Security Audit – deBridge on Solana

Neodyme AG

March 11, 2022

Contents

Introduction

deBridge engaged Neodyme to do a detailed security analysis of their smart contract on Solana. A brief initial review was done in November 2021. The full audit took place in the month leading up to the deBridge launch, beginning on the 7th of February 2022 and ending on the 11th of March 2022.

The initial review and audit both revealed one severe vulnerability, as well as some medium and low-priority findings, all of which were resolved before the bridge went live.

In this report, we outline the most relevant findings.

Project Overview

deBridge can be summarized in one sentence as follows:

> deBridge is a cross-chain interoperability and liquidity transfer protocol.

The bridge itself is live, and already links many blockchains together. Solana support is now implemented but not yet live and was the target for this audit.

The Solana part of DeBridge allows sending and receiving bridged assets to and from other blockchains. Conceptually, this works by locking native funds on the source chain into the deBridge contract and then minting wrapped tokens on the target chain. These can be transferred back at any time, so each wrapped token is 1:1 backed by a native token.

To make this system secure, deBridge uses a number of oracles. They watch for deBridge sent events on all supported chains and create signatures. With enough signatures, an event can be claimed on another chain.

To make the bridge user-friendly, the claim process can be automated via executors, which get a user-selectable percentage of the transfer amount as a fee.

Further, deBridge supports external calls. These transfer calls have instruction data attached, which the executors will automatically execute for the user.

On the Solana side, there are two contracts. One is responsible for oracle signature verification and settings; the other escrows/mints funds and executes external calls. Both contracts are written with the up-to-date Anchor Framework, currently the leading framework for writing Solana contracts.

Scope

The scope for this audit were deBridge's two Solana contracts. At the time of audit, they were developed in a private GitHub repository: debridge-finance/solana-contracts. The newest commit hash on 7th of Feb 2022 was bb89c0dfe9dfc77985a5b4a2ca305d322636f828. After all fixes were applied, the hash was 1e4e30e12fa8e4a104d3dfc11a0f10883c521f42.

Cross-chain projects inherently have a larger attack surface than single-chain projects. A vulnerability in the oracles or a contract of another chain can also affect the Solana part. As this audit only includes the Solana contracts, certain assumptions on the other protocol parts are made. In particular, we assume all other chains to be secure, all other deBridge contracts to be secure, and the oracle network to be secure. This boils down to: "All messages with enough valid signatures have the guarantee that the transferred tokens are 1:1 backed on the native chain, and can be transferred back at any time".

Methodology

Neodyme's audit team, which consists of security engineers with extensive experience in Solana smart contract security, reviewed the code of the on-chain contract, paying particular attention to the following:

- Ruling out common classes of Solana contract vulnerabilities, such as:
  - Missing ownership checks
  - Missing signer checks
  - Signed invocation of unverified programs
  - Solana account confusions
  - Re-initiation with cross-instance confusion
  - Missing freeze authority checks
  - Insufficient SPL token account verification
  - Missing rent exemption assertion
  - Casting truncation
  - Arithmetic over- or underflows
  - Numerical precision errors
- Checking for unsafe design, which might lead to common vulnerabilities being introduced in the future
- Checking for any other, as-of-yet unknown classes of vulnerabilities arising from the structure of the Solana blockchain
- Ensuring that the contract logic correctly implements the project specifications
- Examining the code in detail for contract-specific low-level vulnerabilities
- Ruling out denial-of-service attacks
- Ruling out economic attacks
- Checking for instructions that allow front-running or sandwiching attacks
- Checking for rug-pull mechanisms or hidden backdoors
- Checking for replay protection
- Verifying the signature validation

Roots of Trust

Apart from the security of the smart contracts, there are different kinds of authorities that have to be trusted in the deBridge-Solana ecosystem. Neodyme has not investigated these authorities.

THE ORACLES are one of the most important actors. All bridges fully trust them. If they get tricked into signing things they shouldn't sign, a bridge might mint wrapped tokens or release native tokens where it should not.

In deBridge, there are two types of oracles: Required and optional ones. Of the required oracles, all have to sign a message. As such, each required oracle could halt the protocol if it wants.

The Protocol Authority can change the oracle keys and required signature counts.

THE PROTOCOL AUTHORITY has full control over all settings and oracles.

- It can add and remove oracles and change related settings
- It can add/change/disable chains
- It can change fees

It thus has full control over all funds.

THE STOP TAP is an authority that has the permission to pause bridges but not to resume them again. A paused bridge has deposits and withdrawals disabled.

As the other authorities should be well protected and thus somewhat slow to access, having a separate stop authority allows for quick reactions in case any bridge urgently needs to be stopped.

THE UPGRADE AUTHORITY can upgrade the Solana contracts. On Solana, most contracts are deployed upgradable. This allows for future contract modifications, be it new features or bug fixing. It inherently means that the contract's code can be fully replaced whith all its advantages and drawbacks. The upgrade authoritys' signature guards this upgrade process.

The upgrade authority thus has full control over the contract and, therefore, all funds.

