The Certora Prover takes a contract and a specification as input and formally proves that the contract satisfies the specification in all scenarios. Notably, the guarantees of the Certora Prover are scoped to the provided specification and the Certora Prover does not check any cases not covered by the specification.

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora 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.

Main Issues Discovered

High-01: Upgradability of L1&L2 miniMe token pair can be compromised

Severity: High
Probability: High
Category: Double spend, Operational consideration
File(s): MiniMeToken.sol
Bug description: If both L1 and L2 are MiniMeTokens, then using its upgradability (cloning) feature might compromise the token if not executed exactly at the same time. This is due to the fact that a malicious user could attempt to double their funds by bridging from the old token to the not-yet-cloned one.
Exploit scenario:
A DAO vote passes on both chains to make an upgrade/clone of the current SNT (miniMe) tokens. This could naively be done by calling each one's `createCloneToken()` function.
There would be some delay (and thus parentSnapshotBlock) for the cloned tokens on L1 and L2 - let's call that the "upgrade window". In this example, let's say L1 upgraded first.
An attacker might try to time or front-run a bridging action during that window. So as soon as L1 is cloned, they'll bridge their entire balance from the old L1 token. If this bridging action happened before the upgrade window closed, the attacker's balance on the old L2 token would increase by the same amount he already got on L1, doubling their funds.
Status's response: *Clone feature would not be used for upgradability.*

Will be addresed. Original MiniMeToken would still have createToken/destroyToken operated by Controller only, but OptimismSNT would have mint/burn operated by bridge only.

This was done by abstracting out the generateToken and destoyToken here https://github.com/vacp2p/minime/pull/32

Which now is used by https://github.com/logos-co/optimism-bridge-snt/ and does not implement those functions on the optimism token.

High-02: Reentrancy danger on controller's onTransfer() callback

Severity: Critical
Probability: Low
Category: Reentrancy, double spend
File(s): MiniMeToken.sol
Bug description: The check-effects-interactions pattern was ignored during `MiniMeToken.doTransfer()`, where after making a callback it uses stale balances during the updates that come afterward.
Although the controller is a trusted entity of the system (and current implementation isn't vulnerable), This can lead to needless risk for future or more complex controller implementations (as leveraging the reentrancy does not require any special permissions, just the execution time at that moment).
Another way to look at this is that there's an assumption of the system that the controller must not give arbitrary 3rd party execution. Currently, this assumption has to be enforced on every controller implementation.

Exploit scenario / Numerical Example:
Let's assume a non-malicious controller that can give another callback to the user during `onTransfer()` call.
When an attacker transfers some X amount of funds in a way that gets the 2nd callback, they can just do another Transfer for another X amount, and their balance would only decrease by X.
That's because in the first call, the MiniMeToken already checked the balance requirements and calculated the balanceBefore but didn't send it yet. Making a reentrant call here would double spend the sent amount.

Status's response: *The issue considers that is possible if Controller is compromised, however, the Controller compromised would cause bigger problems than this. Anyway, its true that the reentrancy is possible. In this case, it would cause totalSupply to be lower than actual totalSupply. A test case was created for it, and a fix was implemented.*

Medium-01: `claimTokens()` can't rescue locked native currency

Severity: Medium
Category: Locked funds, Missing logic
File(s): SNTPlaceHolder.sol
Bug description: SNTPlaceHolder is (seemingly) supposed to be SNT's controller. But it doesn't have a fallback or receive function, making it unable to accept any native currency sent by `MiniMeToken.sol:rescueFunds()`.
Status's response: acknowledged and **fixed**:
https://github.com/logos-co/optimism-bridge-snt/pull/11

Medium-02: Difference between calculated and stored uint sizes can lead to overflows

Severity: Medium
Category: Overflows (casting)
File(s): MiniMeToken.sol
Bug description: All balances in MiniMeToken are saved as `uint128` in the checkpoints (as well as total supplies). Since all inputs and calculations are done with `uint256`, there were no checks for the `uin128` size thus allowing overflows.
While on most tokens such amounts are unrealistic (config dependant on the decimals), this might also open up an unintended centralization risk that could've been avoided (minter can deliberately zero out someone's balance - or even the total supply - just by minting, which shouldn't be expected).
Current overflow checks are all on `uint256` size, so they don't circumvent this potential issue.
Status's response: acknowledged and **fixed**:
https://github.com/vacp2p/minime/pull/35
Checking were already done for mint and transfer, see

