SALUS SECURITY

SEP 2025

CODE SECURITY ASSESSMENT

HYPERBOT

Overview

Project Summary

- Name: Hyperbot(BOT)
- Platform: EVM-compatible chains
- Language: Solidity
- Address:
    - 0x59537849f2a119ec698c7Aa6C6DaAdc40C398A25
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Hyperbot(BOT)
Version	v2
Type	Solidity
Dates	Sep 02 2025
Logs	Sep 01 2025; Sep 02 2025

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	2
Total informational issues	7
Total	9

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	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 pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction **4**

1.1 About SALUS 4

1.2 Audit Breakdown 4

1.3 Disclaimer 4

Findings **5**

2.1 Summary of Findings 5

2.2 Notable Findings 6

1. Missing State Change Validation 6

2. Centralization risk 7

2.3 Informational Findings 8

3. Incomplete Docstrings 8

4. Missing two-step transfer ownership pattern 9

5. Lack of Security Contact Information for Responsible Disclosure 10

6. Underscore prefix for public variables 11

7. Lack of Indexed Event Parameters 12

8. Non-explicit Imports Reduce Code Readability 13

9. Custom Errors in require Statements 14

Appendix **15**

Appendix 1 - Files in Scope 15

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Missing State Change Validation	Low	Data Validation	Acknowledged
2	Centralization risk	Low	Centralization	Acknowledged
3	Incomplete Docstrings	Informational	Code Quality	Acknowledged
4	Missing two-step transfer ownership pattern	Informational	Business logic	Acknowledged
5	Lack of Security Contact Information for Responsible Disclosure	Informational	Configuration	Acknowledged
6	Underscore prefix for public variables	Informational	Code Quality	Acknowledged
7	Lack of Indexed Event Parameters	Informational	Code Quality	Acknowledged
8	Non-explicit Imports Reduce Code Readability	Informational	Code Quality	Acknowledged
9	Custom Errors in require Statements	Informational	Gas Optimization	Acknowledged

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Missing State Change Validation

Severity: Low	Category: Data Validation
Target: - SampleToken.sol	

Description

When calling the `setTransferController` function to update the `_transferController` variable, there is no check to see if the new value differs from the existing one. This can result in unnecessary state writes and event emissions, increasing gas costs and potentially impacting contract performance.

SampleToken.sol:L65 - L69

```
function setTransferController(address newValue) external onlyOwner {
    address oldValue = _transferController;
    _transferController = newValue;
    emit ChangeTransferController(oldValue, newValue);
}
```

SampleToken.sol:L58 - L62

```
if (_transferMode != TransferMode.NORMAL) {
    uint256 oldValue = _transferMode;
    _transferMode = newValue;
    emit ChangeTransferMode(oldValue, newValue);
}
```

Recommendation

It is recommended to verify whether the new value differs from the existing value before updating state variables. Perform the update only when the value actually changes to save gas and reduce redundant event emissions.

Status

This issue has been acknowledged by the team.

2. Centralization risk

Severity: Low	Category: Centralization
Target: - SampleToken.sol	

Description

The `SampleToken` contract has privileged accounts. When the contract is deployed, `totalSupply` will be minted to the owner, and the owner has the right to update the `transferController` variable. The `transferController`has the right to change the `transferMode`.

If the owner's private key is compromised, an attacker can freely use all tokens under that address and arbitrarily modify the `transferController` and `transferMode`.

If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the other users.

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

3. Incomplete Docstrings

Severity: Informational	Category: Code Quality
Target: - SampleToken.sol	

Description

The `SampleToken` contract lacks NatSpec (Ethereum Natural Specification) comments for its functions, parameters, return values, and events. NatSpec comments are widely adopted in the Solidity ecosystem to improve code readability, facilitate static analysis, and provide structured documentation for developers, integrators, and automated tools.

The absence of NatSpec documentation may lead to:

- Reduced code clarity and maintainability.
- Higher risk of misusing functions or misunderstanding their intended behavior.

Recommendation

It is recommended to provide complete documentation for all public functions and events, including details on their parameters and return values. Following the Ethereum Natural Specification Format (NatSpec) is highly encouraged to ensure consistency, readability, and usability. Comprehensive documentation improves long-term maintainability and facilitates both auditing and integration processes.

Status

This issue has been acknowledged by the team.

4. Missing two-step transfer ownership pattern	
Severity: Informational	Category: Business logic
Target: - SampleToken.sol	

Description

The `SampleToken` contract inherits from the `Ownable` contract. This contract does not implement a two-step process for transferring ownership. Thus, ownership of the contract can easily be lost when making a mistake in transferring ownership.

Recommendation

Consider using the Ownable2Step contract from OpenZeppelin instead.

