Audit Report

Router Solana Integration: Gateway, Asset Forwarder, Asset Bridge Contracts

DRAFT – DO NOT PUBLISH

v0.5

September 25, 2024

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by Decimal FZC to perform a security audit of Router Solana Integration including Gateway, Asset Forwarder, and Asset Bridge Contracts.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/router-protocol/asset-bridge-contracts
Commit	`acd641eeb6bd0c973fe1804f3d391bafb67838bd`
Scope	All the Solana programs in the `solana` directory are in scope.
Fixes verified at commit	`0db578cee9b360628c1a2029034e160dbbb6a1f0` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Repository	https://github.com/router-protocol/asset-forwarder-contracts
Commit	`ceac739573a8323d4f3988e80dc09b06ca166fb2`
Scope	All the Solana programs in the `solana` directory are in scope.
Fixes verified at commit	`48f3d1a2936a8e8b98b585b9e4e193d23605e19c` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Repository	https://github.com/router-protocol/asset-gateway-contracts
Commit	`d45fb083755226589ceb4d1a75677d3d468cbe49`
Scope	All the Solana programs in the `solana` directory are in scope.
Fixes verified at commit	`7830612f4bc890e6335f07d2ca408711c3ff18c8` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Router protocol is a layer one chain focusing on blockchain interoperability, enabling cross-chain communications.

The scope of the audit is restricted to the integration with Solana, including Gateway, Asset Forwarder, and Asset Bridge Contracts.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium	There are many outstanding `TODO` comments throughout the codebase, along with unimplemented functionalities. Large sections of code are duplicated across different programs.
Level of documentation	Medium	The client provided diagrams and generic documentation about the protocol
Test coverage	Low-Medium	Unit tests are not implemented for programs. Integration tests are defined but some programs and instructions are not covered.

Summary of Findings

No	Description	Severity	Status
1	Missing `TokenAccount` validation allows attackers to circumvent fees and fake transfers and deposits	Critical	Resolved
2	The `IReceive` instruction of the asset bridge always fails	Critical	Resolved
3	`USDC` tokens can be stolen from anyone who has given approval to the external bridge authority	Critical	Resolved
4	Unverified information in deposit instructions allows attackers to counterfeit `FundsDeposited` and `FundsDepositedWithMessage` events	Critical	Resolved
5	Missing prefixes in PDA seeds allow attackers to manipulate account data	Critical	Resolved
6	Users can arbitrarily define the fee required by the `IDepositInfo` instruction	Critical	Acknowledged
7	Lack of access control allows attackers to overwrite data or execute restricted actions	Critical	Resolved
8	Missing ownership validation for `PacketAccount`	Major	Resolved
9	Multiple emissions of the `DepositInfoUpdate` for the same `deposit_id` are permitted	Major	Acknowledged
10	The `ISend` instruction can be executed if the program is paused	Major	Resolved
11	Admin role can be removed	Major	Resolved
12	Possible protocol DoS due to accidental account closure	Major	Resolved
13	Missing validation in the `CreateValsetArgsAccounts` instruction	Minor	Acknowledged
14	Missing role segregation	Minor	Resolved
15	Permissionless account initialization	Minor	Resolved
16	Template-like implementations	Minor	Acknowledged
17	Invalid signatures are not skipped in the	Minor	Resolved

	`check_validator_signatures` function		
18	Lack of validation for protocol fees	Minor	Acknowledged
19	The `verify_sig` function is susceptible to signature malleability	Minor	Acknowledged
20	Usage of native token is discouraged	Minor	Acknowledged
21	Potential mismatch between validators and powers in `check_validator_signatures` function	Minor	Acknowledged
22	Lack of differentiation for different request types	Informational	Acknowledged
23	Generic roles can be set	Informational	Resolved
24	Lack of event emission during configuration updates	Informational	Resolved
25	TODO comments across the codebase	Informational	Resolved
26	Miscellaneous comments	Informational	Partially Resolved

Detailed Findings

1. Missing `TokenAccount` validation allows attackers to circumvent fees and fake transfers and deposits

Severity: Critical

In several instruction contexts, `TokenAccount`s are defined to handle token transfers between entities:

- `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/i_deposit.rs:10-37`
- `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/i_deposit_info.rs:17-24`
- `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/i_deposit_message.rs:16-29`
- `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/i_relay.rs:21-27`
- `asset-forwarder-contracts:solana/programs/external_bridge/src/instructions/i_deposit_usdc.rs:26`
- `asset-bridge-contracts:solana/programs/asset_bridge/src/instructions/stake.rs:18-40`
- `asset-bridge-contracts:solana/programs/asset_bridge/src/instructions/stake.rs:18-40`
- `asset-bridge-contracts:solana/programs/asset_bridge/src/instructions/unstake.rs:18-40`
- `asset-bridge-contracts:solana/programs/asset_bridge/src/instructions/i_transfer_token.rs:32-45`
- `asset-bridge-contracts:solana/programs/asset_bridge/src/instructions/i_transfer_token_with_instruction.rs:32-45`

However, no validation is performed to ensure that `mint`, `authority`, and `token_program` are correct and refer to the legitimate entity and token.

