// Private Solana Program Security Assessment 05.10.2024 - 05.24.2024

LayerZero OFT

LayerZero

/ DRAFT /

HALBORN

LayerZero OFT - LayerZero

Prepared by: HALBORN

Last Updated 05/29/2024

Date of Engagement by: May 10th, 2024 - May 24th, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
4	0	0	0	1	3

TABLE OF CONTENTS

1. Introduction

The LayerZero team engaged Halborn to conduct a security assessment on their Solana Validator program beginning on May 10th, 2024 and ending on May 24th, 2024. The security assessment was scoped to the smart contracts provided in the GitHub repository [monorepo](https://github.com/LayerZero-Labs/monorepo/), commit hashes, and further details can be found in the Scope section of this report.

LayerZero is deploying the Omnichain Fungible Token (OFT) version for the Solana Blockchain. This new version is an adaptation of the already deployed OFT smart contracts for Ethereum compatible blockchains.

2. Assessment Summary

Halborn was provided 2 weeks for the engagement and assigned one full-time security engineer to review the security of the Solana Programs 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:

- Validate that the OFT implementation for the Solana blockchain adheres to the specifications outlined in the LayerZero documentation.
- Ensure that the OFT implementation in Solana does not introduce vulnerabilities into the LayerZero ecosystem.
- Analyze the inherent constraints of the Solana blockchain that might impact the security of the OFT program.
- Review the OFT codebase for common vulnerabilities typically found in Solana programs.

In summary, Halborn identified 1 low and 3 informational finding that were accepted and acknowledged by the `LayerZero team`:

1.- Lack of validation of new admin on `transfer_admin` function

2.- Lack of validation of peer address equal to `0x0`

3.- `_credit` and `_debit` helper functions are not implemented as separate functions

4.- Lack of functionality to close a `Peer` account

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.
- Manual testing by custom scripts.
- Automated detection of vulnerabilities in dependencies using `cargo-audit`

4. RISK METHODOLOGY

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the LIKELIHOOD of a security incident and the IMPACT should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

- **10** - CRITICAL
- **9 - 8** - HIGH
- **7 - 6** - MEDIUM
- **5 - 4** - LOW
- **3 - 1** - VERY LOW AND INFORMATIONAL

5. SCOPE

FILES AND REPOSITORY

(a) Repository: monorepo

(b) Assessed Commit ID: 25c8b94

(c) Items in scope:

- monorepo/packages/layerzero-v2/solana/programs/oft/

Out-of-Scope: Integration with other LayerZero programs

REMEDIATION COMMIT ID:

