SALUS SECURITY

SEP 2024

CODE SECURITY ASSESSMENT

CHASM

Overview

Project Summary

- Name: Chasm
- Platform: EVM-compatible chains
- Language: Solidity
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Chasm
Version	v2
Type	Solidity
Dates	Sep 18 2024
Logs	Sep 12 2024; Sep 18 2024

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	1
Total Low-Severity issues	1
Total informational issues	3
Total	6

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

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	Reverts in back() and backWithReferrer() functions	High	Business Logic	Resolved
2	Centralization risk	Medium	Centralization	Acknowledged
3	Missing events for functions that change critical state	Low	Logging	Acknowledged
4	Redundant code	Informational	Redundancy	Acknowledged
5	Hardcoded Gas Limit	Informational	Code Quality	Resolved
6	Use of floating pragma	Informational	Configuration	Resolved

2.2 Notable Findings

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

1. Reverts in back() and backWithReferrer() functions	
Severity: High	Category: Business Logic
Target: - Backer.sol	

Description

If the gas fees prepaid to `payNativeGasForContractCall()` or `payNativeGasForContractCallWithToken()` exceeds the actual amount needed for relaying a message to the destination contract, the Gas Service will automatically calculate the excess gas amount and refund it to the payer's wallet address by calling `refund()` on the `AxelarGasService` contract. The refunded amount consists of the total amount paid, minus the network base fee, the actual gas used, and the gas costs estimated for transferring the refund. Refunds are made on the source chain.

Within the `back()` and `backWithReferrer()` functions, `_sendCrossChainMessage()` initiates a gas prepayment using `gasService.payNativeGasForContractCall{value: estimatedFee}`. This function designates `address(this)` as the refund address. However, `Backer.sol` is missing an implementation for `receive()`, which is necessary for handling these payments. As a result, if the refund amount is greater than 0, the calls to `back()` and `backWithReferrer()` will revert.

Recommendation

Consider implementing the `receive()` function to handle incoming payments appropriately.

Status

The team has resolved this issue.

2. Centralization risk

Severity: Medium	Category: Centralization
Target: - All	

Description

All contracts have an `owner` role or utilize the `Access Control` module by OpenZeppelin. The `owner` has special permissions such as `setMerkleRoot()`, `updateNftContract()`, etc. Additionally, the `ADMIN_ROLE` can grant any permissions to any address, enabling further privileged operations.

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.

3. Missing events for functions that change critical state

Severity: Low	Category: Logging
Target: - All	

Description

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes that allow users to evaluate them. Missing events do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in protocol users.

In all contracts, certain key variables are modified without triggering the corresponding events, such as the `updateApprovedSource()`, `updateNftContract()`, `pause()`, `unpause()`, `setUnlockDate()`, and `toggleMint()` operations.

Recommendation

It is recommended to emit events for critical state changes.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

4. Redundant code

Severity: Informational	Category: Redundancy
Target: - Backer.sol	

Description

`Backer` contract inherits `IAxelarGateway` contract with default implementation. But actually there is no use anywhere and can be considered as redundant.

Recommendation

Consider removing the redundant code.

Status

This issue has been acknowledged by the team.

5. Hardcoded Gas Limit

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

Description

The `_sendCrossChainMessage()` function currently utilizes a hardcoded gas limit as 200000. Given that the `Backer` contract lacks an upgrade mechanism, it's advisable to manage the gas limit through a separate variable controlled by the contract owner. This approach follows good practice.

Recommendation

Consider implementing a separate variable for managing the gas limit.

Status

The team has resolved this issue.

6. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - All	

Description

```
pragma solidity ^0.8.21;
```

All contracts use a floating compiler version ^0.8.21.

Using a floating pragma ^0.8.21 statement is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

Status

The team has resolved this issue.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit :

File	SHA-1 hash
Backer.sol	9c241afc757a84b4afc3996d2f1d549115b38ec0
BackerFactory.sol	ed36cd8c5bdfd6f732d6cda3338fab6b4d843c4f
BackerSync.sol	7b292aee646b99ef19820331c72f0e331b2f19b6
ChasmScout.sol	3d4d1cba90e35e05a2679b6c041bdda795b4605e
MerkleMinter.sol	63b5959cc03abdeb387e1dccacb2f31d31e9b6c1