Consequently, any `TokenAccount` would be processed and handled by the instruction logic regardless of the owner and the tokens that are represented allowing attackers to circumvent fees and fake transfers and deposits.

A proof-of-concept showcasing an attacker faking a deposit is provided in Appendix A-1.

Recommendation

We recommend validating `TokenAccount`s provided in the instruction context.

Status: Resolved

2. The `IReceive` instruction of the asset bridge always fails

Severity: Critical

The `_handle_record` function defined in `asset-bridge-contracts:solana/programs/asset_bridge/src/instructions/i_receive.rs` is responsible for verifying if a specific `IReceive` record was already handled.

However, according to the comment in the function, the code assumes that successfully computing a PDA (Program Derived Address) indicates that an `IReceive` record has been handled. This assumption is flawed because the PDA can always be calculated, thus its existence does not confirm any processed record. This error likely stems from a misunderstanding of the `Pubkey::create_program_address` function, which calculates the PDA using a provided bump, unlike `Pubkey::find_program_address`, which finds the canonical bump.

Since the function uses two methods to calculate the PDA and then compares their results, it always fails and returns an `AssetBridgeError::WrongPdaGenerated` error because the seeds used in `Pubkey::create_program_address` are missing the `primary_seed` value.

Consequently, it would not be possible to execute the `IReceive` instruction of the asset bridge.

Recommendation

We recommend implementing the following modifications:

- After calculating the PDA, verify the corresponding account. If the account contains data or has a non-zero lamports balance, it can be considered as already handled. If not, the record should be deemed unhandled, and data should be added to the account after handling it.
- It is guaranteed that `Pubkey::create_program_address` and `Pubkey::find_program_address` functions will result in the same PDA given the same seeds. Therefore, performing the calculation twice and checking if the results match is redundant and can be safely omitted.

Status: Resolved

3. `USDC` tokens can be stolen from anyone who has given approval to the external bridge authority

Severity: Critical

The `IDepositUSDC` instruction in `asset-forwarder-contracts:solana/programs/external_bridge/src/instructions/i_deposit_usdc.rs` requires the external bridge authority (`to` account) to

have approval for transferring `USDC` from the depositor's token account (`from` account) to authorize the transfer to the authority's token account (`to_ata` account), see lines `91-100`.

However, this transfer process is not sufficiently restricted to the `depositor` account that signs the `IDepositUSDC` instruction.

Consequently, a malicious `depositor` could abuse anyone's `USDC` approval to the external bridge authority to deposit or steal their tokens by specifying the victim's token account as the source instead of using their own.

Recommendation

We recommend requiring the `depositor` account, which is also the signer, to be the owner of the `from` account or have the approval for transfers from the `from` account.

Status: Resolved

4. Unverified information in deposit instructions allows attackers to counterfeit `FundsDeposited` and `FundsDepositedWithMessage` events

Severity: Critical

In the `IDeposit`, `IDepositMessage` and `IDepositInfo` instructions, defined respectively in `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/i_deposit.rs`, `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/i_deposit_message.rs` and `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/i_deposit_info.rs`, tokens are transferred from the user to an escrow, followed by the emission of a `FundsDeposited` event which is utilized by the other components of the protocol to handle the deposit.

However, the information reported in this event could not be trusted, as `deposit_data` is provided directly by the sender without performing any verification in the instruction logic.

For instance, the sender can specify arbitrary values for `dest_token`, `dest_amount`, and `src_token`. This would allow an attacker to deposit a small amount of a worthless token and emit an event claiming a deposit of a large amount of any other token.

Recommendation

We recommend validating information stored in `deposit_data` before emitting the event.

Status: Resolved

The client states that it is not possible to verify `dest_token` and `dest_amount` in this contract. Malicious operations would be blocked by the forwarder.

5. Missing prefixes in PDA seeds allow attackers to manipulate account data

Severity: Critical

The protocol widely leverages PDAs for initializing and manipulating accounts. A PDA (Program Derived Address) is an address that is derived from the program's address using a set of provided seeds and bumps.

Its key characteristic is that there is no private key associated with it so it can be safely used as a storage for programs, similar to a `HashMap`.

However, it was identified that there are many occurrences of a PDA initialization using seeds provided by the user as instruction arguments without prefixing them.

