Zellic

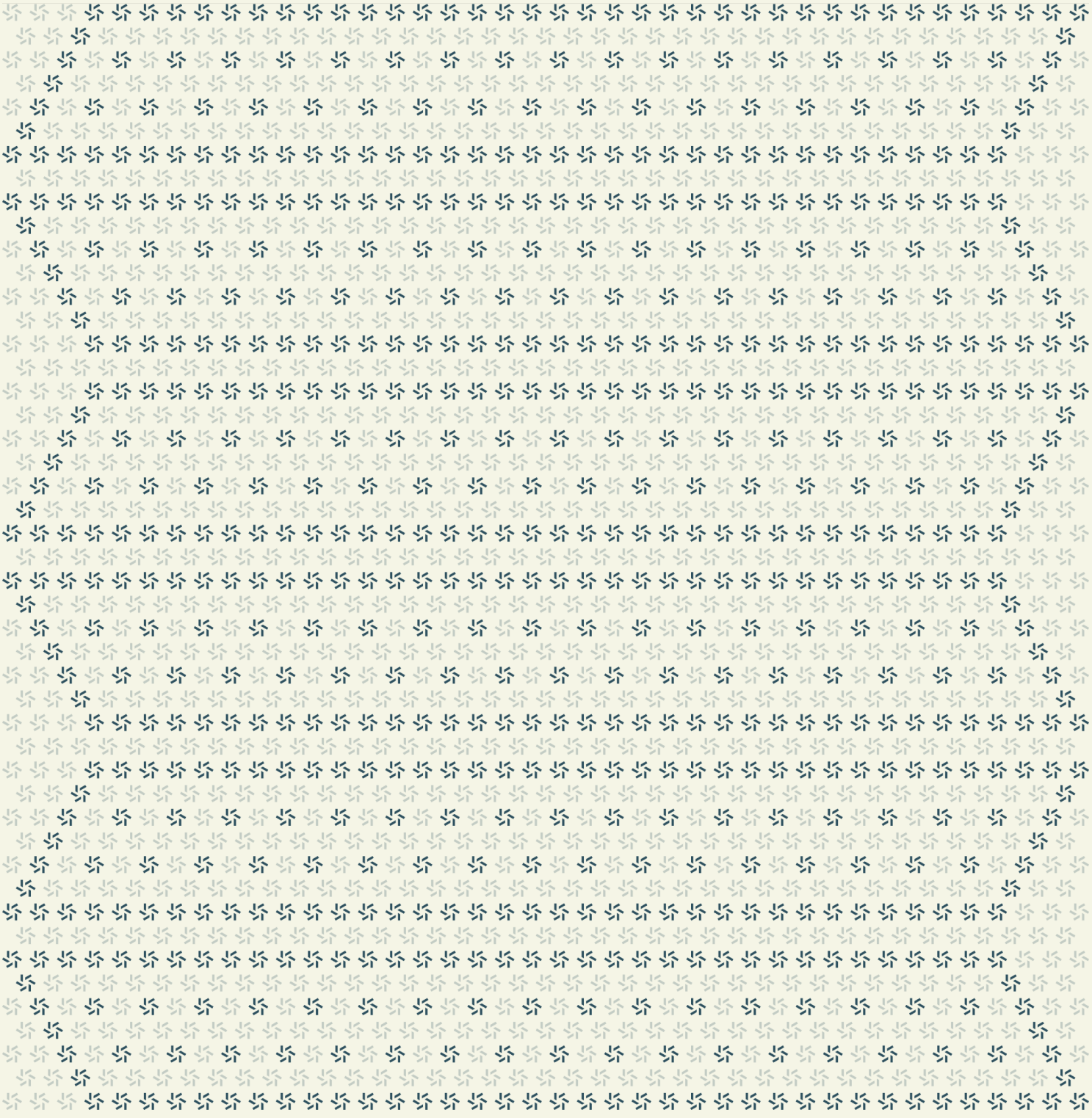
Prepared for
Albert Limos
Chainflip

Prepared by
Frank Bachman
Junyi Wang
Zellic

August 13, 2024

Chainflip

Smart Contract Security Assessment



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Chainflip from July 29th to August 1st. During this engagement, Zellic reviewed Chainflip's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Are there any bugs that can lead to the loss of funds in the vault?
- Is there a lack of function access control in either programs?
- Is it possible for a malicious party to be able to trigger a swap event on the programs without actually having transferred the correct asset?
- Are there any bugs on the cross-chain messaging logic (e.g., `execute_ccm_*_swap` calling a user's program with `cf_receive_*`) that would allow the receiver of a CCM swap to execute malicious logic and steal funds from the vault?
- Is there a way to put either programs in a state that results in a DOS?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