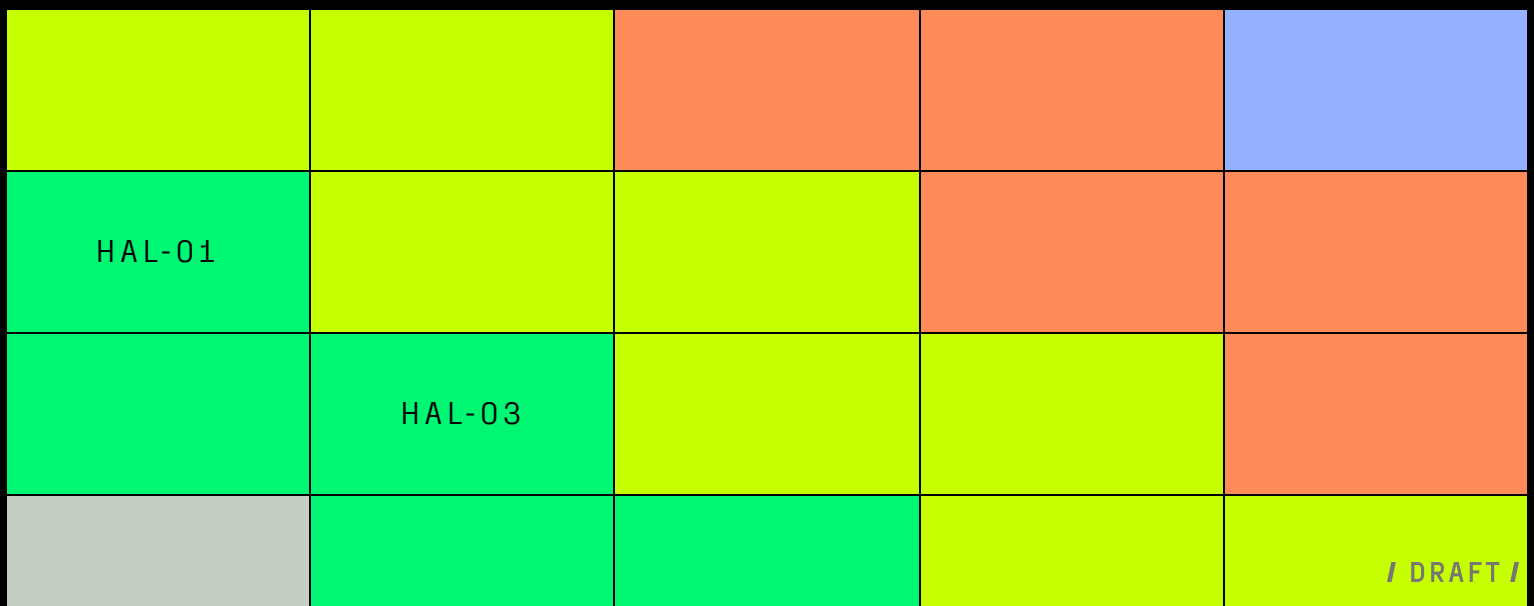
- a2e5ad2a2e5ad2

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	1	3

IMPACT X LIKELIHOOD



HAL-02 HAL-04				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - LACK OF VALIDATION OF NEW OWNER ON TRANSFER_ADMIN FUNCTION	Low	RISK ACCEPTED
HAL-03 - LACK OF VALIDATION OF PEER ADDRESS EQUAL TO 0X0	Informational	ACKNOWLEDGED
HAL-02 - _CREDIT AND _DEBIT ARE NOT IMPLEMENTED AS SEPARATED FUNCTIONS	Informational	ACKNOWLEDGED
HAL-04 - LACK OF FUNCTIONALITY TO CLOSE A PEER ACCOUNT	Informational	ACKNOWLEDGED

7. FINDINGS & TECH DETAILS

7.1 (HAL-01) LACK OF VALIDATION OF NEW OWNER ON TRANSFER_ADMIN FUNCTION

// LOW

Description

The `transfer_admin` function does not validate that the `params.admin` is valid.
This parameter corresponds to the new admin address, which holds critical privileges for managing the OFT program. If the admin is set to an incorrect address, admin privileges would be disabled with no way to revert this, as admin privileges require a signature for execution.
transfer_admin.rs

```
15 impl TransferAdmin<'_> {
16     pub fn apply(ctx: &mut Context<TransferAdmin>, params:
17 &TransferAdminParams) -> Result<()> {
18         ctx.accounts.oft_config.admin = params.admin;
19         Ok(())
20     }
21 }
22
23 #[derive(Clone, AnchorSerialize, AnchorDeserialize)]
24 pub struct TransferAdminParams {
25     pub admin: Pubkey,
   }
```

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:C/D:N/Y:N (2.0)

Recommendation

The recommended way to implement a `transfer_authority` functionality is to use a two-step ownership transfer pattern.
First, implement a function that designates a "new owner candidate." The ownership is not transferred until the new owner "accepts" it by sending a signed transaction.
This method prevents the assignment of a 0x0 owner and ensures that the new owner explicitly accepts the ownership transfer.

Remediation Plan

RISK ACCEPTED: The **LayerZero team** accepted the risk of this finding.

Remediation Hash

a2e5ad2e582f98bb32d27ff774476a051b201cc5

References

https://hackmd.io/@donosonaumczuk/ownership-transfer

7.2 (HAL-03) LACK OF VALIDATION OF PEER ADDRESS EQUAL TO 0X0

// INFORMATIONAL

Description

The `set_peer` function is not checking that the `params.peer` is not `0x0`.
The `set_peer`is an admin managed function, so only the admin of the OFT can update the peers.
A peer is the address of the OFT on a destination chain. If the mentioned address is set to 0 by the OFT admin, the `send` operations will effectively burn the tokens on the destination chain.

```
impl SetPeer<'_> {
    pub fn apply(ctx: &mut Context<SetPeer>, params: &SetPeerParams) ->
Result<()> {
        ctx.accounts.peer.address = params.peer;
        ctx.accounts.peer.bump = ctx.bumps.peer;
        Ok(())
    }
}
```

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:M/I:N/D:H/Y:N (1.8)

Recommendation

Add a validation in the set_peer function to revert the instruction in case that the `params.peer` value is 0.

