Protocol Guild

Agora Token

Security Assessment Report

Version: 2.1

September, 2025

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Protocol Guild 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 Protocol Guild 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*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

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

Overview

This engagement focused on the `PGUILD` token contract. `PGUILD` is an upgradeable, non-transferable NFT that integrates with the `IVotes` interface such that it can be used with an Agora DAO for onchain governance.

Security Assessment Summary

Scope

The review was conducted on the files hosted on the voteagora/pg-token repository.

The scope of this time-boxed review was strictly limited to files at commit cc4dc03, which represents the approximate state of the codebase at the time of review, as the repository name was subsequently changed by the client.

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 4 issues during this assessment. Categorised by their severity:

- Informational: 4 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Protocol Guild 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
AGTK-01	`BURNER_ROLE` Can Burn A Non-Existing `tokenId`	Informational	**Closed**
AGTK-02	Missing Initialisation For `VotesUpgradeable`	Informational	**Resolved**
AGTK-03	Usage of Solidity Version Newer Than `0.8.20`	Informational	**Closed**
AGTK-04	Miscellaneous General Comments	Informational	**Resolved**

AGTK-01	`BURNER_ROLE` Can Burn A Non-Existing `tokenId`
Asset	`GovernanceToken.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

If an address with the `BURNER_ROLE` calls `burn()` there is no check to ensure that `tokenId` exists. As such, it is possible to successfully call `burn()` for a non-existing token. This has little direct impact, as burning a non-existing token does not meaningfully change any state. However, any integrating contracts or offchain components may expect an actual token to be burned if `burn()` is called successfully. As a result, issues may occur in these integrating components.

GovernanceToken.sol

```
function burn(uint256 tokenId) public {
    require(
        hasRole(BURNER_ROLE, _msgSender()) || ownerOf(tokenId) == _msgSender(),
        "Caller must be owner or have BURNER_ROLE"
    );
    _update(address(0), tokenId); //@audit tokenId is unconstrained if called by `BURNER_ROLE`
}
```

Recommendations

Consider implementing a check that reverts the call if `tokenId` does not exist.

Resolution

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

> *We will not be addressing this because tracking the current `tokenId` just adds storage in a scenario where there are no negative consequences to burning a non-existent token.*

AGTK-02	Missing Initialisation For `VotesUpgradeable`
Asset	`GovernanceToken.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The initialiser for `GovernanceToken` does not call the initialiser for `VotesUpgradeable`. Currently, `__Votes_init()` is a no-op and, as such, this issue has no impact. However, if `VotesUpgradeable` is ever modified such that it must be initialised, issues could occur.

GovernanceToken.sol

```
function initialize(address defaultAdmin, address _timelock, string memory name_, string memory symbol_)
public
initializer
{
  __AccessControl_init();
  __UUPSUpgradeable_init();
  __EIP712_init("Protocol Guild Membership", "1");
  //@audit `__Votes_init()` is not called

  // ...
}
```

Recommendations

Consider calling `__Votes_init()` in the initialiser as a best practice.

Resolution

This issue was fixed by initialising `VotesUpgradeable`.

AGTK-03	Usage of Solidity Version Newer Than `0.8.20`
Asset	`GovernanceToken.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The smart contracts are meant to be compiled with Solidity 0.8.26 which uses opcodes like `PUSH0` and `MCOPY` by default. However, not all rollups support these newer opcodes (e.g. Linea). As such, if this contract is meant to be deployed on a rollup ensure that it has support for these opcodes.

Recommendations

Ensure every target chain supports the above opcodes. If not, a lower EVM target version can be set in the Solidity compiler.

Resolution

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

> *We will be keeping the current Solidity version as it aligns with our governor and we know we will be deploying on Sepolia and Mainnet Ethereum, thus those OP codes are supported by default.*

AGTK-04	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. **Unused Error Definition**
   Related Asset(s): GovernanceToken.sol
   The `NotTransferable()` error is defined but not used.
   Consider removing it to enhance code clarity.
2. **Unused Internal Functions**
   Related Asset(s): GovernanceToken.sol
   The `_burn()` and `_increaseBalance()` functions are unused and can be removed to increase code clarity. Additionally, the `_safeMint()` function could be removed by having `mint()` call `_mint()` directly.
   If these functions are not meant to be used by an inheriting contract, consider removing them.
3. **Unused State Variables**
   Related Asset(s): GovernanceToken.sol
   The `admin` and `timelock` state variables are both unused.
   Consider removing these variables for code clarity.

Recommendations

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

Resolution

The issues were resolved by removing the unused storage and functions.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are given along with this document. The `forge` framework was used to perform these tests and the output is given below.

```
Ran 9 tests for test/tests-local/GovernanceToken.t.sol:GovernanceTokenTest
[PASS] test_burn() (gas: 126056)
[PASS] test_burn_multiple() (gas: 190598)
[PASS] test_burn_revertsWhenCallerNotOwnerOrBurner() (gas: 150295)
[PASS] test_burn_revertsWhenTokenDoesNotExist() (gas: 22214)
[PASS] test_initialized() (gas: 29589)
[PASS] test_initialized_reinitialize() (gas: 18746)
[PASS] test_mint() (gas: 222088)
[PASS] test_mint_revertsWhenNotMinter() (gas: 22160)
[PASS] test_mint_revertsWhenReceiverIsZero() (gas: 23351)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 12.53ms (2.91ms CPU time)
```

Appendix B 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.

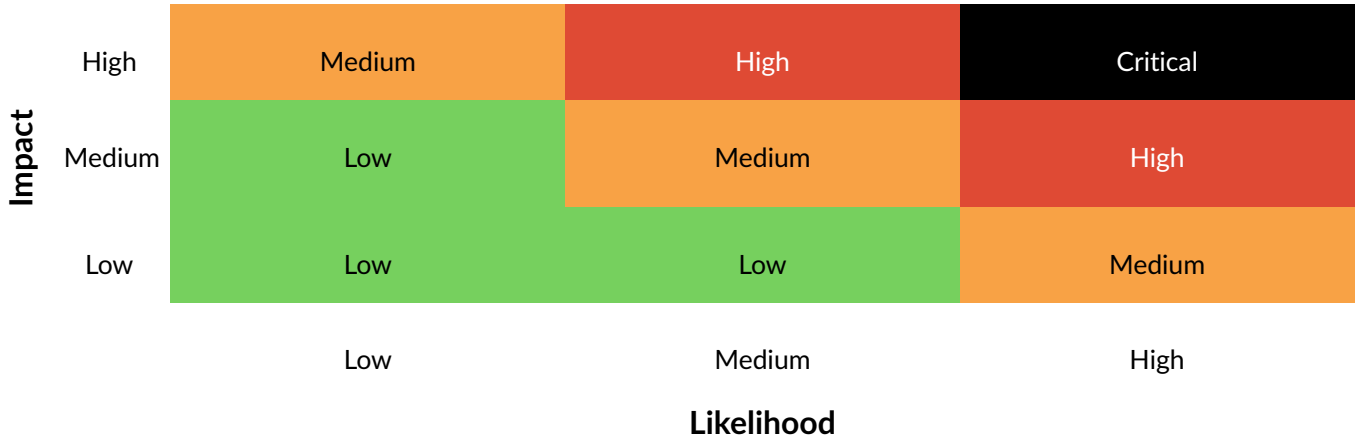

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

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].