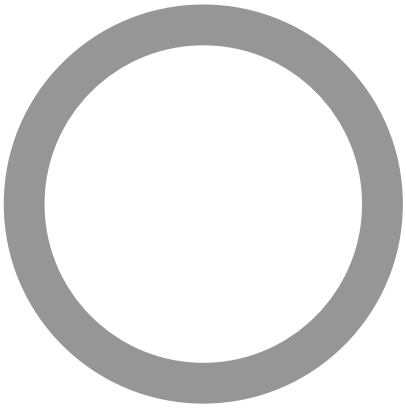
- Front-end components
- Infrastructure relating to the project
- Key custody

1.4. Results

During our assessment on the scoped Chainflip contracts, we discovered three findings, all of which were informational in nature.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	0
Informational	3



2. Introduction

2.1. About Chainflip

Chainflip contributed the following description of Chainflip:

> Chainflip is a cross-chain asset exchange protocol. At its core are two major innovations:
>
> 1. Fully distributed and permissionless 100-of-150 multi-signature vaults using the FROST signing protocol.
> 2. A novel and highly capital-efficient JIT (Just-In-Time) AMM.
>
> This allows transfer and exchange of assets across chains without the need for wrapped tokens or trusted intermediaries.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We

also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

2.3. Scope

The engagement involved a review of the following targets:

Chainflip Contracts

Type	Rust
Platform	Solana
Target	chainflip-sol-contracts
Repository	https://github.com/chainflip-io/chainflip-sol-contracts ↗
Version	`cea567305393b70730e9ca0011aebd3a096705a5`
Programs	`vault` `swap-endpoint`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 1.1 person-weeks. The assessment was conducted by two consultants over the course of four calendar days.

Contact Information

The following project manager was associated with the engagement:

Chad McDonald
Engagement Manager
chad@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Frank Bachman
Engineer
frank@zellic.io ↗

Junyi Wang
Engineer
junyi@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

July 30, 2024	Kick-off call
July 29, 2024	Start of primary review period
August 1, 2024	End of primary review period

3. Detailed Findings

3.1. Size calculation for program accounts

Target	vault		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The vault program uses the Anchor framework, which uses Borsh serialization for program accounts. It was found that `core::mem::size_of` was used to compute the size to be reserved for multiple program accounts.

```
#[account(
    init,
    seeds = [&DATA_ACCOUNT_SEED],
    bump,
    payer = initializer,
    space = size_of::<DataAccount>() + DISCRIMINATOR_SIZE,
)]
pub data_account: Account<'info, DataAccount>
```

This is currently not an issue because the types being used for the current program structures just happen to have the same sizes with Borsh and native Rust types. However, there are types for which Borsh sizing varies.

Impact

The computed size could be smaller if changes are made to any of these structures in the future.

Recommendations

It is recommended to use the Anchor `InitSpace` macro to compute the sizes instead.

Remediation

This issue has been acknowledged by Chainflip, and a fix was implemented in commit `1cb51482` ↗.

3.2. Potential seed collision

Target	vault, swap-endpoint		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

It was found that PDA accounts in both vault and swap-endpoint programs do not use constant prefixes with pubkeys to derive the PDA addresses.

```
#[account(
    init_if_needed,
    seeds = [deposit_channel_associated_token_account.key().as_ref()],
    bump,
    payer = agg_key,
    space = size_of::<DepositChannelHistoricalFetch>() + DISCRIMINATOR_SIZE,
)]
```

Impact

This might result in seed collisions in case of any future updates to the code.

Recommendations

It is recommended to use constant prefixes with pubkeys to derive the PDA addresses.

