sec3™

DRAFT

FOR REVIEW ONLY

Security Assessment Report

Hyperlane Sealevel Programs

July 15th, 2023

Summary

The sec3 team (formerly Soteria) was engaged to do a thorough security analysis of the Hyperlane Sealevel Solana smart contract programs. The artifact of the audit was the source code of the following smart contracts excluding tests in https://github.com/hyperlane-xyz/hyperlane-monorepo/tree/78a5aea/rust/sealevel

- rust/sealevel/programs/mailbox
- rust/sealevel/programs/ism
- rust/sealevel/libraries/ecdsa-signature
- rust/sealevel/libraries/multisig-ism
- rust/sealevel/libraries/hyperlane-sealevel-token
- rust/sealevel/programs/hyperlane-sealevel-token
- rust/sealevel/programs/hyperlane-sealevel-token-collateral
- rust/sealevel/programs/hyperlane-sealevel-token-native
- rust/sealevel/programs/validator-announce
- rust/sealevel/libraries/access-control
- rust/sealevel/libraries/account-utils
- rust/sealevel/libraries/hyperlane-sealevel-connection-client
- rust/sealevel/libraries/interchain-security-module-interface
- rust/sealevel/libraries/message-recipient-interface
- rust/sealevel/libraries/serializable-account-meta

The initial audit was done on commit 78a5aea7181696a62ac412d5686c8253f0b5cf9a of the following smart contracts and shared utilities.

The audit revealed 8 issues or questions. This report describes the findings and resolutions in detail.

Table of Contents

Result Overview

Issue	Impact	Status
[L-1] Encode data in all set_return_data()	Low	Open
[L-1] Enable runtime overflow checks	Low	Open
[I-1] Duplicated signatures allowed	Informational	Open
[I-2] Signature malleability	Informational	Open
[I-3] Inconsistent comments	Informational	Open
[I-4] Create a 0 sized account owned by the system_program	Informational	Open
[I-5] TODO cleanup	Informational	Open
[I-6] Make the verify in multisig-ism lib more self-contained	Informational	Open

Findings in Detail

[L -1]   Encode data in all set_return_data()

The data in the following `set_return_data` may have trailing zeros. It's recommended to encode them too.

```
/* sealevel/programs/mailbox/src/processor.rs */
set_return_data(id.as_ref());

/* sealevel/programs/mailbox/src/processor.rs */
set_return_data(&count.to_le_bytes());

/* sealevel/programs/mailbox/src/processor.rs */
set_return_data(&ret_buf);

/* sealevel/programs/mailbox/src/processor.rs */
set_return_data(root.as_ref());

/* sealevel/programs/mailbox/src/processor.rs */
set_return_data(
    &outbox
        .owner
        .try_to_vec()
        .map_err(|err| ProgramError::BorshIoError(err.to_string()))?,
);

/* sealevel/programs/ism/multisig-ism-message-id/src/processor.rs */
set_return_data(
    &access_control_data
        .owner
        .try_to_vec()
        .map_err(|err| ProgramError::BorshIoError(err.to_string()))?,
);

/* sealevel/libraries/hyperlane-sealevel-connection-client/src/lib.rs */
fn set_interchain_security_module_return_data(&self) {
    let ism: Option<Pubkey> = self.interchain_security_module().cloned();
    set_return_data(
        &ism.try_to_vec()
            .map_err(|err| ProgramError::BorshIoError(err.to_string()))
```

```
                .unwrap()[..],
    );
}

/* sealevel/programs/ism/multisig-ism-message-id/src/processor.rs */
fn get_validators_and_threshold(
) -> ProgramResult {
    set_return_data(
        &validators_and_threshold
            .try_to_vec()
            .map_err(|err| ProgramError::BorshIoError(err.to_string()))?,
    );
}

/* sealevel/programs/ism/multisig-ism-message-id/src/processor.rs */
fn get_owner(program_id: &Pubkey, accounts: &[AccountInfo]) -> ProgramResult {
    set_return_data(
        &access_control_data
            .owner
            .try_to_vec()
            .map_err(|err| ProgramError::BorshIoError(err.to_string()))?,
    );
}
```

[L-2] Enable runtime overflow checks

The addition at line 363 may overflow.

```
/* sealevel/programs/mailbox/src/processor.rs */
176 | fn inbox_process(
180 | ) -> ProgramResult {
363 |     inbox.processed_count += 1;
```

Consider enabling the runtime overflow check and adding the following in `Cargo.toml`

```
[profile.release]
overflow-checks = true
```

[I - 1] Duplicated signatures allowed

When loading signatures from caller-controlled arguments, it doesn't check if there are duplicated signatures.

```
/* sealevel/programs/ism/multisig-ism-message-id/src/metadata.rs */
impl TryFrom<Vec<u8>> for MultisigIsmMessageIdMetadata {
    fn try_from(bytes: Vec<u8>) -> Result<Self, Self::Error> {
        let signature_count = signature_bytes_len / SIGNATURE_LENGTH;
        let mut validator_signatures = Vec::with_capacity(signature_count);
        for i in 0..signature_count {
            let signature_offset = SIGNATURES_OFFSET + (i * SIGNATURE_LENGTH);
            let signature = EcdsaSignature::from_bytes(
                &bytes[signature_offset..signature_offset + SIGNATURE_LENGTH],
            )
            .map_err(|_| Error::InvalidMetadata)?;
            validator_signatures.push(signature);
        }
        Ok(Self {
            validator_signatures,
        })
    }
}
```

However, the signature quorum check is still safe, since validator_index moves once there is a hit, and there are no duplicated validators due to the check at processor.rs:372.

