Omni Network

Nomina Bridge Contracts

Security Assessment Report

Version: 2.0

September, 2025

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Omni Network components in scope. The review focused solely on the security aspects of the Solidity implementation of the contract, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the components in scope. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the Omni Network components contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Omni Network components in scope.

Overview

This codebase primarily concerns updating the ERC20 representation native token on Ethereum to Nomina from Omni. This involves replacing the Omni token with the Nomina token, with the Ethereum side of the bridge converting all Omni tokens to Nomina at a 75x redenomination rate during initialisation.

Security Assessment Summary

Scope

The review was conducted on the files hosted on the omni repository.

The scope of this time-boxed review was strictly limited to files at commit 5abe64e. The fixes of the identified issues were assessed at commit a849c79.

- `WNomina.sol`
- `NominaBridgeNative.sol`
- `NominaBridgeCommon.sol`
- `NominaBridgeL1.sol`

Note: third party libraries and dependencies were excluded from the scope of this assessment.

Approach

The security assessment covered components written in Solidity.

For the Solidity components, the manual review focused on identifying issues associated with the business logic implementation of the contracts. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout).

Additionally, the manual review process focused on identifying vulnerabilities related to known Solidity anti-patterns and attack vectors, such as re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers.

For a more detailed, but non-exhaustive list of examined vectors, see [1, 2].

To support the Solidity components of the review, the testing team may use the following automated testing tools:

- Aderyn: `https://github.com/Cyfrin/aderyn`
- Slither: `https://github.com/trailofbits/slither`
- Mythril: `https://github.com/ConsenSys/mythril`

Output for these automated tools is available upon request.

Coverage Limitations

Due to the time-boxed nature of this review, all documented vulnerabilities reflect best effort within the allotted, limited engagement time. As such, Sigma Prime recommends to further investigate areas of the code, and any related functionality, where majority of critical and high risk vulnerabilities were identified.

Findings Summary

The testing team identified a total of 6 issues during this assessment. Categorised by their severity:

- Medium: 2 issues.
- Low: 1 issue.
- Informational: 3 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Omni Network components in scope. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the components, including optimisations, are also described in this section and are labelled as "informational".

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected components(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
OMNB-01	Gas Limit Issue In `withdraw()` May Lead To Stuck Of Funds	Medium	Resolved
OMNB-02	Desynchronized Pausing Between Nomina Bridges And Omni Portal Can Lock Funds	Medium	Closed
OMNB-03	Delay In Calling `setup()` Could Result In Accounting Errors	Low	Closed
OMNB-04	Non-Standard Naming Convention And Unnecessary Caching Of Immutable Variable	Informational	Resolved
OMNB-05	Missing Validation In Constructor Could Render `withdraw()` Inoperable	Informational	Closed
OMNB-06	Miscellaneous General Comments	Informational	Resolved

OMNB-01	Gas Limit Issue In `withdraw()` May Lead To Stuck Of Funds		
Asset	`NominaBridgeNative.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

The portal allocates `XCALL_WITHDRAW_GAS_LIMIT = 80_000` gas for the `withdraw()` function call, as shown here:

NominaBridgeL1.sol::_bridge()

```
portal.xcall{ value: msg.value }(
    nominaChainId, ConfLevel.Finalized, Predeploys.NominaBridgeNative, xcalldata, XCALL_WITHDRAW_GAS_LIMIT
);
```

In the `NominaBridgeNative` contract, simulations indicate that lines [**104-111**] consume approximately 28,000 gas, leaving ~52,000 gas available (assuming the full 80,000 is available at line [**104**]).