Remediation

This issue has been acknowledged by Chainflip, and a fix was implemented in commit 5c784600 ↗.

3.3. The `CloseEventAccounts` event emitted without validating event account

Target	swap-endpoint		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

In the `CloseEventAccounts` instruction, the `swap-endpoint` program seems to assume reorgs for the deserialization failure of the `SwapEvent` account.

```
            if let Ok(event_account)
    = Account::<SwapEvent>::try_from(event_account_info) {
[...]
            } else {
                // Ideally this should never happen but it could happen if the
    state chain
                // were to go out of sync (e.g. large reorg, witnessing
    missbehaviour...)
                emit!(events::CantDeserializeEventAccount {
                    event_account: event_account_info.key(),
                    payee: payee.key(),
                });
```

However, it only checks the `SwapEvent` account's PubKey after the deserialization is successful.

Impact

This has no security impact since the event emitted is likely just used for debugging.

Recommendations

The `SwapEvent` account's PubKey should be validated before trying to deserialize the account.

Remediation

This issue has been acknowledged by Chainflip, and a fix was implemented in commit `019c6b94` ↗.

4. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

4.1. Module: swap-endpoint

Function: `initialize`

Simply initializes `SwapEndpointDataAccount` to default values.

Accounts

- `swap_endpoint_data_account`
  - **Validation**: Account being initialized — no validation required.
  - **Impact**: N/A.
- `signer`
  - **Validation**: Account is signer.
  - **Impact**: N/A.

Instruction parameters

N/A.

Function: `x_swap_native`

This function does the following:

- Checks that the number of open event accounts is larger than the maximum-allowed event accounts
- Adds the event-account pubkey to the list of open event accounts
- Transfers lamports from the signer to the aggregate key account
- Updates the event-data account with the params

Accounts

- `data_account`
  - **Validation**: Account owner and discriminator are checked. Checked if swaps are suspended.

- **Impact**: N/A.

- `agg_key`
  - **Validation**: Key validated through `data_account`.
  - **Impact**: Lamports from the native swap are sent to this account.
- `from`
  - **Validation**: Account is a signer.
  - **Impact**: Lamports are sent from this account.
- `event_data_account`
  - **Validation**: Newly initialized account — no validation required.
  - **Impact**: Stores data for the swap.
- `swap_endpoint_data_account`
  - **Validation**: Account owner and discriminator are checked.
  - **Impact**: Stores the pubkey for the event-data account.

Instruction parameters

- `swap_native_params`
  - The `amount` and `decimals` values in the structure are used to transfer lamports from the signer.
  - The amount is checked to be larger than the minimum native swap amount.
  - The remaining values are stored in the event-data account, and no validation is performed.

Function: `x_swap_token`

This function does the following:

- Checks that the number of open event accounts is larger than the maximum-allowed event accounts
- Adds the event-account pubkey to the list of open event accounts
- Transfers lamports from the signer to the aggregate key account
- Updates the event-data account with the params

Accounts

- `data_account`
  - **Validation**: Account owner and discriminator are checked. Checked if swaps are suspended.
  - **Impact**: N/A.
- `agg_key`
  - **Validation**: Key validated through `data_account`.
  - **Impact**: Lamports from the native swap are sent to this account.

- `from`
  - **Validation**: Account is a signer.
  - **Impact**: Lamports are sent from this account.
- `event_data_account`
  - **Validation**: Newly initialized account — no validation required.
  - **Impact**: Stores data for the swap.
- `swap_endpoint_data_account`
  - **Validation**: Account owner and discriminator are checked.
  - **Impact**: Stores the pubkey for the event-data account.

Instruction parameters

- `x_swap_token`
  - The `amount` and `decimals` values in the structure are used to transfer lamports from the signer.
  - The amount is checked to be larger than the minimum native swap amount.
  - The remaining values are stored in the event-data account, and no validation is performed.

Function: `x_swap_token`

This function does the following:

- Checks the mint from the `SupportedToken` account
- Checks that the number of open event accounts is larger than the maximum-allowed event accounts
- Adds the event-account pubkey to the list of open event accounts
- Transfers tokens from the user to the token vault
- Updates the event-data account with the params

Accounts

- `data_account`
  - **Validation**: Account owner and discriminator are checked. Checked if swaps are suspended.
  - **Impact**: N/A.
- `token_vault_associated_token_account`
  - **Validation**: Checked to be the correct ATA through the mint and authority from the data account.
  - **Impact**: Tokens to swap are transferred to this token account.
- `from`
  - **Validation**: Account is a signer.

