FYEO

Security Assessment of the Streamflow Protocols

Streamflow Finance

April 2023
Version 1.1

Presented by:
FYEO Inc.
PO Box 147044
Lakewood CO 80214
United States

Security Level
Strictly Confidential

Table of Contents

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

OVERVIEW

Streamflow Finance engaged FYEO Inc. to perform a Security Assessment of the Streamflow Protocols.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on March 23 - March 29, 2023, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the FYEO Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

KEY FINDINGS

The following issues have been identified during the testing period. These should be prioritized for remediation to reduce the risk they pose:

- FYEO-SF-01 – Rent exemption on metadata account in create_unchecked not enforced
- FYEO-SF-02 – Code cleanup suggestions
- FYEO-SF-03 – Error variant is never constructed
- FYEO-SF-04 – Partner is a Vec, is limited to 250 entries
- FYEO-SF-05 – Payment period of 4 bytes would suffice
- FYEO-SF-06 – Superfluous return statements
- FYEO-SF-07 – Two create instructions imply that one has optional checks
- FYEO-SF-08 – Unnecessary while loop calculation

Based on account relationship graph analysis and our review process, we conclude that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

The FYEO Review Team performed a Security Assessment of the Streamflow Protocols. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a private repository at https://github.com/streamflow-finance/protocol with the commit hash c82ce81db42b78343826223a77ff50db87dec1e1.

A re-review took place on April 19, 2023, with the commit hash 66e00fbaf63f92e5b0c53e8d3d9050733575c033.

Files included in the code review
```
streamflow/
├── programs/
│   ├── protocol/
│   │   ├── src/
│   │   │   ├── cancel.rs
│   │   │   ├── create.rs
│   │   │   ├── create_unchecked.rs
│   │   │   ├── entrypoint.rs
│   │   │   ├── error.rs
│   │   │   ├── instruction.rs
│   │   │   ├── lib.rs
│   │   │   ├── pause.rs
│   │   │   ├── process.rs
│   │   │   ├── state.rs
│   │   │   ├── topup.rs
│   │   │   ├── transfer.rs
│   │   │   ├── try_math.rs
│   │   │   ├── unpause.rs
│   │   │   ├── update.rs
│   │   │   ├── utils.rs
│   │   │   └── withdraw.rs
│   │   ├── test-sdk/
│   │   │   ├── src/
│   │   │   │   ├── cookies.rs
│   │   │   │   ├── lib.rs
│   │   │   │   └── tools.rs
│   │   │   └── Cargo.toml
│   │   ├── tests/
│   │   │   ├── cancel_test.rs
│   │   │   ├── create_test.rs
│   │   │   ├── create_unchecked_test.rs
│   │   │   ├── fascilities.rs
```

Files included in the code review

```
│   │   │   ├── pause_test.rs
│   │   │   ├── state_test.rs
│   │   │   ├── topup_test.rs
│   │   │   ├── transfer_test.rs
│   │   │   ├── update_test.rs
│   │   │   └── withdraw_test.rs
│   │   ├── Cargo.lock
│   │   └── Cargo.toml
│   └── wrapper/
│       ├── src/
│       │   └── lib.rs
│       └── Cargo.toml
├── scripts/
│   └── localnet-set.sh
├── target/
│   └── deploy/
│       └── partner_oracle.so
├── tests/
│   ├── layout.js
│   ├── test-wallet.json
│   └── timelock.js
├── Anchor.toml
├── Cargo.lock
├── Cargo.toml
├── LICENSE
├── README.md
├── Xargo.toml
├── package-lock.json
├── package.json
└── rustfmt.toml
```

Table 1: Scope

TECHNICAL ANALYSES AND FINDINGS

During the Security Assessment of the Streamflow Protocols, we discovered:

- 1 finding with HIGH severity rating.
- 7 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