THE EXECUTORS are crankers, that automatically claim deBridge transactions on behalf of users. They are not required to be trusted, as the deBridge contract enforces that the user's funds only get used in the user-specified way.

Findings

This section discusses the overall design of deBridge's Solana contracts, followed by a detailed description of all our findings and their resolutions.

deBridge takes great care not to run into one of Solana's most common sources of vulnerabilities: account confusions and missing signer and owner checks. They use the Anchor framework, which enforces account ownership and type, and additionally, only use program-derived addresses (PDAs), whose seeds are checked in all cases.

They follow good commit practices and nicely comment on all external instructions, making protocol roles very clear.

The fact that the protocol is split into two contracts increases the attack surface compared to a single contract but makes it more flexible in the future. The most complicated part of the contract is the send and receive logic, as there are many different supported flows: simply send tokens, send tokens with an attached external call, of which the data can either be hashed or plain, and which can be aborted if a certain key signs.

All findings are classified in one of four severity levels:

- **Critical**: Bugs that will likely cause loss of funds. This means that an attacker can, with little or no preparation, trigger them, or they are expected to happen accidentally. Effects are difficult to undo after they are detected.
- **High**: Bugs, which can be used to set up a loss of funds in a more limited capacity, or to render the contract unusable.
- **Medium**: Bugs that do not cause direct loss of funds but lead to other exploitable mechanisms.
- **Low**: Bugs that do not have a significant immediate impact and could be fixed easily after detection. It also contains bad practices which in the current program state are fine but could cause issues in the future.

Name	Severity	Status
Signature Verification Bypass	Critical	Resolved
Non-Unique Bump Seeds	Critical	Resolved
Account Creation DoS	Medium	Resolved
Broken Fallback Mechanism for External Calls	Medium	Resolved
Storage Accounts are Missing Discriminator	Low	Resolved
Solana Logs can get Truncated	Info	Info

Signature Verification Bypass (Critical; Resolved)

Severity	Impact	Affected Component	Status
Critical	Loss of Funds	Signature Verification	Resolved

The intended way to verify signatures inside a Solana contract is notoriously hard to use. Most projects using it have had a bug in their implementation, and deBridge is no different. By lacking proper verification on a crucial field, anyone could bypass signature verification and thus redeem arbitrary deBridge transfers.

Signature checking is an expensive operation, and Solana has implemented it so that it is easy for the Solana-validator to verify: With a separate Secp256k1 Program. The intended use-case is to call this program in your transaction, which will only succeed if all included signatures are valid. The signature consumer then has to use instruction introspection via the instruction-sysvar to figure out if there were any signatures and on which data they were. The consumer knows the signatures are valid, as otherwise, the Solana runtime would have aborted the transaction execution.

To be efficient with the instruction data size, the Secp256k1 program allows you to sign data from any instruction in the current TX. This is implemented with a header containing a list of SecpSignatureOffsets, which in turn point to the message, pubkey, and signatures:

```
pub struct SecpSignatureOffsets {
    pub signature_offset: u16,              // offset to [signature, recovery_id] of 64+1 bytes
    pub signature_instruction_index: u8,    // instruction in which to read at the specified offset
    pub eth_address_offset: u16,            // offset to eth_address of 20 bytes
    pub eth_address_instruction_index: u8, // instruction in which to read at the specified offset
    pub message_data_offset: u16,           // offset to start of message data
    pub message_data_size: u16,             // size of message data
    pub message_instruction_index: u8,      // instruction in which to read at the specified offset
}
```

Most important, this struct contains not only message_data_offset and message_data_size, but also message_instruction_index.

The deBridge implementation did not verify the instruction_index, and assumed that it is always equal to the current Secp256k1 IX. Since this does not have to be the case, an attacker can cause a

mismatch between the message that the Secp256k1 program thinks is signed and the message that the deBridge program thinks is signed.

To do so, he needs one valid set of oracle signatures of any message of the correct length. This is easy to get; just do a valid test transaction. Now the attacker calls deBridges claim function again, specifies the same signatures, but places the original signed message in a tailing memo instruction. The attacker then sets the instruction indices to said memo instruction, so the signature verification program will succeed. From its point of view, the message has valid signatures.

The DeBridge will always read the message from the Secp256k1 IX, which now can be completely arbitrary of the correct length. If the original message was a transfer message, an attacker could thus mint an unlimited number of deBridge tokens.

Fix deBridge quickly fixed this bug by adding a check that signature_instruction_index, eth_address_instruction_index and message_instruction_index are all equal to the instruction index of the Secp256k1 instruction. Neodyme verified the fix.

In addition, we opened an issue aiming to improve Solanas API for everyone: Secp256k1 Program is hard to safely use.

Non-Unique Bump Seeds (Critical; Resolved)

Severity	Impact	Affected Component	Status
Critical	Loss of Funds	Replay Protection	Resolved

In the deBridge protocol, all cross-chain transactions have a unique submission_id. For replay protection, deBridge allocates a submission account derived from said ID whenever one is redeemed. This guarantees that each ID is only redeemed once.

During our brief initial review, while deBridge was still under heavy development, deBridge had a crucial error here, which allowed a bypass of the replay protection.

