Stargate

Security Assessment

April 26th, 2024 — Prepared by OtterSec

Jessica Clendinen jc0f0@osec.io

Nicholas R. Putra nicholas@osec.io

Robert Chen notdeghost@osec.io

Table of Contents

Appendices

01 — Executive Summary

Overview

Stargate Protocol engaged OtterSec to assess the `stargate-v2` program. This assessment was conducted between March 22nd and April 19th, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 10 findings throughout this audit engagement.

We identified several vulnerabilities, including utilizing incorrect bounds within `ride` for checking the allowed number of passengers on the `bus` (OS-STG-ADV-01) and a miscalculation in `redeemSend` that overlooks transfer fees, possibly resulting in an inaccurate credit balance (OS-STG-ADV-02). Additionally, we highlighted the bypass of the standard deposit mechanism due to direct token transfers to the pool (OS-STG-ADV-00).

We also recommended implementing a validation check for the `_baseFareMultiplierBps` parameter within `setBaseFareMultiplierBps` (OS-STG-SUG-00) and advised verifying that the status is not `ENTERED` before unsetting the `reentrancy` guard in the pause functionality (OS-STG-SUG-01). Furthermore, we highlighted the possibility of creating an invalid reward pool (OS-STG-SUG-02).

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/stargate-protocol/stargate-v2. This audit was performed against commit fecfeb2.

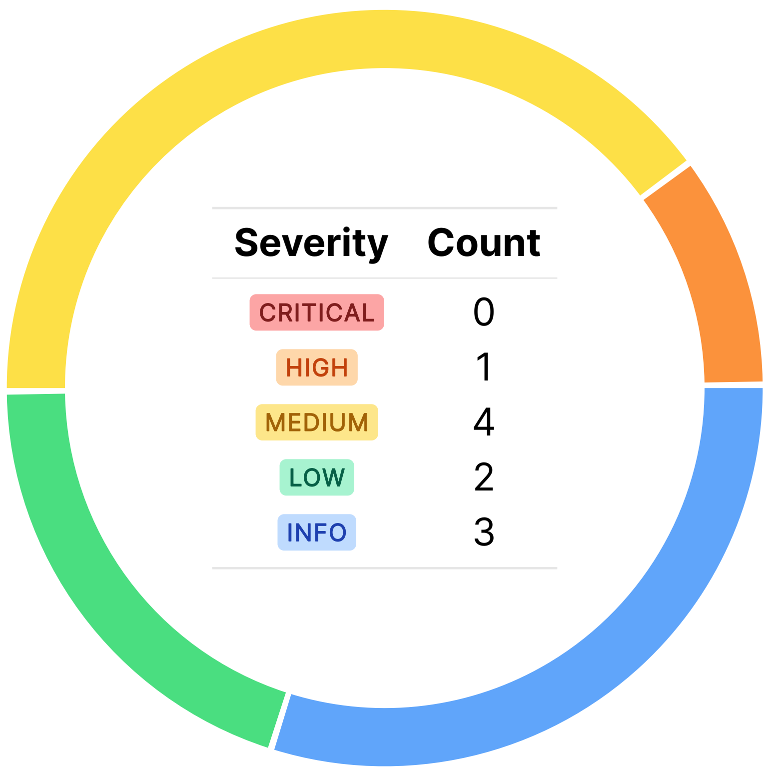
A brief description of the programs is as follows:

Name	Description
stargate-v2	Facilitates the cross-chain communication of tokens through the LayerZero ecosystem, and manages various tasks such as setting configurations, sending tokens, managing fees, and handling messaging.