Such an implementation means that the protocol can create only a single PDA account using a specific set of seeds regardless of the instruction that would be creating that account.

Various instances of this behavior were identified and are as follows:

- The `initialize_request_payload_account` instruction handler in `gateway-contracts:solana/programs/gateway/src/instructions/args_account.rs:125-140` uses the user-provided seeds to create a `RequestPayloadAccount`. However, the same program also defines a `set_dapp_metadata` instruction meant to be used by dApps to integrate with the protocol by using the predefined prefix with the signer's key as seeds. The user can execute the `initialize_request_payload_account` instruction with the seeds matching those used in the `set_dapp_metadata` instruction by manually specifying them to be equal to the predefined prefix along with some key. As a consequence, a malicious user can initialize an account with the PDA meant for the dApp integration while making it a `RequestPayloadAccount`. Effectively, such action makes future integration for the dApp not possible as the account the program would create for integration already exists.
- The `initialize_packet_account` instruction handler in `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/args_account.rs` uses the user-provided seeds to create a `PacketAccount`. The same program also implements a `grant_role` instruction which creates an `EmptyAccount` portraying a particular role within the program. The

`grant_role` instruction uses seeds consisting of a role name and the role holder's public key. Consequently, a malicious user can use the `initialize_packet_account` instruction and specify the seeds to match a particular role name and public key of a given user. As a result, a malicious user can prevent assigning a role to the victim, as the PDA that their role account would create is already taken by a `PacketAccount`.

- The `i_relay_message` instruction handler in `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/i_relay_message.rs` uses user-provided seeds to create an `IRelayMessageExecution` account. Similarly, the `IRelay` instruction handler in `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/i_relay.rs` is using the user-provided seeds to create an `EmptyAccount` serving as `msg_hash_account`. Using one set of seeds for one of these instructions will automatically make it impossible to use them in the other one. A malicious user could frontrun a legitimate user's transaction with one of those instructions. As a consequence, a legitimate user's transaction would not be processed successfully as the calculated PDA would correspond to the already initialized account.

- The `initialize_signature_account` instruction handler in `gateway-contracts:solana/programs/gateway/src/instructions/args_account.rs:64-69` uses the user-provided seeds to create a `SignatureAccount`. The same program also implements a `grant_role` instruction which creates an `EmptyAccount` portraying a particular role within the program. The `grant_role` instruction uses seeds consisting of a role name and the role holder's public key. Consequently, a malicious user can use the `initialize_signature_account` instruction and specify the seeds to match a particular role name and public key of a given user. As a result, a malicious user can prevent assigning a role to the victim, as the PDA that their role account would create is already taken by a `SignatureAccount`.

Recommendation

We recommend specifying a custom prefix for all defined PDAs so that users cannot maliciously or accidentally create an account using seeds that would be expected in a different instruction.

Status: Resolved

6. Users can arbitrarily define the fee required by the `IDepositInfo` instruction

Severity: Critical

The `IDepositInfo` instruction handler, defined in `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/i_deposit_info.rs`, takes a user-provided `fee_amount` as one of the parameters.

During the execution, the instruction calls an `_is_in_limit` function to verify that the provided `fee_amount` is less than or equal to a specified `MAX_TRANSFER_SIZE` constant value. No other verification of the `fee_amount` is performed, which allows a user to specify an arbitrarily low value, including `0`. Such a scenario would result in the user paying no fee.

However, the `fee_amount` is present in the emitted `DepositInfoUpdate` event. Depending on other protocol's components that are consuming emitted events, there are two possible scenarios:

1. If the `fee_amount` is not verified in other components, then the user can complete the whole process without paying any fees.
2. If the `fee_amount` is verified and the protocol will prevent the process from completing successfully if the fee is too small. In this case, it will lead to a loss of funds for the user, since it would not be possible to retrieve it.

Recommendation

We recommend implementing a verification mechanism that will assure the fee is within a reasonable range, including both minimal and maximal values.

Status: Acknowledged

The client states that passing a low fee will not affect the system. This function is used to increase the fee for previously pending transactions due to a low fee, users will continue to lose funds if low values are passed knowingly.

7. Lack of access control allows attackers to overwrite data or execute restricted actions

Severity: Major

In the following instances, due to a lack of access control, any `signer` account can invoke crucial instructions that overwrite data or execute actions that should be permissioned:

- The `DappSetDappMetadata` instruction in `dapp/src/lib.rs`.
- The `CreateRequestPayloadAccount` instruction in `gateway-contracts:solana/programs/gateway/src/instructions/args_account.rs` allows to overwrite any `request_payload_account` data.