NominaBridgeNative.sol::withdraw()

```
function withdraw(address payor, address to, uint256 amount) external whenNotPaused(ACTION_WITHDRAW) {
    XTypes.MsgContext memory xmsg = portal.xmsg();

    require(msg.sender == address(portal), "NominaBridge: not xcall"); // this protects against reentrancy
    require(xmsg.sender == l1Bridge, "NominaBridge: not bridge");
    require(xmsg.sourceChainId == l1ChainId, "NominaBridge: not L1");

    l1Deposits += amount;

    (bool success,) = to.call{ value: amount }("");

    if (!success) claimable[payor] += amount;

    emit Withdraw(payor, to, amount, success);
}
```

Due to EIP-150, 63/64 of the remaining 52,000 gas (~51,187) is forwarded to the `to` address. If the `to` contract's fallback/receive functions consume all forwarded gas (e.g., by modifying storage), only ~812 gas remains (1/64 of 52,000). Since the call fails with an out-of-gas error, the if-block executes:

NominaBridgeNative.sol::withdraw()

```
if (!success) claimable[payor] += amount;
```

However, this remaining gas is insufficient for a storage update, causing a revert.

Because portal message delivery is non-retriable and indifferent to call outcomes, funds can become permanently stuck.

Recommendations

Consider the mitigations.

1. Increase the gas limit. Assuming line [**114**] consumes 22,000 gas, `XCALL_WITHDRAW_GAS_LIMIT` should satisfy:

```
(1/64) * (XCALL_WITHDRAW_GAS_LIMIT - 28,000) > 22,000
=> XCALL_WITHDRAW_GAS_LIMIT > 1,436,000
```

This high value may be impractical due to fee constraints.

2. Reserve gas for the fallback, by explicitly reserving gas for the failure case. Note there should be more than 22,000 gas left.

```
(bool success,) = to.call{ value: amount, gas: gasleft() - 22,000 }("");
```

Resolution

This issue has been addressed in the commit a849c79 by allocating 32k gas as buffer.

OMNB-02	Desynchronized Pausing Between Nomina Bridges And Omni Portal Can Lock Funds		
Asset	`NominaBridgeL1.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

The pausing mechanism in Nomina bridges (`NominaBridgeL1` and `NominaBridgeNative`) operates separately from the pausing mechanism in the Omni portal. This creates a scenario where `ACTION_WITHDRAW` may be paused in a Nomina bridge, but the corresponding `ActionXSubmit` in the portal remains active.

The `NominaBridgeL1.withdraw()` function includes pausing protection:

NominaBridgeL1.sol::withdraw()

```
function withdraw(address to, uint256 amount) external whenNotPaused(ACTION_WITHDRAW) {
    XTypes.MsgContext memory xmsg = portal.xmsg();

    require(msg.sender == address(portal), "NominaBridge: not xcall");
    require(xmsg.sender == Predeploys.NominaBridgeNative, "NominaBridge: not bridge");
    require(xmsg.sourceChainId == portal.omniChainId(), "NominaBridge: not omni portal");

    nomina.transfer(to, amount);
    emit Withdraw(to, amount);
}
```

The bridge pausing is controlled independently:

NominaBridgeCommon.sol::pause()

```
/// @notice Pausable key for withdraws
bytes32 public constant ACTION_WITHDRAW = keccak256("withdraw");

/// @notice Pause `action`
function pause(bytes32 action) external onlyOwner {
    _pause(action);
}

/// @notice Revert if `action` is paused
modifier whenNotPaused(bytes32 action) {
    require(!_isPaused(action), "NominaBridge: paused");
    _;
}
```

However, the Omni Portal has separate pausing controls for `ActionXSubmit` :

OmniPortal.sol::pauseXSubmit()

```
function pauseXSubmit() external onlyOwner {
    _pause(ActionXSubmit);
    emit XSubmitPaused();
}

function pauseXSubmitFrom(uint64 chainId_) external onlyOwner {
    _pause(_chainActionId(ActionXSubmit, chainId_));
    emit XSubmitFromPaused(chainId_);
}
```

For example, during an incident, if `ACTION_WITHDRAW` is paused in `NominaBridgeL1` but the Omni portal `ActionXSubmit` is not paused (or there is a delay), calling `xSubmit()` on `OmniPortal` for an inflight message will attempt to call `withdraw()` on `NominaBridgeL1`. As the bridge is paused, this call will revert. Since the Omni Portal architecture does not handle call results and does not allow retries, the withdrawal becomes un-retriable, leading to funds being stuck.

Recommendations

It is recommended to enforce synchronisation between the Nomina bridge and the Omni portal. This can be achieved by ensuring the portal for the source chain is paused whenever the bridge is paused, or by introducing a centralised pausing mechanism to manage both in sync.