03 — Findings

Overall, we reported 10 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



Severity	Count
CRITICAL	0
HIGH	1
MEDIUM	4
LOW	2
INFO	3

04 — Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-STG-ADV-00	HIGH	RESOLVED	The pool bypasses the standard deposit mechanism due to direct token transfers to the pool.
OS-STG-ADV-01	MEDIUM	RESOLVED	`ride` uses incorrect bounds while checking the number of passengers allowed on the `bus`.
OS-STG-ADV-02	MEDIUM	RESOLVED	The adjustment of the local credit amount in `redeemSend` contains a miscalculation that overlooks transfer fees, possibly resulting in an inaccurate credit balance.
OS-STG-ADV-03	MEDIUM	RESOLVED	`_lzReceiveBus` attempts to transfer leftover native tokens to the Layer Zero endpoint, which lacks a fallback function, reverting the transaction.
OS-STG-ADV-04	MEDIUM	RESOLVED	`checkTickets` allows a user to pass an invalid `numPassengers` parameter to `driveBus`, resulting in incorrect head ticket identification and overwriting the hash chain.
OS-STG-ADV-05	LOW	RESOLVED	`transferNative`, does not encapsulate the low-evel call inside an assembly block resulting in potential gas exhaustion error.
OS-STG-ADV-06	LOW	RESOLVED	The assembly block in `decodePassenger` fails to check if the memory slot is zeroed out before loading data, potentially resulting in incorrect values being read due to memory slot pollution.

Inflation Of Pool Balances HIGH

OS-STG-ADV-00

Description

In both `StargatePool` and `StargatePoolNative`, `_getPoolBalance` retrieves the pool balance by directly calling `balanceOf` on the associated `ERC20` token contract. However, this method fails to account for tokens sent directly to the contract address, thus bypassing the deposit mechanism. As a result, if users send tokens directly to the pool contract without following the correct deposit and minting process, the pool balance will become inflated.

This inflation results in inaccurate exchange rate calculations between liquidity provider tokens and underlying tokens, potentially disrupting subsequent fee calculations, which depend on accurate measurements of the pool's balance and the supply of liquidity provider tokens.

>_ stg-evm-v2/src/StargatePool.sol solidity

```solidity
function _getPoolBalance() internal view virtual returns (uint256 balance) {
    balance = IERC20(token).balanceOf(address(this));
}
```

Moreover, in `FeeLibV1::applyFeeView`, a check exists to prevent the pool balance from exceeding the total value locked (TVL) represented by liquidity provider tokens. If this check fails due to an inflated pool balance, the function will revert, disrupting the system's operation.

Remediation

Maintain an internal record of the pool balance within `StargatePool` and `StargatePoolNative`, distinct from the token's balance retrieved utilizing `balanceOf`. Moreover, any alterations to the pool balance or local credit should prompt adjustments to the `TVL` variable to guarantee precise fee computations.

Patch

Fixed in 13edc95.

Incorrect Bounds Check MEDIUM OS-STG-ADV-01

Description

Within `BusLib::ride`, a check exists to verify if `bus` is at full capacity before permitting a passenger to board. However, the condition employed for this assessment is `ticketId - _bus.headTicketId >= _busCapacity`. This condition examines whether the disparity between the current `ticketId` and the `headTicketId` of the `bus` surpasses or equals `bus`' capacity (`_busCapacity`).

>_ stg-evm-v2/src/libs/Bus.sol solidity

```solidity
function ride(
    [...]
) internal returns (uint56 ticketId, bytes memory passengerBytes, uint128 fare, uint256 refund)
    ↪ {
    [...]
    // check if the bus is full
    if (ticketId - _bus.headTicketId >= _busCapacity) revert Bus_BusFull();
    [...]
}
```

The vulnerability stems from the condition allowing the number of passengers to match `bus`' capacity. This implies that the final available `ticketId` may equate to `_busCapacity + _bus.headTicketId - 1`. In such a scenario, `ride` permits a passenger to board despite the `bus` being technically full. `bus` serves as a means of grouping messages together.

Each message's hash is stored in a linked chain, and if the hash is not inserted into the `hashChain` or if the position is overwritten in `hashChain`, then the message is lost, resulting in subsequent failures in the `checkTickets` logic.

>_ stg-evm-v2/src/libs/Bus.sol solidity

```solidity
function checkTickets(
    [...]
) internal view returns (ThisBus memory drivingBus) {
    [...]
    // check the hash of the last passenger
    uint56 lastTicketIdToDrive = startTicketId + numPassengers - 1;
    if (lastTicketIdToDrive >= _bus.tailTicketId || lastHash !=
        ↪ _bus.hashChain[lastTicketIdToDrive % _busCapacity])
    revert Bus_InvalidPassenger();
}
```


Remediation

Revise `ride` to verify that the maximum number of passengers is one less than the `bus` capacity:

>_ *stg-evm-v2/src/libs/Bus.sol* solidity

```
function ride(
    Bus storage _bus,
    uint56 _busCapacity,
    uint32 _dstEid,
    TransferPayloadDetails memory _passenger,
    uint16 _baseFareMultiplierBps,
    uint128 _extraFare
) internal returns (uint56 ticketId, bytes memory passengerBytes, uint128 fare, uint256 refund)
    ↪ {
    [...]
    // check if the bus is full
    if (ticketId - _bus.headTicketId >= _busCapacity - 1) revert Bus_BusFull();
    [...]
}
```

Patch

Fixed in 06d5975.

