Polygon

LXLY Upgradeable Wrapped Tokens

Security Assessment Report

Version: 2.0

May, 2025

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Polygon smart contract components in scope. The review focused solely on the security aspects of the Solidity implementation of the contracts, 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 Polygon 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 Polygon components in scope.

Overview

This review focused on changes that facilitate deploying an upgradeable proxy contact for wrapped tokens from the LXLY bridge.

This change allows the implementation contracts of wrapped tokens to be modified, and so allows for a more asset centric approach in future versions.

Security Assessment Summary

Scope

The review was conducted on the files hosted on the Agglayer Contracts repository.

The scope of this time-boxed review was strictly limited to the changes made to Solidity files and `updateVanillaGenesis.ts` in the following diff v10.0.0-rc.6...v10.1.0-rc.1.

The fixes of the identified issues were assessed at tag v10.1.0-rc.4.

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

Approach

The security assessment covered components written in Solidity.

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: 1 issue.
- Informational: 5 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Polygon 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 contract(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
UWT-01	Sensitive Storage Values From The Genesis Input Are Not Checked	Medium	Resolved
UWT-02	No Check For Mintable WETH Tokens	Informational	Resolved
UWT-03	Repeated Code	Informational	Resolved
UWT-04	Zero Address Checks	Informational	Resolved
UWT-05	Public Variable With Getter Function	Informational	Resolved
UWT-06	Miscellaneous General Comments	Informational	Resolved

true

UWT-01	Sensitive Storage Values From The Genesis Input Are Not Checked		
Asset	`updateVanillaGenesis.ts`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

There are multiple values in the final genesis that are not modified by the script but are instead determined by the initial genesis input:

- The `owner` of the `ProxyAdmin`, `PolygonZkEVMDeployer` contracts
- `polygonRollupManager`, `pendingProxiedTokensManager` and `proxiedTokensManager` in `bridgeProxy.storage`
- `pendingGlobalExitRootUpdater` and `pendingGlobalExitRootRemover` in `gerProxy.storage`
- The settings of the timelock contract
- Whether the tester account from `1_createGenesis.ts` has zero ether balance
- Whether the bytecode of the global exit root manager implementation contract contains the correct bridge address.

These values are sensitive from a security perspective, but are determined not by the operations of this script, but rather the input genesis that this script receives.

The development team stated that the input genesis would either be generated internally or reviewed internally. It is therefore likely to have the correct data.

It is conceivable, however, in a worst case scenario, that an internal actor, at some point in the genesis production process, might alter the genesis to modify these values. Whilst this is unlikely, the impact is so high that the values should be checked before deployment.

Recommendations

If these values are being checked in other scripts in the deployment process, it may be that no modifications to this script are required.

Otherwise, it is recommended to add checks to validate these storage values in this script.

Resolution

The recommended checks were added at commit 3388ea1.

UWT-02	No Check For Mintable WETH Tokens
Asset	`updateVanillaGenesis.ts`
Status	**Resolved:** See Resolution
Rating	Informational

Description

If the sovereign WETH token is mintable, the script checks that the relevant value in the mapping `wrappedAddressIsNotMintable` is set to `true`, but it does not check the inverse case.

Whilst it is very unlikely that the value of the storage slot would be anything but zero, given the context this script runs in, it is a simple test to add, which would make the checks more comprehensive.

Recommendations

Consider adding an `else` block to this check and confirming that the inverse case is also covered.

Resolution

The relevant checks was added at commit 9dfc9d5.

UWT-03	Repeated Code
Asset	`updateVanillaGenesis.ts`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The same conditional check appears on both line [**372**] and line [**503**], despite no mutations or relevant changes to the checked data occurring between these lines.

Repeating the same check without state changes introduces unnecessary code complexity and may slightly increase gas usage.

Recommendations

Remove the duplicated check and confirm whether the second occurrence was meant to validate a different variable or condition.

Resolution

The second check was removed at commit 3ec25fa.

UWT-04	Zero Address Checks
Asset	`BridgeL2SovereignChain.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The `initialize()` function does not perform zero address checks for its input arguments.

This omission could lead to critical misconfiguration of the contract during its initialisation phase and result in the assignment of critical roles, ownership, or configuration settings to the zero address.

Recommendations

Add explicit checks within the `initialize()` function to ensure that all critical address parameters are not set to the zero address.

Resolution

The recommended checks were added at commit 3ec25fa.

