deBridge Core Protocol Solana Contracts

deBridge

HALBORN

Report a Bug

deBridge Core Protocol Solana Contracts - deBridge

Prepared by: HALBORN Last Updated 11/14/2024 Date of Engagement by: September 16th, 2024 - October 7th, 2024

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW
1	0	0	0	0

INFORMATIONAL
1

1. Introduction

deBridge team engaged Halborn to conduct a security assessment on their debridge and settings programs beginning on September 16th, 2024 and ending on October 7th, 2024. The security assessment was scoped to the smart contracts provided in the GitHub repositories solana-contracts. Commit hashes, and further details, can be found in the Scope section of this report.

deBridge is releasing a new version of debridge and settings, that includes new implementations for the different instructions necessary to coordinate token orders across different blockchains.

2. Assessment Summary

Halborn was provided 1,5 weeks for the engagement and assigned one full-time security engineer to review the security of the Solana Program in scope. The engineer is a blockchain and smart contract security expert with advanced smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the codebase.
- Check that the codebase does not have any known vulnerability that might affect the participants' funds
- Validate that the implemented changes do not affect the asynchronous nature and cross chain liquidity capabilities of the DLN network.

In summary, Halborn identified one improvement to reduce the likelihood and impact of multiple risks, which has been acknowledged by the deBridge team. It was the following:

- Lack of documentation on new functions

3. Test Approach And Methodology

`Halborn` performed a combination of a manual review of the source code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the program assessment. While manual testing is recommended to uncover flaws in business logic, processes, and implementation; automated testing techniques help enhance coverage of programs and can quickly identify items that do not follow security best practices.

The following phases and associated tools were used throughout the term of the assessment:

- Research into the architecture, purpose, and use of the token airdrop program.
- Manual program source code review to identify business logic issues.
- Mapping out possible attack vectors
- Thorough assessment of safety and usage of critical Rust variables and functions in scope that could lead to arithmetic vulnerabilities.
- Scanning dependencies for known vulnerabilities (`cargo audit`).

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: solana-contracts

(b) Assessed Commit ID: 97c5dbb

(c) Items in scope:

- crates/client/src/client/settings_client.rs
- crates/client/src/settings_instructions_builder.rs
- programs/debridge/Cargo.toml
- programs/debridge/src/external_call_storage.rs
- programs/debridge/src/lib.rs
- programs/settings/Cargo.toml
- programs/settings/src/deploy_info.rs
- programs/settings/src/lib.rs

Out-of-Scope:

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

1

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
LACK OF DOCUMENTATION ON NEW FUNCTIONS	INFORMATIONAL	ACKNOWLEDGED - 11/14/2024

7. FINDINGS & TECH DETAILS

7.1 LACK OF DOCUMENTATION ON NEW FUNCTIONS

// INFORMATIONAL

Description

Several newly added functions lacks an accompanying documentation. Proper documentation is essential in development, as it provides crucial context about the function's purpose, parameters, return values, and expected behavior. Without documentation, developers and maintainers may struggle to understand the function's intent, leading to potential misuses, maintenance challenges, and increased onboarding time for new team members.
The mentioned functions are:

- `update_mint_bridge_token_names` in `crates/client/src/client/settings_client.rs` in line 1335
- `new_update_mint_bridge_token_names` in `crates/client/src/settings_instructions_builder.rs` in line 1049
- `update_mint_bridge_token_names` in `programs/settings/src/lib.rs` in line1659

Which can be seen in the snippets below:

settings_client.rs

```
1333 #[async_trait]
1334 pub trait UpdateMintBridgeTokenNames: SendTransactionBy<Role> {
1335     async fn update_mint_bridge_token_names(
1336         &self,
1337         token_mint: Pubkey,
1338         name: String,
1339         symbol: String,
1340     ) -> Result<Signature, Error> {
```

```
        self.send_by(
            &[Role::ProtocolAuthority],
            &[Instruction::new_update_mint_bridge_token_names(
                self.get_role_pubkey(Role::ProtocolAuthority)
                    .ok_or_else(|| {
                        Error::NotEnoughAuthorization(Role::Protocol
                    })?,
                token_mint,
                name,
                symbol,
            )],
        )
        .await
    }
}
impl<C: SendTransactionBy<Role>> UpdateMintBridgeTokenNames for C {}
```