- The `CreateCrossChainAckPayloadAccount` instruction in `gateway-contracts:solana/programs/gateway/src/instructions/args_account.rs` allows to overwrite any `crosschain_ack_payload_account` data.
- All `UpdatePacketAccount`-based instructions in `gateway-contracts:solana/programs/gateway/src/instructions/args_account.rs` and `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/args_account.rs` allow to take over anyone's previously initialized `packet_account` as long as it is new, i.e. was not written to yet. It is recommended to set the creator already in the preceding `InitializePacketAccount` instruction.
- The `UpdateSignatureAccount` instruction in `gateway-contracts:solana/programs/gateway/src/instructions/args_account.rs` allows to take over anyone's previously initialized `signature_account` as long as it is new, i.e. `get_signatures_length() == 0`. It is recommended to set the creator already in the preceding `InitializeSignatureAccount` instruction.
- The `IAck` instruction in `gateway-contracts:solana/programs/gateway/src/instructions/i_ack.rs`.
- The `MintRouteToken` instruction in `gateway-contracts:solana/programs/gateway/src/instructions/i_receive.rs`.
- The `CompleteIReceive` instruction in `gateway-contracts:solana/programs/gateway/src/instructions/i_receive.rs`.

Recommendation

We recommend constraining the allowed signers appropriately such that the above instructions can only be invoked by the entitled accounts.

Status: Resolved

8. Missing ownership validation for `PacketAccount`

Severity: Major

The `IDepositMessage` instruction, defined in `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/i_deposit_message.rs`, allows users to deposit a token and relay a message to the destination chain.

The message is stored in a `PacketAccount`, which has been previously created and initialized by the sender.

However, since the instruction handler does not check the `creator` of the `packet_account`, there is no validation for the ownership of this account.

Consequently, a malicious front end or application could provide a different account in the instruction context to send fraudulent messages on behalf of the signer.

Recommendation

We recommend validating the `PacketAccount` ownership.

Status: Resolved

9. Multiple emissions of the `DepositInfoUpdate` for the same `deposit_id` are permitted

Severity: Major

The `IDepositInfo` instruction, defined in `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/i_deposit_info.rs`, allows users to emit a `DepositInfoUpdate` event by targeting a specific `deposit_id`.

However, there is no check in place to ensure that multiple and discordant `DepositInfoUpdate` events are emitted for the same `deposit_id`.

Consequently, it could be possible to initiate multiple withdrawals or generically emit multiple `DepositInfoUpdate` targeting the same `deposit_id`, leading to an inconsistent state.

Recommendation

We recommend not allowing the emission of multiple `DepositInfoUpdate` for the same `deposit_id` before the previous one is handled.

Status: Acknowledged

The client states that on the source chain, it is not possible to know if the previous transaction was relayed on the destination chain unless a method is created to explicitly set it on the source chain, which is cost-intensive.

The client explains that since all state is managed on Router Chain, even if the user increases the fee by calling `i_deposit`, they can still withdraw from Router Chain.

10. The `ISend` instruction can be executed if the program is paused

Severity: Major

The `i_send` instruction defined in `gateway-contracts:solana/programs/dapp/src/lib.rs` does not validate if the program is paused.

As a consequence, if the program is paused, for instance, for maintenance or due to an incident that has occurred, the `i_send` instruction would still be executable, which might lead to an unexpected outcome.

Recommendation

We recommend adding a `_when_unpause` call at the beginning of the `i_send` instruction implementation to ensure that it can only be executed if the protocol is not paused.

Status: Resolved

11. Admin role can be removed

Severity: Major

In the following locations:

- `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/access_control.rs`
- `asset-forwarder-contracts:solana/programs/external_bridge/src/instructions/access_control.rs`
- `gateway-contracts:solana/programs/dapp/src/lib.rs`
- `gateway-contracts:solana/programs/gateway/src/instructions/access_control.rs`
- `asset-bridge-contracts:solana/programs/asset_bridge/src/instructions/access_control.rs`

the execution of the `RevokeRole` instruction can inadvertently be used to revoke the `ADMIN` role. This occurs because the seed format of the other roles is the same as the `ADMIN` role.

This behavior may be unintended and poses a risk of removing all admins leading to the impossibility of managing the program configurations and roles.

Recommendation

We recommend implementing checks to prevent the revocation of all admins and ensuring that the `RevokeRole` instruction cannot manipulate the `ADMIN` role in unintended ways.

Status: Resolved

12. Possible protocol DoS due to accidental account closure

Severity: Major

It was observed that the protocol is transferring native tokens from accounts owned by the programs. Those accounts usually contain data critical to overall bridge execution, for instance, the nonces used in event emissions.