Pool State Calculation Inconsistency MEDIUM OS-STG-ADV-02

Description

`StargatePool::redeemSend` increases local credit based solely on rewards received during deposit, without accounting for credit decrease due to transfer fees. When users redeem liquidity provider tokens and initiate token transfers, transfer fees may apply, affecting the credit balance of the pool's paths. However, `redeemSend` overlooks these fees, potentially resulting in an artificially inflated credit balance and inaccurate calculations of available credit.

>_ stg-evm-v2/src/StargatePool.sol solidity

```solidity
function redeemSend(
    SendParam calldata _sendParam,
    MessagingFee calldata _fee,
    address _refundAddress
)
    external
    payable
    nonReentrantAndNotPaused
    returns (MessagingReceipt memory msgReceipt, OFTReceipt memory oftReceipt)
{
    [...]
    // charge fees and handle credit
    FeeParams memory params = _buildFeeParams(_sendParam.dstEid, amountInSD, true, true, true);
    (uint64 amountOutSD, uint64 reward, ) = _chargeFee(
        params,
        RideBusParams("", 0),
        _ld2sd(_sendParam.minAmountLD)
    );
    // due to the local credit was already increased when deposit, we don't need to do it again
    // only increase the local credit if the reward is not zero
    _handleCredit(_sendParam.dstEid, amountOutSD, reward);
    [...]
}
```

As a result, if a fee for sending a token exists in the path, the path's credit will not be sufficiently decreased. The pool will have more credit than it should, negatively impacting protocol health

Remediation

Decrease the credit if a fee exists for sending the token and no reward is applied.

Patch

Fixed in 710fa6b.

Inability To Accept Native Token Transfers MEDIUM OS-STG-ADV-03

Description

The vulnerability in `TokenMessaging::_lzReceiveBus` stems from the handling of native tokens. After processing the bus passengers, any remaining native tokens are transferred to `msg.sender`, which, in this context, is the Layer Zero endpoint. However, the Layer Zero endpoint contract does not implement a `receive` or `fallback` function marked as payable. As a result, the contract cannot accept the native token transfer, leading to a transaction failure.

Remediation

Transfer the remaining native tokens to another contract that has either a receive or payable fallback function, such as a treasurer contract

Patch

Fixed in 06d5975.

Missing Calldata Validation MEDIUM OS-STG-ADV-04

Description

`Bus::checkTickets` neglects to validate the `numPassengers` parameter to ensure it does not exceed the actual number of tickets. Consequently, `driveBus` is invoked with this invalid parameter, resulting in exceeding the recorded number of passengers/tickets in the `bus` state. `drive`, responsible for updating the `bus` state, depends on the `numPassengers` parameter provided in the payload to determine the ticket range for processing.

>_ stg-evm-v2/src/libs/Bus.sol solidity

```solidity
function checkTickets(
    [...]
) internal view returns (ThisBus memory drivingBus) {
    // check the hash of the last passenger
    uint56 lastTicketIdToDrive = startTicketId + numPassengers - 1;
    if (lastHash != _bus.hashChain[lastTicketIdToDrive % BUS_CAPACITY]) revert
        ↪ Bus_InvalidPassenger();
}
```

Since `drive` determines the head of the ticket list for the `bus` based on `numPassengers`, an invalid `numPassengers` value will result in incorrect positioning within `hashChain`. Suppose `numPassengers` exceeds the actual number of tickets. In that case, the function may set the head of the ticket list beyond the valid range, overwriting existing ticket data in `hashChain`. Consequently, if the head of the ticket list is incorrectly set due to an invalid `numPassengers`, subsequent calls to `driveBus` on the same `bus` may fail.

Remediation

Check if `lastTicketIdToDrive` is greater than `tailTicketId`.

Patch

Fixed in a6342bb.