Remediation Plan

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

Remediation Hash

a2e5ad2e582f98bb32d27ff774476a051b201cc5

References

https://docs.layerzero.network/v2/developers/evm/oft/quickstart#setting-trusted-peers

7.3 (HAL-02) _CREDIT AND _DEBIT ARE NOT IMPLEMENTED AS SEPARATED FUNCTIONS

// INFORMATIONAL

Description

According to the LayerZero documentation, in order to keep track of the supplies in all the different chains that an OFT is deployed, is necessary to increase/decrease the volume of tokens on each chain to avoid minting tokens unexpectedly.
To do that, both the `lz_receive` and `send` functions should implement a helper function called `_credit` and `_debit` correspondingly, which sends or burns tokens depending on the function called.
The current implementation does not implement the mentioned functions as isolated helpers.
Using isolated helpers not only will comply with the documentation specification, but it will help with future reviews.

lz_receive.rs

```
91          let amount_received_ld = match &ctx.accounts.oft_config.ext {
92              OftConfigExt::Adapter { token_escrow: escrow } => {
93                  if let Some(escrow_acc) = &ctx.accounts.token_escrow {
94                      let balance_before =
95  ctx.accounts.token_dest.amount;
96                      // unlock
97                      let seeds: &[&[u8]] =
98                          &[OFT_SEED, &escrow.to_bytes(), &
99  [ctx.accounts.oft_config.bump]];
100                     token_interface::transfer_checked(
101                         CpiContext::new(
102
103 ctx.accounts.token_program.to_account_info(),
104                             TransferChecked {
105                                 from: escrow_acc.to_account_info(),
106                                 mint:
107 ctx.accounts.token_mint.to_account_info(),
108                                 to:
109 ctx.accounts.token_dest.to_account_info(),
110                                 authority:
111 ctx.accounts.oft_config.to_account_info(),
112                             },
113                         )
114                         .with_signer(&[&seeds]),
115                         amount_ld,
116                         ctx.accounts.token_mint.decimals,
117
```

```
                    )?;

                    ctx.accounts.token_dest.reload()?;
                    ctx.accounts.token_dest.amount - balance_before
                } else {
                    return Err(OftError::InvalidTokenEscrow.into());
                }
            },
            OftConfigExt::Native { mint_authority: _ } => {
                // mint
                let cpi_accounts = MintTo {
                    mint: ctx.accounts.token_mint.to_account_info(),
                    to: ctx.accounts.token_dest.to_account_info(),
                    authority:
ctx.accounts.oft_config.to_account_info(),
                };
                let cpi_program =
ctx.accounts.token_program.to_account_info();
                let cpi_context = CpiContext::new(cpi_program,
cpi_accounts);
                token_interface::mint_to(cpi_context.with_signer(&
[&seeds]), amount_ld)?;
                amount_ld
            },
        };
```

send.rs

```
        let amount_received_ld = match &ctx.accounts.oft_config.ext {
            OftConfigExt::Adapter { token_escrow: _ } => {
                if let Some(escrow_acc) = &mut
ctx.accounts.token_escrow {
                    // lock
                    let initial_escrow_balance = escrow_acc.amount;
                    token_interface::transfer_checked(
                        CpiContext::new(

ctx.accounts.token_program.to_account_info(),
                            TransferChecked {
                                from:
ctx.accounts.token_source.to_account_info(),
                                mint:

```

```
ctx.accounts.token_mint.to_account_info(),
                                to: escrow_acc.to_account_info(),
                                authority:
ctx.accounts.signer.to_account_info(),
                            },
                        ),
                        amount_sent_ld,
                        ctx.accounts.token_mint.decimals,
                    )?;
                    escrow_acc.reload()?;
                    escrow_acc.amount - initial_escrow_balance // ==
expected_received_ld
                } else {
                    return Err(OftError::InvalidTokenEscrow.into());
                }
            },
            OftConfigExt::Native { mint_authority: _ } => {
                // burn
                let cpi_accounts = Burn {
                    mint: ctx.accounts.token_mint.to_account_info(),
                    from: ctx.accounts.token_source.to_account_info(),
                    authority: ctx.accounts.signer.to_account_info(),
                };
                let cpi_program =
ctx.accounts.token_program.to_account_info();
                let cpi_context = CpiContext::new(cpi_program,
cpi_accounts);
                let seeds: &[&[u8]] = &[
                    OFT_SEED,
                    &ctx.accounts.oft_config.token_mint.to_bytes(),
                    &[ctx.accounts.oft_config.bump],
                ];
                token_interface::burn(cpi_context.with_signer(&
[&seeds]), amount_sent_ld)?;
                amount_sent_ld
            },
        };
```

Score

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

Recommendation

Implement the mentioned helper functions as separated functions to comply with the specification present in the documentation.