Status

This issue has been acknowledged by the team.

5. Lack of Security Contact Information for Responsible Disclosure

Severity: Informational	Category: Configuration
Target: - SampleToken.sol	

Description

The contract `SampleToken` does not specify a security contact point. Including a designated security contact (such as an email address or ENS name) in the contract's NatSpec header facilitates responsible vulnerability disclosure. This makes it easier for external researchers to quickly reach the appropriate team in the event a vulnerability is identified, helping minimize the time window between discovery and mitigation. The Ethereum community has begun standardizing this practice using the `@custom:security-contact` tag, adopted by tools such as OpenZeppelin Wizard and ethereum-lists.

Recommendation

Consider adding a NatSpec comment at the top of the contract with a `@custom:security-contact` field pointing to the preferred disclosure channel.

Status

This issue has been acknowledged by the team.

6. Underscore prefix for public variables

Severity: Informational	Category: Code Quality
Target: - SampleToken.sol	

Description

When declaring variables in a smart contract, it is common practice to add a leading underscore for non-external variables, while public variables are written without it. This helps developers efficiently distinguish variable visibility and improves code readability. In some cases, `constructor` parameters may have the same name as internal variables. Using a leading underscore can avoid such naming conflicts.

Recommendation

Consider removing the leading underscore prefix from public variables, and apply it only to internal and private variables, in line with the recommendations provided by soliditylang's guidelines.

Status

This issue has been acknowledged by the team.

7. Lack of Indexed Event Parameters

Severity: Informational	Category: Code Quality
Target: - SampleToken.sol	

Description

Within the `SampleToken` contract, multiple events are missing indexed parameters. This prevents off-chain services from efficiently querying or filtering logs based on specific values . As a result, applications, analytics tools, and monitoring systems must scan all emitted logs instead of filtering directly, which increases computational overhead, reduces efficiency, and makes it harder to track user actions or state changes. In large-scale systems, this limitation can cause performance bottlenecks and hinder real-time monitoring or analytics capabilities.

SampleToken.sol:L23 - L24

```
event ChangeTransferController(address oldValue, address newValue);
event ChangeTransferMode(uint256 oldValue, uint256 newValue);
```

Recommendation

To improve the ability of off-chain services to search and filter for specific events, consider indexing event parameters.

Status

This issue has been acknowledged by the team.

8. Non-explicit Imports Reduce Code Readability

Severity: Informational	Category: Code Quality
Target: - SampleToken.sol	

Description

The contract `SampleToken.sol` uses wildcard or global-style import statements such as `import "@openzeppelin/contracts@4.9.6/access/Ownable.sol";`, which introduce all symbols from the imported file into the current compilation unit. While functional, this approach can reduce code readability and make it unclear which specific contracts, interfaces, or types are actually being used in the file. Explicit import { A, B } from "<path>"; declarations are generally preferred, as they make dependencies explicit and reduce the potential for namespace conflicts or unintentional symbol usage.

Recommendation

Consider refactoring a complete import statement to use named import syntax.

Status

This issue has been acknowledged by the team.

9. Custom Errors in require Statements

Severity: Informational	Category: Gas Optimization

Target:
- SampleToken.sol

Description

SampleToken.sol:L36 - L63

```
function _beforeTokenTransfer(
    address from,
    address to,
    uint256 amount
) internal virtual override {
    super._beforeTokenTransfer(from, to, amount);
    if (_transferMode == TransferMode.RESTRICTED) {
        revert("Transfer is restricted");
    }
    if (_transferMode == TransferMode.CONTROLLED) {
        require(from == _transferController || to == _transferController, "Invalid
transfer");
    }
}

function setTransferMode(uint256 newValue) external {
    require(msg.sender == _transferController, "Caller is not the transfer controller");
    require(newValue <= TransferMode.MAX_VALUE, "Invalid mode");

    if (_transferMode != TransferMode.NORMAL) {
        uint256 oldValue = _transferMode;
        _transferMode = newValue;
        emit ChangeTransferMode(oldValue, newValue);
    }
}
```

The `SampleToken.sol` contract uses <if (condition) revert("error message") statements and require(condition, "error message") statements>. Since Solidity version 0.8.26, require statements support custom errors, which are more gas-efficient and improve code clarity. Initially, this feature was only available through the IR pipeline, but starting from Solidity 0.8.27, it is also supported in the legacy pipeline.

Recommendation

Consider replacing all if-revert statements and require(condition, "error message") statements with require(condition, CustomError()) to improve readability and reduce gas consumption.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files at address 0x59537849f2a119ec698c7Aa6C6DaAdc40C398A25:

File	SHA-1 hash
SampleToken.sol	cded6e076ed7ec9b78d31d00571e95b6dbfd0e87