- *Generate Token:*
  - *totalSupply overflow: https://github.com/logos-co/optimism-bridge-snt/blob/2cd0c8f5be303c2111010f53c8281e709fab94c2/contracts/token/MiniMeToken.sol#L264*
  - *balance overflow: https://github.com/logos-co/optimism-bridge-snt/blob/2cd0c8f5be303c2111010f53c8281e709fab94c2/contracts/token/MiniMeToken.sol#L266*
- *Transfer:*
  - *balance https://github.com/logos-co/optimism-bridge-snt/blob/2cd0c8f5be303c2111010f53c8281e709fab94c2/contracts/token/MiniMeToken.sol#L222*

However these checks were not sufficient because they were happeinng in uint256.
With further analysis, we determined that the balance checks are impossible to be reached, because if the totalSupply is checked, is mathematically impossible to overflow any account balance. So we removed the balance overflow checks to save on gas. This is also done by OpenZeppelin standard ERC20 implementation.

Low-01: Ownership cannot be revoked/petrified

Severity: Low
Probability: 100%
Category: Missing Logic
File(s): Owned.sol
Bug description: Documentation states the admin can revoke their rights by setting the owner to address `0x0`. But Ownership must be accepted by the receiver to take effect, so there needs to be special logic where when the receiver is `0x0`, it can also be set as the current owner.
Status's response:

This is by design, however the documentation should be fixed.

Low-02: MiniMeTokenCore's createCloneToken would create a MiniMeToken

Severity: Low
Category: Unexpected Behavior
File(s): MiniMeTokenCore.sol, MiniMeTokenFactory.sol
Bug description: The MiniMetokenCore.sol has no GenerateToken or DestroyToken functions, but calling its createClone would actually create an instance of MiniMeToken, where it does have that functionality. This seems unexpected.
Status's response: *This is intended, because the cloneTokens should be manageable by whoever cloned them. Only the bridge token does not have a generate/destroy tokens by controller, because the generate and destroy tokens must be done by bridge only.*

With the new changes, createCloneToken would not exist anymore, but this functionality would still be possible by creating a new MiniMeToken referencing parentToken and parentSnapshotBlock

Informational-01: Deprecated library used for Solidity >= 0.8: SafeMath

Severity: Informational
Category: Gas
File(s): safeMath.sol
Bug description: Solidity compiler version 0.8 and above inserts overflow checks automatically.
The usage of the safeMath.sol library seems completely redundant, unlike the original MiniMeToken.sol material that was on version 0.4.X and did need that.
This is also true for some overflow checks on MiniMeToken.sol.
Status's response: *This library was zombie code from SNT ICO code. This is fixed*

Informational-02: ParentToken Mutability

Severity: Informational
Category: missing keyword
File(s): MiniMeToken.sol
Bug description: The parent of a MiniMeToken must not change in order to keep the integrity of the balance history. To avoid future implementation mistakes, add this keyword or documentation.
This might also apply to:
`MiniMeToken.creationBlock`
`MiniMeToken.decimals`
`MiniMeToken.parentSnapShotBlock`
`MiniMeToken.parentToken`
`MiniMeToken.tokenFactory`
Status's response: acknowledged and **fixed**:
https://github.com/vacp2p/minime/pull/23

Informational-03: Excess Functionality - Controller power

Severity: Informational
Category: Excess Functionality
File(s): MiniMeToken.sol
Bug description: If the main design for the L2 SNT token is to be an L2 token, it could be enough for the bridge to be the only entity to be able to mint and burn tokens arbitrarily.
Status's response: *We are currently looking into removing excess functionality from MiniMeBase.sol*

Informational-04: Excess Functionality - Cloning Logic

Severity: Informational
Category: Excess Functionality
File(s): MiniMeToken.sol
Bug description: If the main design for the SNT tokens beyond ERC20 is to solely retain history (by the checkpoints mechanism), and upgradability is not at all desired, then the MiniMeToken's standard has a lot of excess logic that is not necessarily desired (e.g., the cloning logic, the controller, etc).
This can also be considered against other upgradability patterns if desired.
Status's response: *We are currently looking into removing excess functionality from MiniMeBase.sol*
cloning logic is already removed, but still possible outside the contract.