If the lamports balance of a certain account falls below the rent-exemption threshold, Solana will automatically close the account, effectively removing any data stored in it and returning the leftover lamports, if any, to the accounts creator.

Such accounts could then be recreated if needed. However, they will contain the original default values, as per the instructions that create them. Such values will likely be out-of-sync with other components used by the protocol, including the off-chain ones, which might render the bridge unusable.

The identified places with unchecked lamports transfer are as follows:

- `gateway-contracts:solana/programs/gateway/src/instructions/rescue.rs:44-50`
- `asset-bridge-contracts:solana/programs/asset_bridge/src/instructions/rescue.rs:44-50`
- `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/rescue.rs:45-51`
- `asset-forwarder-contracts:solana/programs/external_bridge/src/instructions/rescue.rs:47-54`
- `asset-bridge-contracts:solana/programs/asset_bridge/src/instructions/unstake.rs:95-101`

Recommendation

We recommend introducing a check that will assure accounts stay rent-exempt after the lamports transfer. Alternatively, we recommend using a wrapped SOL instead of native SOL to eradicate the issue.

Status: Resolved

13. Missing validation in the `CreateValsetArgsAccounts` instruction

Severity: Minor

In `gateway-contracts:solana/programs/gateway/src/instructions/args_account.rs:17-42`, the `create_valset_args` function handler of the

`CreateValsetArgsAccounts` instruction allows storing a vector of validators and their respective voting powers.

However, there is no logic to deduplicate these entries, permitting the same validator to be stored multiple times. Additionally, there is no verification to ensure that `power` values are within a meaningful range.

Recommendation

We recommend implementing deduplication logic to prevent the same validator from being stored multiple times and validating the `power` values.

Status: Acknowledged

The client states that the valset will only be updated if the contract gets 2/3 power or through the update valset function, where the current checkpoint matches previously stored information.

This information is set during initialization. Even if duplication occurs during initialization, functions like `i_receive`, `i_ack`, and `update` will not proceed because a duplication check is already in place on Router Chain.

The client notes that, as the deployer, they will keep this in mind when deploying the contract.

14. Missing role segregation

Severity: Minor

The `GrantRole` and `Revoke` role instructions do not enforce segregation between different roles to maintain distinct administrative boundaries allowing the same account to potentially assume the responsibilities of multiple roles such as `ADMIN`, `PAUSER`, and `RESOURCE_SETTER`.

As a consequence, this lack of role differentiation could lead to a concentration of power and responsibilities, undermining the principle of least privilege.

Recommendation

We recommend maintaining separate accounts for each distinct role.

Status: Resolved

15. Permissionless account initialization

Severity: Minor

In the following instances, any `signer` account can invoke initialization-related instructions during the first invocation. This action assigns ownership or an admin role to the caller, allowing them to partially control the protocol, which may necessitate a redeployment:

- The `Initialize` instruction in `gateway-contracts:solana/programs/dapp/src/lib.rs` allows compromising the `admin_pda_account`.
- The `Initialize` instruction in `asset-bridge-contracts:solana/programs/asset_bridge/src/instructions/initialize.rs`, `asset-forwarder-contracts:solana/programs/external_bridge/src/instructions/initialize.rs`, `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/initialize.rs`, and `gateway-contracts:solana/programs/gateway/src/instructions/initialize.rs` allows compromising the `admin_pda_account`.
- The `Initialize` instruction in `asset-forwarder-contracts:solana/programs/message_handler/src/lib.rs` and `asset-bridge-contracts:solana/programs/message_handler/src/lib.rs` allows compromising the `message_handler_account`.

Recommendation

We recommend constraining the allowed signers appropriately such that the above instructions can only be invoked by the entitled accounts. Alternatively, this can be resolved by combining program deployment and necessary initialization instructions within one transaction.

Status: Resolved

16. Template-like implementations

Severity: Minor

Many programs defined across the codebase seem to have template-like implementations. Instructions simply return an `Ok` variant of the `Result` type with no meaningful logic.

Those programs likely serve the purpose of a template for developers wishing to use the protocol. However, this was not specified in the code. In case they are not templates, all implementations can be considered as a `no-op`.

The following programs were identified to have template-like implementations:

- `asset-bridge-contracts:solana/programs/message_handler/src/lib.rs:handle_message`

- `asset-forwarder-contracts:solana/programs/message_handler/src/lib.rs:handle_message`
- `gateway-contracts:solana/programs/asm/src/lib.rs:verify_cross_chain_request`

Recommendation

We recommend documenting, enhancing, and explicitly marking all educational code.

Status: Acknowledged

The client states that these contracts are solely for testing purposes and do not contain any logic in their respective functions.