Gas Exhaustion in TransferNative LOW OS-STG-ADV-05

Description

In the `transferNative` function within the `transfer` method, there is a vulnerability caused by the Solidity compiler's behavior of copying all `returndata` from low-level calls into memory. This issue arises because the `call`, used for low-level calls, automatically transfers the `returndata` to memory. When a call is made without encapsulating it inside an assembly block, Solidity performs this copying operation by default. If the called contract returns a large amount of `returndata`, this operation can consume a significant amount of gas, potentially exceeding the gas limit set for the transaction.

Remediation

Encapsulate the low-level call within an assembly block to prevent the automatic copying of `returndata` to memory.

Patch

Fixed in 06d5975.

Unsafe Memory Handling LOW OS-STG-ADV-06

Description

`BusCodec::decodePassenger` decodes a byte array into a `BusPassenger` structure. The vulnerability arises from how `decodePassenger` handles the decoding process. It uses `mloads` to load the passenger structure in an assembly block. The concern is with the potential for ”dirty bits” — unintended data in memory slots — which can lead to incorrect interpretations of values. Particularly, when converting a byte slice to a boolean (`nativeDrop`), the conversion presumes that the byte is clean (all other bits are zero). However, if the memory slot contains leftover data, these dirty bits can mistakenly cause the boolean to be set to true..

Remediation

Mask the bytes or remove the assembly block and use Solidity types instead (the use of assembly here does not actually save much gas).

Patch

Fixed in 06d5975.

05 — General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-STG-SUG-00	`setBaseFareMultiplierBps` allows the planner to set fare multipliers to arbitrary values not expressed in basis points, potentially resulting in incorrect fare calculations.
OS-STG-SUG-01	`setPause` may be vulnerable to a reentrancy attack.
OS-STG-SUG-02	Creating an invalid reward pool within `getOrCreatePoolId` is possible.

Invalid Fare Multipliers

OS-STG-SUG-00

Description

The issue arises from potential incorrect fare calculation resulting from improper configuration parameters set by the planner using `setBaseFareMultiplierBps`. Specifically, the planner may set a fare multiplier to any number that is not expressed in basis points (bps), resulting in unintended and inaccurate fare calculations.

>_ *stg-evm-v2/src/StargateBase.sol* solidity

```solidity
function setBaseFareMultiplierBps(uint32 _dstEid, uint16 _baseFareMultiplierBps) external
    ↪  onlyCaller(planner) {
    paths[_dstEid].baseBusFareMultiplierBps = _baseFareMultiplierBps;
}
```

Remediation

Implement a validation check within `setBaseFareMultiplierBps` to ensure that the `_baseFareMultiplierBps` parameter is a multiple of 10,000, confirming that it is expressed in basis points.

Reentrancy Guard Enhancement

OS-STG-SUG-01

Description

`setPause` changes the status of `Stargate` based on the `_paused` parameter. However, it does not check whether the current status is `ENTERED` before allowing the status to be updated. This implies that if the status is `ENTERED`, the `reentrancy` guard is unset without verification.

Remediation

Add a check to ensure that the status is not `ENTERED` before unsetting the `reentrancy` guard, achieved by adding the following line of code at the beginning of the function:

solidity
```solidity
if (status == ENTERED) revert Stargate_ReentrantCall();
```

Invalid Reward Pool

OS-STG-SUG-02

Description

There is no validation to verify that the `rewardToken` is registered before creating a pool in `getOrCreatePoolId`. Without validation, if a pool is created with a non-existent `rewardToken`, the pool may be created with an invalid `rewardToken` address.

Remediation

Ensure the `rewardToken` is registered before creating a pool in `getOrCreatePoolId`.

A — Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings.

CRITICAL	Vulnerabilities that immediately result in a loss of user funds with minimal preconditions. Examples: • Misconfigured authority or access control validation. • Improperly designed economic incentives leading to loss of funds.
HIGH	Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit. Examples: • Loss of funds requiring specific victim interactions. • Exploitation involving high capital requirement with respect to payout.
MEDIUM	Vulnerabilities that may result in denial of service scenarios or degraded usability. Examples: • Computational limit exhaustion through malicious input. • Forced exceptions in the normal user flow.
LOW	Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions.
INFO	Best practices to mitigate future security risks. These are classified as general findings. Examples: • Explicit assertion of critical internal invariants. • Improved input validation.

B — Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.