The submission account is a program-derived address (PDA). These addresses are the result of a hash, which gets the program_id and program-specified seeds as inputs. To enforce that there can never be a collision between a user wallet and a PDA, Solana enforces that all PDAs are “off-curve”, i.e. that they are not valid points in Solanas cryptosystem. In practice, this is done by adding an 8 bit bump seed to the end of the seeds.

For any specific seed, there are multiple valid bump seeds. Solana programs usually use the canonical seed, the lowest valid one. deBridge allowed the user to specify the seed himself, which meant multiple submission accounts could be created for the same transaction.

Fix deBridge quickly fixed the issue by always using the canonical bump-seed by calculating it inside the program with Solanas Pubkey::find_program_address. Neodyme verified the fix.

Account Creation DoS (Medium; Resolved)

Severity	Impact	Affected Component	Status
Medium	Block transactions from being redeemed	Replay Protection	Resolved

As described in the previous bug, the replay protection works by creating a submission account. The initial implementation was faulty, letting anyone "create" such a submission account, thus blocking the claim of a specific transaction. The transaction could be unblocked by fixing this issue afterward and upgrading the program.

There were two different but related issues with the same cause: Anyone can transfer an arbitrary amount of lamports to any account.

Firstly, deBridge used system_instruction :: create_account to create the submission accounts. This is an issue because the instruction fails if the target account already has lamports.

Secondly, the check that a submission account already exists was also done by checking if the submissions account has greater than zero lamports.

An attacker could thus block the execution of arbitrary deBridge transfers of which he knows the submission id.

Fix The first bug was quickly fixed by using a create_account_safe wrapper that does a manual combination of transfer , assign and allocate and works in all cases. The second bug was resolved by changing the account-existence-check into an ownership test. This is sufficient since only the program itself can change the owner of its own PDA.

Neodyme verified both fixes.

Broken Fallback Mechanism for External Calls (Medium; Resolved)

Severity	Impact	Affected Component	Status
Medium	Failing External Calls can lead to locked funds	External Calls	Resolved

This issue was already known to the deBridge Team, but the solution wasn't implemented yet when we started the audit.

deBridge allows attaching an External Call to a transaction. This call is then automatically executed by the operator who redeems the transaction. If the user makes an error, or for example, some slippage protection triggers, the external call hardcoded in the deBridge transaction might fail.

In such a case, the user needs a way to get his funds back. The external call mechanism is implemented via cross-program-invocation (CPI). This allows the deBridge program to provide the user-transferred funds to the external call safely.

Unfortunately, there is no way to recover from a failed CPI in Solana. The transaction will always abort. A naive "detect if this call fails" fallback mechanism is thus not possible. The other naive method of allowing the executor to fallback in case of error does not work since an attacker could fail every deBridge transaction.

Fix The solution was to add a new instruction, make_fallback_for_external_call, where a user-specified authority always has the right to cancel an external call, even if it's not executed. This authority is included in the receiver field of the transaction, which is unused in the case of external calls. (since the external call usually gets the funds, not a normal receiver). The receiver is thus now a cancel authority.

Storage Accounts are Missing Discriminator (Low; Resolved)

Severity	Impact	Affected Component	Status
Low	Makes account fakes possible	External Call	Resolved

deBridge allows doing automated external calls. The transactions for these external calls need to be verified. Since they might not fit in a single Solana transaction, all external transactions are uploaded to a storage account, hashed, and verified.

This storage account is not an Anchor account and does not have an account discriminator. This is not an issue per se, but it allows anyone to fake any Anchor account type, since both length and the first few bytes where Anchor stores the discriminator are fully controlled.

Also: Account deletion is scary. But also alright here, since a storage account is always used, created, and deleted together with an Anchor-controlled storage metadata account, the storage account inherits Anchors deletion protections.

Since deBridge always verifies the PDA of accounts, no actual exploitable issue is present. But it does remove a powerful protection that Anchor usually provides, and adding a custom discriminator is an easy hardening.

Fix deBridge agreed and added an account discriminator. Neodyme verified this.

Solana Logs can get Truncated (Info; Info)

The cross-chain communication in deBridge is implemented via Events. Whenever the deBridge contract wants to send something, it emits an event, which the off-chain oracles can then pick up.

Anchor provides the event infrastructure. It base64 encodes the event then logs it into Solana's program log. This can be an issue since Solana's logs are limited in length and can be truncated. Recovering from truncated logs is awkward if not noticed immediately. Not all information contained in the event is available without chain-replay, specifically the nonce. This is because Solana RPC does not provide historical account data. But in such cases, manual intervention can still recover any funds.

The log can grow to a maximum of 10kB, so for most normal uses of deBridge, this truncation will not be an issue. However, there are certain edge cases where an attacker could block an automated transaction from occurring.

Both Solana and Anchor are aware of this issue and are working on a solution:

- solana-labs/solana: Syscall to get the remaining log units
- project-serum/anchor: Events might be dropped because Solana truncates Logs

Neodyme AG

Dirnismaning 55
Halle 13
85748 Garching
E-Mail: contact@neodyme.io

https://neodyme.io