17. Invalid signatures are not skipped in the `check_validator_signatures` function

Severity: Minor

The `check_validator_signatures` function in `gateway-contracts:solana/programs/gateway/src/_impl.rs:154-181` intends to skip invalid signatures indicated by `v == 0` when it was not possible to get a signature from a validator, according to an inline comment.

However, skipping such signatures is not implemented and every signature is checked via the `verify_sig` function which fails on invalid signatures.

Consequently, one invalid signature can revert the whole signature verification instead of being skipped.

Recommendation

We recommend skipping the call to `verify_sig` in case the signature's v-value is zero. This can be done since the cumulative power of all valid signatures is checked against a threshold at the end of the function.

Status: Resolved

18. Lack of validation for protocol fees

Severity: Minor

The fee mechanism implemented in the protocol depends on the fee values set in storage.

However, there are no checks implemented on the program level that would ensure the fee is set at a reasonable value.

Consequently, it is possible to set the fee value to an unreasonable amount. Not only could it equal 100%, but it also could be set to an even greater value that would cause math issues in the protocol.

The identified fee setters are as follows:

- `gateway-contracts:solana/programs/gateway/src/instructions/setter.rs:31`
- `gateway-contracts:solana/programs/dapp/src/lib.rs:61`
- `asset-forwarder-contracts:solana/programs/external_bridge/src/instructions/setter.rs:67`

Recommendation

We recommend implementing a verification mechanism that will ensure fees are within a reasonable range.

Status: Acknowledged

19. The `verify_sig` function is susceptible to signature malleability

Severity: Minor

The `verify_sig` function in `gateway-contracts:solana/programs/gateway/src/signature_utils.rs:66-76` which relies on the subsequent `ecrecover_to_eth_address` function allows both low-s and high-s signatures.

Therefore, there always exist two valid signatures for one digest to recover the correct address.

We are reporting this issue with minor severity since while we have not found direct security implications, it could cause issues in future contract revisions.

Recommendation

We recommend rejecting high-s signatures in case they are not desired according to the example in Appendix B-1.

Status: Acknowledged

20. Usage of native token is discouraged

Severity: Minor

The protocol allows the use of the native SOL token within its defined functionalities which is generally considered a bad practice.

Namely, it introduces a risk of accidentally closing an account if its lamports balance falls below the rent-exemption threshold. Additionally, it increases the code complexity, as a separate transfer logic needs to be implemented for native transfers.

Recommendation

We recommend using a Wrapped SOL token instead of native SOL.

Status: Acknowledged

21. Potential mismatch between validators and powers in `check_validator_signatures` function

Severity: Minor

The `check_validator_signatures` function in `gateway-contracts:solana/programs/gateway/src/_impl.rs` assumes that the `self.powers` array in the state corresponds directly to the `validators` array passed in via the parameters, with both arrays indexed in the same order.

However, there is no check to ensure that the `validator` at index `i` in the `validators` array matches the power at index `i` in the `self.powers` array, creating a potential mismatch between validators and their corresponding powers.

Recommendation

We recommend adding a validation step to ensure that each validator in the `validators` array is correctly matched with its corresponding power in the `self.powers` array.

Status: Acknowledged

22. Lack of differentiation for different request types

Severity: Informational

The `_is_execute_record` function defined in `asset-bridge-contracts:solana/programs/asset_bridge/src/instructions/i_receive.rs` is responsible for checking if a particular record has already been handled.

However, two of the execution flows resolve to calling the `_handle_record` function with the same arguments, which may lead to a potentially invalid check.

Recommendation

We recommend verifying if those separate execution flows are effectively the same. If they are not, we recommend changing the arguments to appropriate ones. In case the current implementation is correct, we recommend documenting this via a comment in the code.

Status: Acknowledged

23. Generic roles can be set

Severity: Informational

In the following locations:

- `asset-forwarder-contracts:solana/programs/asset_forwarder/src/lib.rs`
- `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/access_control.rs`
- `asset-forwarder-contracts:solana/programs/external_bridge/src/instructions/access_control.rs`
- `router-gateway-contracts:solana/programs/dapp/src/lib.rs`
- `router-gateway-contracts:solana/programs/gateway/src/instructions/access_control.rs`
- `asset-bridge-contracts:solana/programs/asset_bridge/src/instructions/access_control.rs`

the `grant_role` and `revoke_role` functions allow the admin to set and revoke any role since roles are defined as strings.

This flexibility, however, can lead to inconsistencies and potential misuse because roles are not enumerated.

Recommendation

We recommend enumerating the roles (e.g., `PAUSER`, `RESOURCE_SETTER`) to enforce consistent role management.