Resolution

The development team has closed the issue with the following comment:

> *We are not going to implement changes for OMNB-02 because pausing withdrawals is not a mechanism we intend to use. It is merely an emergency lever we can pull in the event of a crisis, if we want to. If the messaging layer were attacked, we would start by pausing the portals, rather than the bridge itself. Having the portals centralize pausing of the bridges is unnecessary complexity, and ensuring that the portal is paused before pausing withdrawals on the bridge is not a full solution, as you'd still need synchronized unpausing to prevent the issue (without it, the bridge could be unpaused after portals and still lose funds). It is more likely that we simply remove the ability to pause withdrawals later on, as pausing xsubmit achieves the same effect.*

OMNB-03	Delay In Calling `setup()` Could Result In Accounting Errors		
Asset	`NominaBridgeNative.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

Calling `setup()` after `initializeV2()` will cause incorrect accounting in the `NominaBridgeNative` contract. Resulting in the `l1Deposits` variable inaccurately representing the amount of layer 1 deposits.

This is only applicable once the contract is deployed. Which means, it is unlikely to occur as long as the owner ensures correct ordering of transactions during deployment.

The `setup()` function is intended to allow the owner of the contract to update specified variables in the contract. As shown here:

NominaBridgeNative.sol::setup()

```
function setup(uint64 l1ChainId_, address portal_, address l1Bridge_, uint256 l1Deposits_) external onlyOwner {
        l1ChainId = l1ChainId_;
        portal = INominaPortal(portal_);
        l1Bridge = l1Bridge_;
        l1Deposits = l1Deposits_;
        emit Setup(l1ChainId_, portal_, l1Bridge_, l1Deposits_);
}
```

During the initialisation of the contract, the `initializeV2()` function updates the `l1Deposits` variable to account for the exchange rate between OMNI and NOM.

NominaBridgeNative.sol::initializeV2()

```
function initializeV2() external reinitializer(2) {
        l1Deposits *= _CONVERSION_RATE;
}
```

Due to the lack of access control on the `initializeV2()` function, it could by anyone at anytime, which would be problematic if called before `setup()`. This would skew the accounting of the `l1Deposits` variable. Depending on the new value of `l1Deposits`, the contract would be holding an incorrect value of deposits which could lead to edge cases where users are unable to withdraw.

Recommendations

1. Ensure the `setup()` function is called prior to the `initializeV2()` function in the deploy script.
2. Set access control on the `initializeV2()` function to ensure it may only be called by the owner or a specified role. With this solution, it is still recommended to ensure the `setup()` function is called prior to the `initializeV2()` function.
3. Set a variable to indicate true when `setup()` has been called. In the `initializeV2()` function, verify your variable is indicating true.

Resolution

The development team has closed the issue with the following comment:

> *No changes implemented. I think there may have been a misunderstanding here, as there are no new contracts being deployed. The existing bridges are deployed under TransparentUpgradeableProxy contracts from OpenZeppelin, and will be getting upgraded during our rollout. The values configured by setup have been configured already in the proxy's storage, so there is no risk of initializeV2 causing incorrect accounting. setup is purely an administrative function and is not expected to be needed on public networks moving forward. Additionally, because OZ's transparent proxy style involves upgrades being performed by a ProxyAdmin contract that can also forward calldata to execute atomically with the upgrade (such as calling initializeV2), access controls on the initializer functions are not necessary. This is the pattern we follow for all contract upgrades, and is why you will not see access controls on our initializers anywhere in our codebase, except for when they are not intended to be called atomically with the upgrade.*

OMNB-04	Non-Standard Naming Convention And Unnecessary Caching Of Immutable Variable
Asset	`NominaBridgeL1.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The contract contains two code quality issues related to the `omni` and `nomina` immutable variables that deviate from standard Solidity conventions and best practices.