    - **Impact**: Signer for the token account.
- `from_token_account`
    - **Validation**: Account checked to be owned by the signer.
    - **Impact**: Tokens to be swapped are sent from this token account.
- `event_data_account`
    - **Validation**: Newly initialized account — no validation required.
    - **Impact**: Stores data for the swap.
- `swap_endpoint_data_account`
    - **Validation**: Account owner and discriminator are checked.
    - **Impact**: Stores the pubkey for the event-data account.
- `token_supported_account`
    - **Validation**: Account owner and discriminator are checked.
    - **Impact**: Used to check if the token swapped from is supported.
- `mint`
    - **Validation**: Mint is checked within the instruction logic.
    - **Impact**: N/A.

Instruction parameters

- `swap_native_params`
    - The `amount` and `decimals` values in the structure are used to transfer lamports from the signer.
    - The amount is checked to be larger than the minimum native swap amount.
    - The remaining values are stored in the event data account, and no validation is performed.

`close_event_accounts`

This function does the following:

- Iterates through the remaining accounts to fetch `SwapEvent` accounts and payee accounts
- Checks that both accounts are writable
- Tries to deserialize the event account but emits a failure in case of failed deserialization due to a reorg
- Checks that the payee key is equal to `event_account.sender`
- Closes the event account and refunds the rent to the original payee
- Gets the index for the event account key from `swap_endpoint_data_account`
    - Deletes the index from `swap_endpoint_data_account`
    - Computes the new size by subtracting the pubkey size
    - Verifies that the new size is larger or equal to the minimum size
    - Reallocates `swap_endpoint_data_account` with the new size and transfers

the rent difference to the original payee

Accounts

- `data_account`
  - **Validation**: Account owner and discriminator are checked. Checked if swaps are suspended.
  - **Impact**: N/A.
- `agg_key`
  - **Validation**: Key validated through `data_account`.
  - **Impact**: N/A.
- `swap_endpoint_data_account`
  - **Validation**: Account owner and discriminator are checked.
  - **Impact**: Used to check if pubkeys for event accounts are valid.

Instruction parameters

N/A.

4.2. Module: vault

Verified key-gated instructions

The following instructions were verified to have proper access controls as intended per the provided documentation:

- ☑ `Initialize`
- ☑ `EnableTokenSupport`
- ☑ `DisableTokenSupport`
- ☑ `RotateAggKey`
- ☑ `FetchNative`
- ☑ `TransferTokens`
- ☑ `FetchTokens`
- ☐ `SwapNativeCtx`
- ☐ `SwapTokenCtx`
- ☑ `ExecuteCcmNativeCall`
- ☑ `ExecuteCcmTokenCall`
- ☑ `SetGovKeyWithAggKey`
- ☑ `GovKeySetValues`
- ☑ `UpgradeVaultProgram`
- ☑ `TransferUpgradeAuthority`

Swap token instructions

The swap token instructions are not key-gated. These functions do the following:

- Check that the swap amount is above the minimum amount required for that token
- Check that the cross-chain message fits the length requirements
- Transfer the token to the account that holds funds for the vault corresponding to the token
- Emit an event signalling to off-chain components that the swap has happened

These functions do not otherwise mutate the chain state of the vault, and thus key-gating is not necessary.

CCM call instructions

The CCM call instructions perform cross-program invocations on the local Solana chain. Cross-program invocations on Solana pass signer privileges to the called program when the account is passed through in the invocation. Notably, the `agg_key` is present as a signer on the CCM call instruction. This can potentially lead to a scenario where the called program calls back into the protocol's programs with the `agg_key` as a signer, escalating privileges. We have verified that this is not possible.