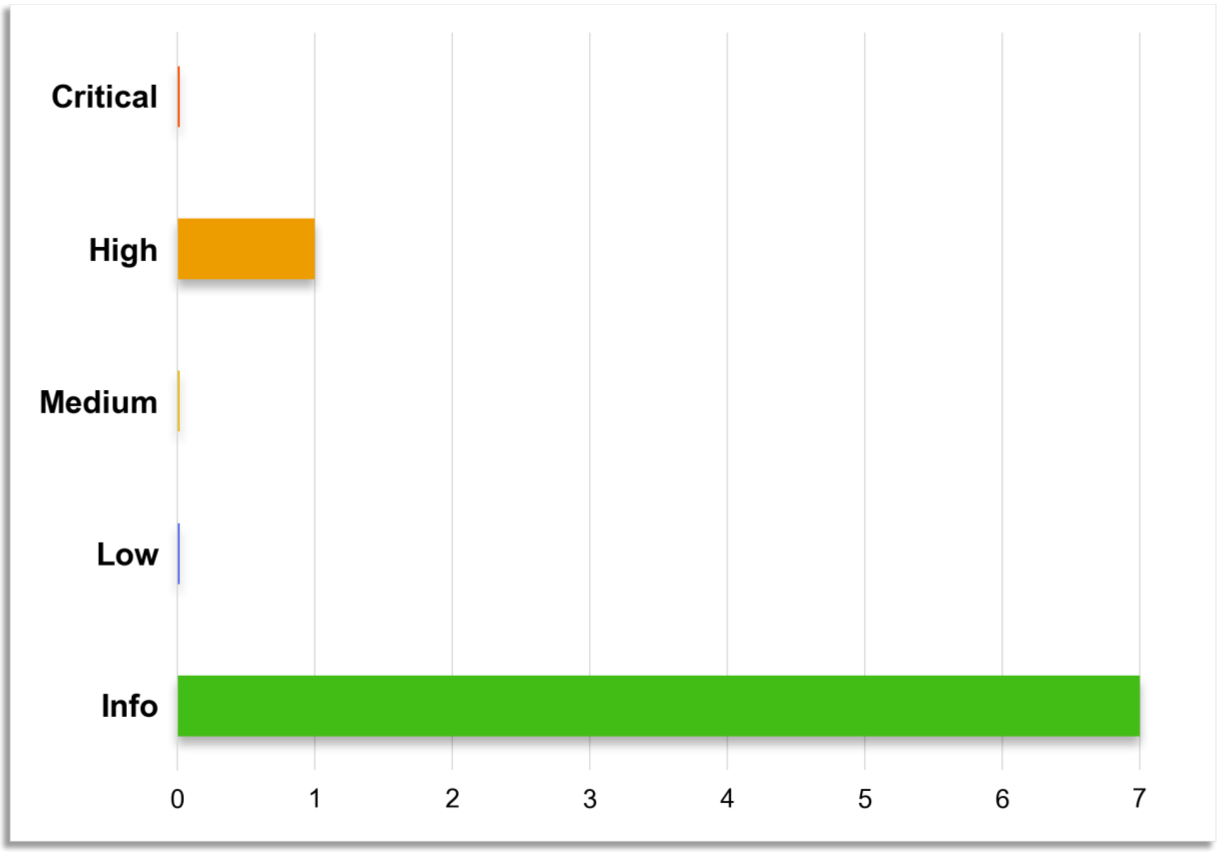


Figure 1: Findings by Severity

Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

Finding #	Severity	Description
FYEO-SF-01	**High**	Rent exemption on metadata account in create_unchecked not enforced
FYEO-SF-02	**Informational**	Code cleanup suggestions
FYEO-SF-03	**Informational**	Error variant is never constructed
FYEO-SF-04	**Informational**	Partner is a Vec, is limited to 250 entries
FYEO-SF-05	**Informational**	Payment period of 4 bytes would suffice
FYEO-SF-06	**Informational**	Superfluous return statements
FYEO-SF-07	**Informational**	Two create instructions imply that one has optional checks
FYEO-SF-08	**Informational**	Unnecessary while loop calculation

Table 2: Findings Overview

Technical Analysis

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

Authorization

The review used relationship graphs to show the relations between account input passed to the instructions of the program. The relations are used to verify if the authorization is sufficient for invoking each instruction. The graphs show if any unreferenced accounts exist. Accounts that are not referred to by trusted accounts can be replaced by any account of an attacker's choosing and thus pose a security risk.

In particular, the graphs will show if signing accounts are referred to. If a signing account is not referred to then any account can be used to sign the transaction causing insufficient authorization.

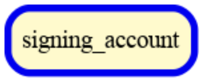
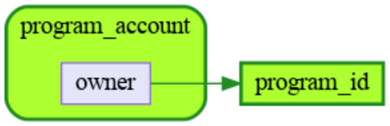
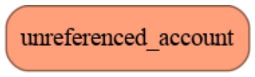
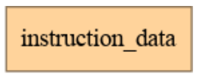
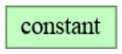
NO INSUFFICIENT AUTHORIZATION WAS FOUND BASED ON THE ANALYZES OF THE RELATIONSHIP GRAPHS. FOR DETAILS, SEE SECTION RELATIONSHIP GRAPHS

A relationship graph shows relational requirements to the input of an instruction. Notice, that it does not show outcome of the instruction.