It's still a good idea to reject duplicated signatures.

```
/* sealevel/libraries/multisig-ism/src/multisig.rs */
pub fn verify(&self) -> Result<(), MultisigIsmError> {
    let signed_digest = self.signed_data.eth_signed_message_hash();
    let signed_digest_bytes = signed_digest.as_bytes();
    let validator_count = self.validators.len();
    let mut validator_index = 0;
    // Assumes that signatures are ordered by validator
    for i in 0..self.threshold {
        let signer = self.signatures[i as usize]
            .secp256k1_recover_ethereum_address(signed_digest_bytes)
            .map_err(|_| MultisigIsmError::InvalidSignature)?;
        while validator_index < validator_count && signer != self.validators[validator_index] {
            validator_index += 1;
        }
        if validator_index >= validator_count {
            return Err(MultisigIsmError::ThresholdNotMet);
        }
        validator_index += 1;
    }
    Ok(())
}

/* sealevel/programs/ism/multisig-ism-message-id/src/processor.rs */
fn set_validators_and_threshold(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
    config: Domained<ValidatorsAndThreshold>,
) -> ProgramResult {
    // Validate the provided validators and threshold.
    config.data.validate()?;
```

[I -2] Signature malleability

The solana secp256k1_recover function does not prevent signature malleability. This is in contrast to the Bitcoin secp256k1 library, which does prevent malleability by default. Solana accepts signatures with S values that are either in the high order or in the low order, and it is trivial to produce one from the other.

Reference: https://docs.rs/sol-chainsaw/

However, for the same reason mentioned in [I-1] (the validator_index moves once a hit is found), it's not possible to take advantage of the signature malleability to break the check.

Consider rejecting signatures with high-order S values to prevent malleability.

```
/* sealevel/programs/validator-announce/src/processor.rs */
fn verify_validator_signed_announcement(
    announce: &AnnounceInstruction,
    validator_announce: &ValidatorAnnounce,
) -> Result<(), ProgramError> {
    let announcement = Announcement {
        validator: announce.validator,
        mailbox_address: validator_announce.mailbox.to_bytes().into(),
        mailbox_domain: validator_announce.local_domain,
        storage_location: announce.storage_location.clone(),
    };
    let announcement_digest = announcement.eth_signed_message_hash();
    let signature = EcdsaSignature::from_bytes(&announce.signature[..])
        .map_err(|_| ProgramError::from(Error::SignatureError))?;

    let recovered_signer = signature
        .secp256k1_recover_ethereum_address(&announcement_digest[..])
        .map_err(|_| ProgramError::from(Error::SignatureError))?;

    if recovered_signer != announcement.validator {
        return Err(ProgramError::InvalidAccountData);
    }

    Ok(())
}
```

[I -3] Inconsistent comments

At processor.rs:173, N+2..M. should be N+3...M.

At plugin.rs:108 and plugin.rs:217, it's a token transfer instead of burning the tokens.

```
/* sealevel/programs/mailbox/src/processor.rs */
172 | // N+2.    [executable] ISM
173 | // N+2..M. [??] Accounts required to invoke the ISM's Verify instruction.
176 | fn inbox_process(

/* sealevel/programs/hyperlane-sealevel-token-native/src/plugin.rs */
107 | /// Transfers tokens into the program so they can be sent to a remote chain.
108 | /// Burns the tokens from the sender's associated token account.
113 | fn transfer_in<'a, 'b>(

/* sealevel/programs/hyperlane-sealevel-token-collateral/src/plugin.rs */
216 | /// Transfers tokens to the escrow account so they can be sent to a remote chain.
217 | /// Burns the tokens from the sender's associated token account.
224 | fn transfer_in<'a, 'b>(
```

[I -4] Create a 0 sized account owned by the system_program

An account owned by the system program with 0 space is confusing. Potentially, it cannot prevent the account creation being called again so that this contract may be initialized several times, which is not the intention of the initialization process.