The protocol uses the following mitigation for the above scenario.

```
check_remaining_accounts(
    ctx.remaining_accounts,
    &[
        ctx.program_id,
        &ctx.accounts.data_account.agg_key,
        &ctx.accounts.data_account.token_vault_pda,
    ],
)?;
```

```
fn check_remaining_accounts(
    remaining_accounts: &[AccountInfo<'_>],
    keys: &[&Pubkey],
) -> Result<()> {
    for account in remaining_accounts.iter() {
        for &key in keys.iter() {
            require_keys_neq!(*account.key, *key,
    VaultError::InvalidRemainingAccount);
        }

        require!(
            !account.is_signer,
            VaultError::InvalidRemainingAccountSigner
```

```
            );
        }
        Ok(())
    }
```

This function is called on the `remaining_accounts`, which are passed into the called program. It requires that the `agg_key` and the program ID of the vault are not present. This prevents a call into the vault (since the program ID is missing) and prevents the `agg_key` from being used (since only accounts present in the caller can be passed to the callee). Furthermore, the function verifies that none of the accounts passed into the called program are signers. On Solana, an account can only be passed as a signer into a called program if it was a signer in the calling program.

The security of the above requires the following properties of Solana:

1. The program ID of a called program must exist as an account in the calling program.
2. Only accounts that are present in the caller can be passed into the callee.
3. Accounts that are passed as a signer or writable must have that property in the caller.

We have verified that the three properties hold on the current standard Solana validator client, Agave.

Cross-program invocations use the following Agave function.

```
fn cpi_common<S: SyscallInvokeSigned>(
    invoke_context: &mut InvokeContext,
    instruction_addr: u64,
    account_infos_addr: u64,
    account_infos_len: u64,
    signers_seeds_addr: u64,
    signers_seeds_len: u64,
    memory_mapping: &MemoryMapping,
)
```

In turn, `cpi_common` calls `InvokeContext::prepare_instruction`.

Property 1

The following code in `prepare_instruction` searches for the `callee_program_id` in the accounts of the current instruction (i.e., the caller). This code returns an error when it is not found.

```
// Find and validate executables / program accounts
let callee_program_id = instruction.program_id;
let program_account_index = instruction_context
```

```
    .find_index_of_instruction_account(self.transaction_context,
    &callee_program_id)
    .ok_or_else(|| {
        ic_msg!(self, "Unknown program {}", callee_program_id);
        InstructionError::MissingAccount
    })?;
let borrowed_program_account = instruction_context
    .try_borrow_instruction_account(self.transaction_context,
    program_account_index)?;
if !borrowed_program_account.is_executable() {
    ic_msg!(self, "Account {} is not executable", callee_program_id);
    return Err(InstructionError::AccountNotExecutable);
}
```

Property 2

The following code in `prepare_instruction` tries to find every account that is passed into the callee in the caller instruction. It returns an error when it is not found.

```
let index_in_caller = instruction_context
       .find_index_of_instruction_account(
           self.transaction_context,
           &account_meta.pubkey,
       )
       .ok_or_else(|| {
           ic_msg!(
               self,
               "Instruction references an unknown account {}",
               account_meta.pubkey,
           );
           InstructionError::MissingAccount
       })?;
```

Property 3

The following code in `prepare_instruction` checks that the privileges of the account in the callee cannot exceed the caller.

```
let borrowed_account = instruction_context.try_borrow_instruction_account(
    self.transaction_context,
    instruction_account.index_in_caller,
)?;
```

```
// Readonly in caller cannot become writable in callee
if instruction_account.is_writable && !borrowed_account.is_writable() {
    ic_msg!(
        self,
        "{}'s writable privilege escalated",
        borrowed_account.get_key(),
    );
    return Err(InstructionError::PrivilegeEscalation);
}

// To be signed in the callee,
// it must be either signed in the caller or by the program
if instruction_account.is_signer
    && !(borrowed_account.is_signer()
    || signers.contains(borrowed_account.get_key()))
{
    ic_msg!(
        self,
        "{}'s signer privilege escalated",
        borrowed_account.get_key()
    );
    return Err(InstructionError::PrivilegeEscalation);
}
```

5. Assessment Results

At the time of our assessment, the reviewed code was not deployed to the Solana Mainnet.

During our assessment on the scoped Chainflip contracts, we discovered three findings, all of which were informational in nature.

5.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.