settings_instructions_builder.rs

```
pub trait NewUpdateMintBridgeTokenNames {
    fn new_update_mint_bridge_token_names(
        protocol_authority: Pubkey,
        token_mint: Pubkey,
        new_name: String,
        new_symbol: String,
    ) -> Instruction {
        Instruction::new_with_bytes(
            settings::ID,
            &settings::instruction::UpdateMintBridgeTokenNames {
                new_name,
                new_symbol,
            }
```

```
            .data(),
            settings::accounts::UpdateMintBridgeTokenMetadata {
                update_bridge: settings::accounts::UpdateBridge {
                    state: *STATE_PUBKEY,
                    protocol_authority,
                    bridge_data: Pubkey::find_bridge_address(&token_
                    token_mint,
                    token_program: token::ID,
                },
                token_metadata: token_metadata::pda::find_metadata_a
                token_metadata_program: token_metadata::ID,
                token_metadata_master: Pubkey::find_token_metadata_m
            }
            .to_account_metas(None),
        )
    }
}
impl NewUpdateMintBridgeTokenNames for Instruction {}
```

lib.rs

```
    pub fn update_mint_bridge_token_names(
        ctx: Context<UpdateMintBridgeTokenMetadata>,
        new_name: String,
        new_symbol: String,
    ) -> Result<()> {
        let token_metadata::state::Metadata {
            data:
                token_metadata::state::Data {
                    uri,
                    seller_fee_basis_points,
```

```
1669                creators,
1670                ..
1671            },
1672        ..
1673    } = token_metadata::state::Metadata::from_account_info(&ctx.
1674
1675    invoke_signed(
1676        &token_metadata::instruction::update_metadata_accounts_v
1677            token_metadata::ID,
1678            ctx.accounts.token_metadata.key(),
1679            ctx.accounts.update_bridge.bridge_data.key(),
1680            None,
1681            Some(TokenMetadata {
1682                name: new_name,
1683                symbol: new_symbol,
1684                seller_fee_basis_points,
1685                creators,
1686                uri,
1687                collection: None,
1688                uses: None,
1689            }),
1690            None,
1691            None,
1692        ),
1693        &[
1694            ctx.accounts.token_metadata.to_account_info(),
1695            ctx.accounts.update_bridge.bridge_data.to_account_ir
1696        ],
1697        &[&[
1698            Bridge::SEED,
1699            ctx.accounts.update_bridge.token_mint.key().as_ref()
1700            &[ctx.accounts.update_bridge.bridge_data.bumps.bridg
1701        ]],
```

TABLE OF CONTENTS

```
1702            )?;
1703            Ok(())
1704        }
```

Documenting code is widely recognized as a best practice in software development, particularly for functions performing complex or business-critical tasks. Clear documentation:
• Enhances readability, allowing developers to quickly understand the function's logic.
• Improves maintainability by reducing time spent deciphering undocumented code.
• Reduces the risk of introducing bugs, as developers can more easily identify the function's intended behavior.
• Simplifies onboarding for new developers by providing clear guidance on existing code functionality.

Score

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

Recommendation

Consider adding inline documentation for each of the mentioned functions, detailing:
1. **Function Purpose**: A brief overview of what the function does.
2. **Parameters**: Descriptions of each parameter, including data types and expected values.
3. **Return Values**: Information on what the function returns and under what conditions.
4. **Error Handling**: Notes on potential errors or edge cases that the function may encounter.

Remediation

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

8. AUTOMATED TESTING

STATIC ANALYSIS REPORT

Description

Halborn used automated security scanners to assist with detection of well-known security issues and vulnerabilities. Among the tools used was `cargo audit`, a security scanner for vulnerabilities reported to the RustSec Advisory Database. All vulnerabilities published in `https://crates.io` are stored in a repository named The RustSec Advisory Database. `cargo audit` is a human-readable version of the advisory database which performs a scanning on Cargo.lock. Security Detections are only in scope. All vulnerabilities shown here were already disclosed in the above report. However, to better assist the developers maintaining this code, the auditors are including the output with the dependencies tree, and this is included in the cargo audit output to better know the dependencies affected by unmaintained and vulnerable crates.

Cargo Audit Results

ID	PACKAGE	SHORT DESCRIPTION
RUSTSEC-2024-0344	curve25519-dalek	Timing variability in `curve25519-dalek`s Scalar29::sub/Scalar52::sub`
RUSTSEC-2022-0093	ed25519-dalek	Double Public Key Signing Function Oracle Attack on `ed25519-dalek`
RUSTSEC-2024-0332	h2	Degradation of service in h2 servers with CONTINUATION Flood
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-2024-0357	openssl	`MemBio::get_buf` has undefined behavior with empty buffers

RUSTSEC-2023-0063	quinn-proto	Denial of service in Quinn servers
RUSTSEC-2024-0336	rustls	`rustls::ConnectionCommon::complete_io` could fall into an infinite loop based on network input
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.