Although it doesn't seem to have side effects for this initializer, consider allocating more space instead.

```
/* sealevel/programs/hyperlane-sealevel-token-native/src/plugin.rs */
fn initialize<'a, 'b>(
    program_id: &Pubkey,
    system_program: &'a AccountInfo<'b>,
    _token_account: &'a AccountInfo<'b>,
    payer_account: &'a AccountInfo<'b>,
    accounts_iter: &mut std::slice::Iter<'a, AccountInfo<'b>>,
) -> Result<Self, ProgramError> {
    // Account 0: Native collateral PDA account.
    let native_collateral_account = next_account_info(accounts_iter)?;
    let (native_collateral_key, native_collateral_bump) = Pubkey::find_program_address(
        hyperlane_token_native_collateral_pda_seeds!(),
        program_id,
    );
    if &native_collateral_key != native_collateral_account.key {
        return Err(ProgramError::InvalidArgument);
    }

    // Create native collateral PDA account.
    // Assign ownership to the system program so it can transfer tokens.
    create_pda_account(
        payer_account,
        &Rent::get()?,
        0,
        &solana_program::system_program::id(),
        system_program,
        native_collateral_account,
        hyperlane_token_native_collateral_pda_seeds!(native_collateral_bump),
    )?;
}
```

[I-5] TODO cleanup

```
/* sealevel/libraries/hyperlane-sealevel-token/src/processor.rs */
let message = TokenMessage::read_from(&mut message_reader)
    .map_err(|_err| ProgramError::from(Error::TODO))?;
let message = TokenMessage::read_from(&mut message_reader)
    .map_err(|_err| ProgramError::from(Error::TODO))?;
```

[I -6] Make the verify in multisig-ism lib more self-contained

```
/* sealevel/libraries/multisig-ism/src/multisig.rs */
pub fn verify(&self) -> Result<(), MultisigIsmError> {
    let signed_digest = self.signed_data.eth_signed_message_hash();
    let signed_digest_bytes = signed_digest.as_bytes();

    let validator_count = self.validators.len();
    let mut validator_index = 0;

    // Assumes that signatures are ordered by validator
    for i in 0..self.threshold {
        let signer = self.signatures[i as usize]
            .secp256k1_recover_ethereum_address(signed_digest_bytes)
            .map_err(|_| MultisigIsmError::InvalidSignature)?;

        while validator_index < validator_count && signer != self.validators[validator_index] {
            validator_index += 1;
        }

        if validator_index >= validator_count {
            return Err(MultisigIsmError::ThresholdNotMet);
        }

        validator_index += 1;
    }

    Ok(())
}
```

The correctness of this code assumes (1) the threshold <= validator_count and (2) there is no duplications in the validators.

These conditions are currently met because this function is only invoked by multisig-ism-message-id and the threshold and validators are loaded from a PDA owned by multisig-ism-message-id. When setting the validators and threshold, the contract does the validations.

```
/* sealevel/programs/ism/multisig-ism-message-id/src/processor.rs */
fn verify(
) -> ProgramResult {
    let validators_and_threshold = validators_and_threshold(program_id, accounts, message.origin)?;
    multisig_ism
        .verify()
        .map_err(|err| Into::<Error>::into(err).into())
}
```

However, as an independent module, it may be a good idea to add the checks and make it self-contained.

Appendix: Methodology and Scope of Work

The sec3 (formerly Soteria) audit team, which consists of Computer Science professors and industrial researchers with extensive experience in Solana smart contract security, program analysis, testing and formal verification, performed a comprehensive manual code review, software static analysis and penetration testing.

Assisted by the sec3 Scanner developed in-house, the audit team particularly focused on the following work items:

- Check common security issues.
  - Missing ownership checks
  - Missing signer checks
  - Signed invocation of unverified programs
  - Solana account confusions
  - Arithmetic over- or underflows
  - Numerical precision errors
  - Loss of precision in calculation
  - Insufficient SPL-Token account verification
  - Missing rent exemption assertion
  - Casting truncation
  - Did not follow security best practices
  - Outdated dependencies
  - Redundant code
  - Unsafe Rust code
- Check program logic implementation against available design specifications.
- Check poor coding practices and unsafe behavior.
- The soundness of the economics design and algorithm is out of scope of this work

DISCLAIMER

The instance report ("Report") was prepared pursuant to an agreement between Coderrect Inc. d/b/a sec3 (the "Company") and Abacus Works, Inc (the "Client"). This Report solely includes the results of a technical assessment of a specific build and/or version of the Client's code specified in the Report ("Assessed Code") by the Company. The sole purpose of the Report is to provide the Client with the results of the technical assessment of the Assessed Code. The Report does not apply to any other version and/or build of the Assessed Code. Regardless of the contents of the Report, the Report does not (and should not be interpreted to) provide any warranty, representation or covenant that the Assessed Code: (i) is error and/or bug free, (ii) has no security vulnerabilities, and/or (iii) does not infringe any third-party rights. Moreover, the Report is not, and should not be considered, an endorsement by the Company of the Assessed Code and/or of the Client. Finally, the Report should not be considered investment advice or a recommendation to invest in the Assessed Code and/or the Client.

This Report is considered null and void if the Report (or any portion thereof) is altered in any manner.

ABOUT

Founded by leading academics in the field of software security and senior industrial veterans, sec3 (formerly Soteria) is a leading blockchain security company. We are also building sophisticated security tools that incorporate static analysis, penetration testing, and formal verification.

At sec3, we identify and eliminate security vulnerabilities through the most rigorous process and aided by the most advanced analysis tools.

For more information, check out our website and follow us on twitter.

sec3™