DLN deBridge

HALBORN

DLN - deBridge

Prepared by: HALBORN Last Updated 06/21/2024 Date of Engagement by: February 21st, 2024 - March 18th, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
3	0	0	0	0	3

1. Introduction

DLN is a protocol built on top of deBridge's infrastructure that allows cross-chain messaging between any smart contracts and programs on any chains. The protocol allows the creation of any cross-chain liquidity transfer order, settlement, and management of orders.

The deBridge team engaged Halborn to conduct a security assessment of their Solana programs beginning on 21/2/2024 and ending on 3/18/2024. The security assessment was scoped to the crates provided in the GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

2. Assessment Summary

Halborn was provided 3 weeks for the engagement and assigned one full-time security engineer to review the security of the programs in scope. The engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the programs.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn did not identify any significant security risks. However, some improvements were highlighted that should be addressed, which were acknowledged by the `deBridge team`.

3. Test Approach And Methodology

Halborn performed a combination of manual review and security testing based on scripts to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and programs logic/connectivity/functions along with state changes
- Manual testing by custom scripts.

TABLE OF CONTENTS

4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: dln-solana

(b) Assessed Commit ID: b3a2110

(c) Items in scope:

- dln_dst/src/events.rs
- dln_dst/src/evm_instructions.rs
- dln_dst/src/lib.rs
- dln_dst/src/error.rs
- dln_src/src/lib.rs
- debridge-finance/dln-solana/compare/aafb76acffb5452ea337b0e6c5bf59996535d2ab...b3a2110b9374c4bbc10073aeb2a9b3c8323ae090

Out-of-Scope: Third party dependencies., Economic attacks.

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	0

INFORMATIONAL

3

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
POSSIBLE TO PERFORM VALIDATION IN CONTEXT CONSTRAINTS INSTEAD OF INSTRUCTION HANDLERS	INFORMATIONAL	ACKNOWLEDGED
SINGLE STEP OWNERSHIP TRANSFER PROCESS	INFORMATIONAL	ACKNOWLEDGED
STATE CAN BE REALLOCATED WHEN PROGRAM IS PAUSED	INFORMATIONAL	ACKNOWLEDGED

7. FINDINGS & TECH DETAILS

7.1 POSSIBLE TO PERFORM VALIDATION IN CONTEXT CONSTRAINTS INSTEAD OF INSTRUCTION HANDLERS

// INFORMATIONAL

Description

In multiple functions, such as the `realloc_state` instruction, authority validation is performed in the instruction definition.
`programs/dln_dst/src/lib.rs:158`

```
if state
  .protocol_authority
  .eq(&ctx.accounts.protocol_authority.key())
{
  state.bump = *ctx
    .bumps
    .get("state")
    .expect("Unreachable. State is presented in accounts");
  state.exit(&ID)?;
  Ok(())
 } else {
     Err(error!(ErrorCode::WrongSigner))
 }
```

However, it would adhere to Solana programming model better if these authority checks were performed via the account context struct, simplifying the instruction logic and resulting in cleaner instruction logic.
Using the program logic instead of the account context struct for validation also occurs in the `send_order_cancel` function, as shown below. This should also be handled in the context struct for

clarity and code hygiene.

programs/dln_dst/src/lib.rs:458-467

```
if let Some(ref allowed_cancel_beneficiary) = ctx
  .accounts
  .take_order_state
  .allowed_cancel_beneficiary_src()?
{
  require!(
    allowed_cancel_beneficiary.eq(&cancel_beneficiary),
    ErrorCode::NotAllowedCancelBeneficiary);
}
```

Score

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

Consider leveraging Anchor context constraints for authorization control instead of implementing it in instruction handlers.

Remediation Plan

ACKNOWLEDGED: The **deBridge team** acknowledged this finding.

7.2 SINGLE STEP OWNERSHIP TRANSFER PROCESS

// INFORMATIONAL

Description

The programs `protocol_authority` can be changed via the `set_protocol_authority` function. However, this function simply takes the new authority as input via the account context, and sets this new authority to be the protocol authority. A two-step transfer process is recommended, such that the `protocol_authority` cannot accidentally be set to an unrecoverable/incorrect address that is not under control of the deBridge team.
`programs/dln_dst/src/lib.rs:166-170`
`programs/dln_src/src/lib.rs:350-353`

```
pub fn set_protocol_authority(ctx: Context<UpdateProtocolAuthority>) -> Result<()>
{
  ctx.accounts.update_state.state.protocol_authority =
ctx.accounts.new_protocol_authority.key();
  Ok(())
}
```

Score

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

In the `State` accounts, introduce a new `Pubkey` field that would hold the address of the new owner, which needs to confirm the nomination before privileged access is actually transferred.

Remediation Plan

ACKNOWLEDGED: The **deBridge team** acknowledged this finding.

7.3 STATE CAN BE REALLOCATED WHEN PROGRAM IS PAUSED

// INFORMATIONAL

Description

The `State` account contains a field named `is_working`, which allows the protocol authority to pause the program.

`programs/dln_src/src/lib.rs:142-158`

```
pub struct State {
    pub protocol_authority: Pubkey,
    /// Fixed fee in native chain tokes
    pub fixed_fee: u64,
    /// Fee as bps of process amount
    pub percent_fee_bps: u64,
    /// If true in `claim_order_cancel` we return all fix fee back to maker
    pub is_fee_refund: bool,
    /// Address for transfers fees
    pub fee_beneficiary: Pubkey,
    /// Bump from pubkey of `State` account
    pub bump: u8,
    /// Pubkey for pause program
    pub stop_tap: Pubkey,
    /// Is protocol working right now
    pub is_working: bool,
}
```

However, in the `realloc_state` function, there is no check to ensure this field is not set to `false`.

```
#[derive(Accounts)]
pub struct ReallocState<'info> {
  #[account(
    mut,
    seeds = [State::SEED],
    bump,
  )]
  state: AccountInfo<'info>,
  #[account(mut)]
  protocol_authority: Signer<'info>,
  system_program: Program<'info, System>,
}
```

This is inconsistent with the pausable pattern, which dictates only withdraws should be allowed when a program is paused.

Score

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

Do not allow the protocol authority to realloc the `State` account when the program is paused.

Remediation Plan

ACKNOWLEDGED: The **deBridge team** acknowledged this finding.

8. AUTOMATED TESTING

CARGO AUDIT

ID	CRATE	DESCRIPTION
RUSTSEC-2022-0093	ed25519-dalek	Double Public Key Signing Function Oracle Attack on `ed25519-dalek`
RUSTSEC-2024-0003	h2	Resource exhaustion vulnerability in h2 may lead to Denial of Service (DoS)
RUSTSEC-2024-0019	mio	Tokens for named pipes may be delivered after deregistration
RUSTSEC-2023-0063	quinn-proto	Denial of service in Quinn servers
RUSTSEC-2024-0006	shlex	Multiple issues involving quote API
RUSTSEC-2020-0071	time	Potential segfault in the time crate
RUSTSEC-2023-0001	tokio	reject_remote_clients Configuration corruption
RUSTSEC-2023-0065	tungstenite	Tungstenite allows remote attackers to cause a denial of service
RUSTSEC-2023-0052	webpki	webpki: CPU denial of service in certificate path building

---

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.