Remediation Plan

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

Remediation Hash

a2e5ad2e582f98bb32d27ff774476a051b201cc5

References

https://docs.layerzero.network/v2/developers/evm/oft/quickstart

7.4 (HAL-04) LACK OF FUNCTIONALITY TO CLOSE A PEER ACCOUNT

// INFORMATIONAL

Description

The OFT program is not currently implementing a function to close a `Peer` account.
A `Peer` is the address of the OFT on a destination chain. It corresponds to a PDA derived from a string, the oft config, and a `dst_eid`. The `dst_eid` represents the destination id or the endpoint id where the recipient OFT is deployed. If, for some reason, the OFT creator wants to close the communication with the network corresponding to the `dst_eid`, the creator has the alternative to set the corresponding pathway `messagelib` to blocked, which effectively will cut the connection between both chains.
The mentioned solution is not ideal and it can bring unexpected consequences,

lib.rs

```
25  #[program]
26  pub mod oft {
27      use super::*;
28
29      pub fn version(_ctx: Context<GetVersion>) -> Result<Version> {
30          Ok(Version { sdk_version: OFT_SDK_VERSION, oft_version:
31  OFT_VERSION })
32      }
33
34      pub fn init_oft(mut ctx: Context<InitOft>, params: InitOftParams) -
35  > Result<()> {
36          InitOft::apply(&mut ctx, &params)
37      }
38
39      pub fn init_adapter_oft(
40          mut ctx: Context<InitAdapterOft>,
41          params: InitAdapterOftParams,
42      ) -> Result<()> {
43          InitAdapterOft::apply(&mut ctx, &params)
44      }
45
46      // ============================== Admin
47  ==============================
48      pub fn transfer_admin(
49          mut ctx: Context<TransferAdmin>,
50          params: TransferAdminParams,
51      ) -> Result<()> {
52
```

```
        TransferAdmin::apply(&mut ctx, &params)
    }

    pub fn set_peer(mut ctx: Context<SetPeer>, params: SetPeerParams) -
> Result<()> {
        SetPeer::apply(&mut ctx, &params)
    }

    pub fn set_enforced_options(
        mut ctx: Context<SetEnforcedOptions>,
        params: SetEnforcedOptionsParams,
    ) -> Result<()> {
        SetEnforcedOptions::apply(&mut ctx, &params)
    }

    pub fn set_mint_authority(
        mut ctx: Context<SetMintAuthority>,
        params: SetMintAuthorityParams,
    ) -> Result<()> {
        SetMintAuthority::apply(&mut ctx, &params)
    }

    pub fn mint_to(mut ctx: Context<MintTo>, params: MintToParams) ->
Result<()> {
        MintTo::apply(&mut ctx, &params)
    }

    // ============================== Public
==============================

    pub fn quote_oft(
        ctx: Context<QuoteOft>,
        params: QuoteOftParams,
    ) -> Result<(OFTLimits, Vec<OFTFeeDetail>, OFTReceipt)> {
        QuoteOft::apply(&ctx, &params)
    }

    pub fn quote(ctx: Context<Quote>, params: QuoteParams) ->
Result<MessagingFee> {
        Quote::apply(&ctx, &params)
    }

    pub fn send(mut ctx: Context<Send>, params: SendParams) ->
Result<MessagingReceipt> {
```

```
        Send::apply(&mut ctx, &params)
    }

    pub fn lz_receive(mut ctx: Context<LzReceive>, params:
LzReceiveParams) -> Result<()> {
        LzReceive::apply(&mut ctx, &params)
    }

    pub fn lz_receive_types(
        ctx: Context<LzReceiveTypes>,
        params: LzReceiveParams,
    ) -> Result<Vec<oapp::endpoint_cpi::LzAccount>> {
        LzReceiveTypes::apply(&ctx, &params)
    }

    pub fn set_rate_limit(
        mut ctx: Context<SetRateLimit>,
        params: SetRateLimitParams,
    ) -> Result<()> {
        SetRateLimit::apply(&mut ctx, &params)
    }

    // Set the LayerZero endpoint delegate for OApp admin functions
    pub fn set_delegate(mut ctx: Context<SetDelegate>, params:
SetDelegateParams) -> Result<()> {
        SetDelegate::apply(&mut ctx, &params)
    }
}
```

Score

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

Recommendation

Implement a permissioned function, only callable by the OFT admin, to close the `Peer` account.

Remediation Plan

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

Remediation Hash

a2e5ad2e582f98bb32d27ff774476a051b201cc5

8. AUTOMATED TESTING

Cargo Audit

Description

Halborn used automated security scanners to assist with the 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 reviewers 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.

Results

ID	PACKAGE	SHORT DESCRIPTION
RUSTSEC-2022-0093	ed25519-dalek	Double Public Key Signing Function Oracle Attack on `ed25519-dalek`

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.