Status: Resolved

24. Lack of event emission during configuration updates

Severity: Informational

The management functionalities related to the programs' configuration like their initialization, configuration, or pause status updates do not emit events informing about such changes.

It is considered a best practice to emit events on every configuration change to inform users and other parties that such an action took place.

The instructions that lack this event emission are:

- The `initialize` instruction from `gateway-contracts/solana/programs/gateway/src/instructions/initialize.rs`.
- The `pause`, `unpause` and `switch_pause` functions in `asset-forwarder-contracts:solana/programs/external_bridge/src/instruction/pausable.rs`.

Recommendation

We recommend implementing event emission on every successful management function execution.

Status: Resolved

25. TODO comments across the codebase

Severity: Informational

In several instances of the codebase, TODO comments are found:

- `asset-bridge-contracts:solana/programs/asset_bridge/src/instructions/i_receive.rs:179`
- `asset-bridge-contracts:solana/programs/asset_bridge/src/instructions/i_transfer_token.rs:13`
- `gateway-contracts:solana/programs/gateway/src/instructions/i_send.rs:77`
- `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/i_deposit.rs:10`
- `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/i_receive.rs:22`

This suggests that some improvements or functionality have not been implemented in these places. While this does not pose a security risk in itself, it may assist a potential attacker in creating attack vectors against the protocol.

Recommendation

We recommend resolving the TODO comments or removing them.

Status: Resolved

26. Miscellaneous comments

Severity: Informational

Miscellaneous recommendations can be found below.

Recommendation

The following are recommendations to improve the overall code quality and readability:

- In `asset-forwarder-contracts:solana/Anchor.toml:17`, the `wallet` field is set to a path related to a specific dev environment. We recommend not having specific machine paths hardcoded in production.
- In `asset-bridge-contracts:solana/programs/asset_bridge/src/instructions/i_receive.rs:201`, the `AccountDeserialize::try_deserialize` is used. This function only checks Anchor's discriminator. Technically any account with data value that contains the first 8 bytes matching the expected discriminator value would be deserialized correctly. The `try_from` function should be used instead, which will also check the account's owner. In this case, there is no significant security impact as this code is executed only in instructions signed by the `gateway_authority`.
- The Anchor version in use has dependencies that are affected by vulnerabilities such as `ed25519-dalek` which is affected by RUSTSEC-2022-0093. We recommend updating Anchor to the latest version.
- The Anchor version in use has dependencies that are affected by vulnerabilities such as `curve25519-dalek` which is affected by RUSTSEC-2024-0344. We recommend updating Anchor to the latest version.
- In `gateway-contracts:solana/programs/dapp/src/lib.rs:412` the `pub` keyword is missing for the account definition.
- In `gateway-contracts:solana/programs/dapp/src/lib.rs:461` the `signer_associate_account` is not associated with the `signer` via any seed or constraint.
- In `gateway-contracts:solana/programs/gateway/src/instructions/rescue.rs:1` the `AUTHORITY` constant is unused.
- In `gateway-contracts:solana/programs/dapp/src/lib.rs` and `asset-bridge-contracts:solana/programs/asset_bridge/src/instructions/set_dapp_metadata.rs`, the `set_dapp_metadata` instruction handler neglects to check for equivalence of the `fee_payer_account` and the provided `fee_payer` string argument.

- In the `IDepositUSDC` instruction, specifically in `asset-forwarder-contracts:solana/programs/external_bridge/src/instructions/i_deposit_usdc.rs:14` the `partner_id` is defined within the instruction macro but remains unused.
- The `CHUNK_LIMIT` constant defined in `asset-bridge-contracts:solana/programs/asset_bridge/src/constants.rs:8` is unused.
- In `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/i_deposit_message.rs:85` it is assumed during event emission that the `depositor` and `refund_recipient` are always the same.
- In `asset-bridge-contracts:solana/programs/asset_bridge/src/_internal.rs:304-305` the specified `route_amount` is zero and the `route_recipient` is an empty string.
- The `RoleAccount` defined in `gateway-contracts:solana/programs/dapp/src/lib.rs` contains a `bool` member called `have`. The content of this account is not checked anywhere in the codebase, nor is it necessary to exist as the role can be verified by the account's existence. As an optimization, the `have` member should be removed.
- In `asset-bridge-contracts:solana/programs/asset_bridge/src/instructions/i_receive.rs:135`, the `idx` variable is updated but its value remains unused.
- On each relay instruction in `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/i_relay_message.rs:19`, `112` and `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/i_relay.rs:20` new accounts (`i_relay_message_execution`, `msg_hash_account`) are initialized. However, those accounts are never closed although they are only used temporarily until the whole bridging process is finalized.
- In every instance of the `_when_pause` function throughout the codebase, the `PausableError::Paused` is returned in case the program is not paused. However, the `PausableError::UnPaused` was explicitly defined and intended for this case.
- The `IDepositMessage` instruction handler implements a verification mechanism on the amount of tokens the user wishes to send. The verification uses the `_is_in_limit` function that assures the amount of tokens is less than or equal to the specified `MAX_TRANSFER_SIZE` constant value. Such a mechanism limits the flexibility of the protocol, as if a user wished to transfer more tokens, it would require

splitting it into more than one process. We recommend removing the limit on the maximal amount of tokens that could be transferred unless there is a specific business requirement for its existence.