Informational-05: Excess Functionality - `CreateCloneToken()`

Severity: Informational
Category: Excess Functionality
File(s): MiniMeToken.sol
Bug description: Creating a clone of a token seems to be an external process to the Token itself. Understandably this is there for 'usability', but it also be implemented in a separate contract. There is no need for the Token to know its factory, and in any Upgrade that is beyond just a clone, there would need to be a different factory anyway.
Status's response: acknowledged and **fixed**:
https://github.com/vacp2p/minime/pull/34
https://github.com/vacp2p/minime/pull/39

Informational-06: Unused constant

Severity: Informational
Category: CR
File(s): Controlled.sol
Bug description: `ERR_UNAUTHORIZED` variable left unused, probably should've been used in `L#11`.
Status's response: *This was fixed when MiniMeToken contracts were refactored from source to new foundry template, and started using the new error type from solidity.*

Informational-07: `EnableTransfers()` should emit an event

Severity: Informational
Category: Suggestion
File(s): MiniMeToken.sol
Bug description: `transfersEnabled` variable is an important configuration parameter of the system, consider emitting an event when this is changed to let the users, trackers and monitors know.

Centralization-01: Controller Owner

Severity: N/A
Category: Centralization
File(s): SNTPlaceHolder.sol
Bug description: The MiniMeToken has several permissionless perks, but currently the entire system can be compromised by a single entity which is the owner of `SNTPlaceHolder` (current controller).
That's because it can change the SNT's controller to an arbitrary address, and the controller can mint infinite/burn by any amount.
Use most safeguards, or even configure it to be the DAO itself.
Consider issue Info-03, to remove unnecessary power from the Controller entity.
Status's response: *We are currently looking into reducing the power of controller in Optimism token.*

Formal Verification Process

The structure of properties:

1. <notation> <property description> (<*property name in spec code*>)
    - <property specific assumptions>

Function names (and signatures) shall be written in Source code Pro font size 11 with the grey highlight, e.g., `foo(uint256)`

Notations

✅Indicates the rule is formally verified.
❌Indicates the rule is violated.
⏳Indicates the rule is timing out.

MiniMe Token properties

Assumptions

- Loop unrolling: We assume any loop can have at most 1 iteration.
- The properties of the MiniMe Token are proven for token that was cloned (with a parent token) and for tokens that are not (parent token's address is 0x0).
- External calls to contracts that aren't implemented or part of the repositories are summarized as non-deterministic yet valid. Such as call to `ApproveAndCallFallBack, TokenController and external ERC20`

Properties

1. ✅ Each checkpoint.fromBlock must be less than the current block number - no checkpoints from the future. (*checkPointBlockNumberValidity*).
2. ✅ checkpoint.fromBlock is monotonically increasing for each user. (*blockNumberMonotonicInc*)
3. ✅ All block numbers are greater or equal to the CreationBlock. (*allBlockNumbersAreGreaterOrEqualToCreationBlock*).
4. ✅ All block numbers are greater or equal to the ParentSnapShotBlock. (*allFromBlockAreGreaterThanParentSnapShotBlock*).
5. ✅ The balance of each user must be less or equal to the total supply (*balanceOfLessOrEqToTotalSpply*).
6. ✅ Balance of address 0 is always 0 (*ZeroAddressNoBalance*).
7. ✅ Cant change balances and totalSupply history (*historyMutability*).

8. ✅ Verify that there is no fee on transferFrom() (like potentially on USDT) (*noFeeOnTransferFrom*).
9. ✅ Verify that there is no fee on transfer() (like potentially on USDT) (*noFeeOnTransfer*).
10. ✅ Token transfer() works correctly. Balances are updated if returns true. Else, transfer amount was too high, or the recipient is 0. (*transferCorrect*).
11. ✅ Token transferFrom() works correctly. Balances are updated if returns true. Else, transfer amount was too high, or the recipient is 0. (*transferFromCorrect*).
12. ✅ transferFrom should revert if and only if the amount is too high or the recipient is 0 or transfer is not enabled (if the was not made by the controller) or if the blocknumber is not valid (*transferFromReverts*).
13. ✅ Contract calls don't change the token total supply. Except minting and burning functions ().
14. ✅ Test that generateTokens works correctly. Balances and totalSupply are updated corrct according to the paramenters (*integrityOfGenerateTokens*).
15. ✅ Test that destroyTokens works correctly. Balances and totalSupply are updated corrct according to the paramenters (*integrityOfdestroyTokens*).
16. ✅ Transfer from a to b using transfer doesn't change the balance of other addresses (*TransferDoesntChangeOtherBalances*).
17. ✅ Transfer from a to b using transferFrom doesn't change the balance of other addresses (*TransferFromDoesntChangeOtherBalances*).
18. ✅ Allowance changes correctly as a result of calls to approve, approveAndCall, transferFrom and permit.