First, the immutable variables uses lower case naming instead of the standard convention of capitalised constants and immutables:

NominaBridgeL1.sol::omni

```
IERC20 public immutable omni;
```

NominaBridgeL1.sol::nomina

```
IERC20 public immutable nomina;
```

Second, the `initializeV2()` function unnecessarily caches the immutable variable to a stack variable:

NominaBridgeL1.sol::initializeV2()

```
function initializeV2() external reinitializer(2) {
        address _nomina = address(nomina); //@audit unnecessary cache
        omni.approve(_nomina, type(uint256).max);
        INomina(_nomina).convert(address(this), omni.balanceOf(address(this)));
 }
```

Immutable variables are stored directly in the contracts bytecode, meaning they do not incur gas to access them. Typically, caching variables is used to save on gas costs.

Recommendations

Use capitalised naming instead of lower case on immutable variables and remove the unnecessary caching of the `nomina` variable.

```
IERC20 public immutable OMNI;
```

```
IERC20 public immutable NOMINA;
```

```
function initializeV2() external reinitializer(2) {
        OMNI.approve(address(NOMINA), type(uint256).max);
        INomina(address(NOMINA)).convert(address(this), OMNI.balanceOf(address(this)));
 }
```

Note that modifying these variables will impact the getter functions due to the `public` tags.

Resolution

This issue has been addressed in the commit a849c79 by applying standard naming to these immutable variables and have removed unnecessary caching of said variables.

OMNB-05	Missing Validation In Constructor Could Render `withdraw()` Inoperable
Asset	`NominaBridgeL1.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

There is no validation of the `omni_` address in the constructor, which during `initializeV2()`, could lead to approving a non-existent token and rendering the contract inoperable.

The contract's constructor accepts the `omni_` address as a deployment parameter. Since this variable is declared as immutable, it becomes permanently fixed in the contract's bytecode and cannot be modified after deployment.

NominaBridgeL1.sol::constructor()

```
constructor(address omni_, address nomina_) {
        omni = IERC20(omni_);
        nomina = IERC20(nomina_);
        _disableInitializers();
}
```

Currently, there is no validation that the sent in `omni_` parameter matches the live OMNI address on mainnet.

The issue arises in the `initializeV2()` function which approves the `nomina` IERC20 to spend the contracts `omni`. However, if the contracts `omni` address is set incorrectly, it could result in a random token being approved and ultimately result in no tokens being converted during initialisation. As a result, this would render the `withdraw()` function un-useable due to insufficient funds as the contract has zero `nomina` to transfer.

Recommendations

It is recommended to ensure the `omni_` parameter matches the current OMNI address that is live on mainnet by performing a check in the constructor before setting the variable.

```
constructor(address omni_, address nomina_) {
        require(omni_ == 0x5329218a052986EBe688b07E58de61Afd55F9DC9, "Invalid OMNI address");
        omni = IERC20(omni_);
        nomina = IERC20(nomina_);
        _disableInitializers();
}
```

A cleaner and easier solution would be to add this check to the deployment script to ensure the address being sent in to the contract is correct.

Resolution

The development team has closed the issue with the following comment:

No changes implemented. As explained on our audit prep call, our upgrade script performs validation on these variables in order to prevent invalid constructor arguments from bricking the contract. As we atomically call initializeV2 with the upgrade, even if improper arguments are provided, the upgrade will fail. Additionally, as aforementioned, we are deploying to upgrade TransparentUpgradeableProxy contracts, not deploying fresh standalone contracts, so this risk is not apparent. We prefer to enforce storage checks in this manner, rather than hardcoding addresses, which would dramatically complicate devops for our multi-network operations (staging, testnet, mainnet).

OMNB-06	Miscellaneous General Comments
Asset	All contracts
Status	**Resolved:** See Resolution
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **Incorrect Documentation For Variable `omni`**

   Related Asset(s): NominaBridgeL1.sol

   The comment incorrectly refers to the NOM token, while the variable is named omni.

   NominaBridgeL1.sol::omni
   ```
   /**
    * @notice The NOM token contract.
    */
   IERC20 public immutable omni;
   ```

   It is recommended to update the comment to reference omni to keep the documentation consistent.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The development team's responses to the raised issues above are as follows:

We have corrected the comment for the OMNI immutable variable.

Appendix A Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
			Likelihood	

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: https://blog.sigmaprime.io/solidity-security.html. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: http://www.dasp.co/. [Accessed 2018].

σ'