- The `IReceive` instruction handler signature contains two arguments: `request_sender` and `src_chain_id`. Those arguments are not used anywhere in the function. We recommend either removing them if they are not necessary or utilizing them to implement additional verification.
- The `initialize` instruction handler in `asset-forwarder-contracts:solana/programs/external_bridge/src/instructions/initialize.rs` is setting the `usdc_token` value to a provided `cctp_local_token`. However, the USDC token has one known `Pubkey` which could be hardcoded to mitigate errors. Similarly, the `set_usdc_token` function in `asset-forwarder-contracts:solana/programs/external_bridge/src/instructions/setter.rs` is used to modify that value, which is not necessary as USDC `Pubkey` is known.
- The current implementation does not take into account that `USDC` token might introduce a fee-on-transfer mechanism in the future.
- The `i_receive` instruction handler does not emit an appropriate event and does not implement the checks mentioned in the comments in `asset-bridge-contracts:solana/programs/asset_bridge/src/instructions/i_receive.rs:179-182`.

Status: Partially Resolved

Appendix A: Test Cases

1. Test case for **"Missing `TokenAccount` validation allows attackers to circumvent fees and fake transfers and deposits"**

In `asset-forwarder-contracts:solana/programs/asset_forwarder/src/instructions/i_deposit.rs`, the `IDeposit` instruction allows the user to deposit a token in the asset forwarder.
However, there is no validation to ensure that the `depositor_associate_token` account refers to the same `mint`, `token_program`, and `authority` of the specified token.
This oversight could allow attackers to use a malicious `depositor_associate_token` to deposit a different token than the one specified in `deposit_data` as described in this test case:

```
it("i_deposit_from_asset_forwarder_HACK", async () => {
        const tokenInfo = testClient.tokens["USDT"];
        const beforeBalance = await testClient.tokenBalance(
            tokenInfo.deployerAssociatePda,
        );
        const depositData = {
            partnerId: new anchor.BN(1),
            amount: new anchor.BN(10),
            destAmount: new anchor.BN(10),
            srcToken: tokenInfo.mint,
            refundRecipient: testClient.deployer.publicKey,
            destChainIdBytes: Array.from(
                ethers.getBytes(

"0x3500000000000000000000000000000000000000000000000000000000000000",
                ),
            ),
            decimals: tokenInfo.decimal,
        };
        await testClient.approveChecked(
            tokenInfo.mint,
            "USDT",
            depositData.amount,
            testClient.assetForwarderAccount,
        );
        const dst_token = Buffer.from("dst_token");
        const recipient = Buffer.from("recipient");
        const signature = await
```

```
testClient.asset_forwarder_program.methods
            .iDeposit(depositData, dst_token, recipient)
            .accounts({
                assetForwarderAccount: testClient.assetForwarderAccount,
                depositor: testClient.deployer.publicKey,
                depositorAssociateToken: tokenInfo.deployerAssociatePda,
                assetForwarderAssociateToken:
tokenInfo.deployerAssociatePda, //Same associate token account
                mint: tokenInfo.mint,
                tokenProgram: TOKEN_2022_PROGRAM_ID,
                associatedTokenProgram: ASSOCIATED_TOKEN_PROGRAM_ID,
                systemProgram: SystemProgram.programId,
            })
            .signers([testClient.deployer])
            .rpc({
                commitment: "confirmed",
            });
        const fundDepositedEvent = await testClient.getEvent(
            signature,
            [testClient.asset_forwarder_program],
            "FundsDeposited",
        );
        console.log(fundDepositedEvent);
        const afterBalance = await testClient.tokenBalance(
            tokenInfo.deployerAssociatePda,
        );
        assert(beforeBalance.eq(afterBalance));
    });
```

Appendix B: Examples

1. Implementation example for **“The `verify_sig` function is susceptible to signature malleability”**

```
let signature =
libsecp256k1::Signature::parse_standard_slice(&instruction.signature)
   .map_err(|_| ProgramError::InvalidArgument)?;

if signature.s.is_high() {
   msg!("signature with high-s value");
   return Err(ProgramError::InvalidArgument);
}
```