UWT-05	Public Variable With Getter Function
Asset	`PolygonZkEVMBridgeV2.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The variable `wrappedTokenBridgeImplementation` is `public`, which creates a getter function for it with the same name.

```
address public immutable wrappedTokenBridgeImplementation;
```

However, it also has a getter function `getWrappedTokenBridgeImplementation()`.

```
function getWrappedTokenBridgeImplementation()
    external
    view
    returns (address)
{
    return wrappedTokenBridgeImplementation;
}
```

This has little impact beyond a very slightly increase in the gas costs of some calls to the contract. Nevertheless, it is redundant.

Recommendations

Consider modifying the variable `wrappedTokenBridgeImplementation` to be `internal`.

Resolution

The variable `wrappedTokenBridgeImplementation` was made `internal` at commit 3ec25fa.

UWT-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 Information In Comments**
   Related Asset(s): updateVanillaGenesis.ts
   - The comment on line [**160**] is wrong: the address is not checked in the code it covers. It does not need to be checked, as the value has just been assigned, but the comment is inaccurate nonetheless.
   - The comment on line [**194**] has the wrong contract name. The code is actually checking the wrapped token implementation contract's bytecode.
   - The comment on line [**472**] has the wrong contract name. The code is actually updating the `gerProxy` contract's storage.
   - The comment on line [**496**] has the wrong role name. The code is checking the global exit root remover, not updater.

   Correct the comments.
2. **Typos**
   Related Asset(s): PolygonZkEVMBridgeV2.sol BytecodeStorer.sol
   - On line [**1030**] of `PolygonZkEVMBridgeV2.sol`, *"ProxiedTokensManagerR"* should be *"ProxiedTokensManager"*.
   - On line [**8**] of `BytecodeStorer.sol`, *"warped"* should be *"wrapped"*.

   Correct the typos.
3. **Unclear Comment**
   Related Asset(s): PolygonZkEVMBridgeV2.sol
   The comment on line [**69**] is confusing.

```
// Invalid address is set as proxy admin to not allow the proxy to be upgraded on mainnet
address public constant INVALID_WTOKEN_PROXY_ADMIN =
    0xFFfFFfFfFFfFFfFfFFFFFFFFFFFFFFFFFFFFFFFF;
```

   The address is not invalid, it is (presumed) uncontrolled, and the reason this address is required is unclear. Consider a more complete explanation. One version might be:

   > *"This is used to return an uncontrolled default address other than `address(0)` in the function `getProxiedTokensManager()`. Otherwise, the constructor of `TokenWrappedTransparentProxy` will revert when it calls that function, as it will not allow an attempt to change admin to the zero address."*

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.

1. **Incorrect Information In Comments**
   The comments were corrected at commit 3ec25fa.
2. **Typos**
   The typos were corrected at commit 3ec25fa.
3. **Unclear Comment**
   A more complete explanatory comment was added at commit 3ec25fa.

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.

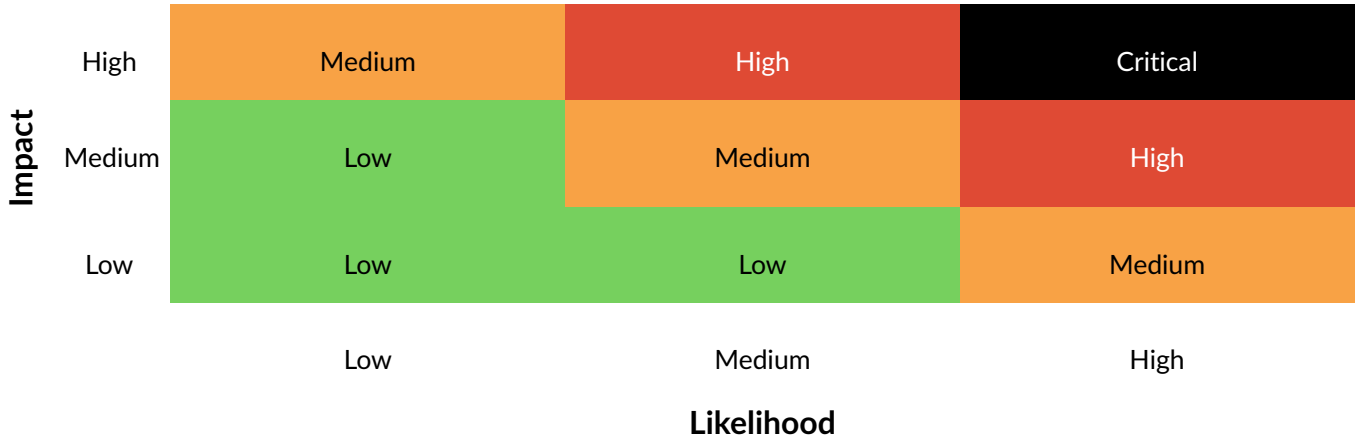

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

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].