Relationship graphs are used to analyze insufficient authorization and unchecked account relations. Insufficient authorization allows unintended access. Unchecked account relations may allow account and data injection resulting in unintended behavior and access.

Various styles are used to highlight special properties. Accounts are shown as boxes with round corners. An account box may contain smaller boxes indicating relevant account data.

The following table shows the basic styles for the relationship graphs.

	Round boxes with thick blue borders indicate accounts required to sign the transaction.
	Green round boxes highlight accounts required to be owned by the program itself.
	Red round boxes indicate accounts where ownership is not validated. Further analysis should be made to ensure this does not allow account injection attacks.
	Orange boxes indicate instruction data. As instruction data does not originate from the blockchain, it may open for data injection attacks. Caution must be taken if used for account validation.
	Green boxes indicate constants. It may refer to either hardcoded values or library code.

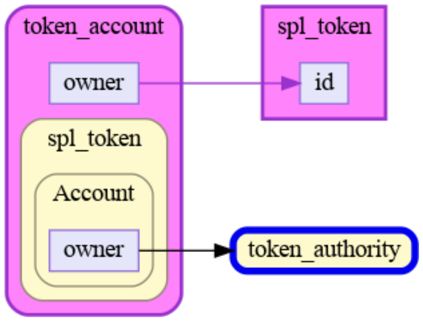
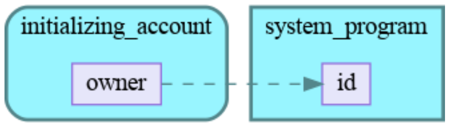
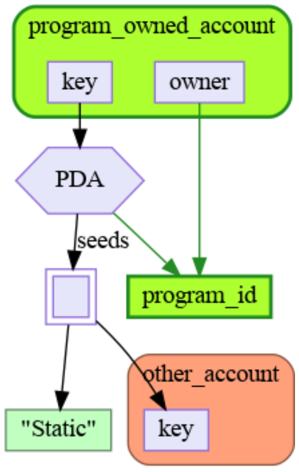
 token_account owner spl_token id spl_token Account owner token_authority	Inner boxes are used to indicate data structures to ease readability. Additional colors may also be used to highlight account ownership from different programs.
	Dashed lines indicate implicitly required relations. For example, relations required by another program during a cross program invocation. This is emphasized as the program cannot guarantee the behavior of external programs.
 program_owned_account key owner PDA seeds program_id other_account "Static" key	Program derived addresses are calculated using the `find_program_address` or `create_program_address` functions. To illustrate the relations to the input used for the PDA calculation, a diamond shaped box is used to show the PDA and a double-edged box to gather the seeds. As seed injection may lead to account injection, it is important that all input to the PDA is verified.

Table 3: Legend for relationship graphs starting on page 25.

Conclusion

Based on account relationship graph analysis and our review process, we conclude that the code implements the documented functionality to the extent of the reviewed code.

Technical Findings

General Observations

During the code review, it was noted that the Rust code is well-written. The code documentation is good but could be more detailed in certain areas. Additionally, some of the code needs to be cleaned up as it appears to have been refactored a few times. Despite these minor issues, the code is well-structured and easy to follow.

One area where the code really shines is in its thorough use of Solana account checks. The account checks performed by the program are excellent and show a deep understanding of Solana. Safe math has been used throughout the code base and shows commitment to building a secure program.

Overall, while there is room to simplify and clean up the Rust code, the program's use of Solana account checks is exemplary and demonstrates a strong understanding of Solana programming.

RENT EXEMPTION ON METADATA ACCOUNT IN CREATE_UNCHECKED NOT ENFORCED

Finding ID: FYEO-SF-01
Severity: **High**
Status: **Remediated**

Description

The metadata account is not checked to be rent exempt and could get deleted unexpectedly.

Proof of Issue

File name: programs/protocol/src/create_unchecked.rs
Line number: 43

```
#[derive(Clone, Debug)]
pub struct CreateAccounts<'a> {
    ...
    /// Expects empty (non-initialized) account.
    pub metadata: AccountInfo<'a>, // [writable]
    ...
```

The comment on the account is also wrong as it is expected to be initialized.

Line number: 96

```
if a.metadata.data_is_empty() {
    return Err(SfError::UninitializedMetadata.into())
}

if a.metadata.owner != pid {
    return Err(SfError::InvalidMetadataAccount.into())
}

// to achieve consistency with protocol side initialization
if a.metadata.data_len() < METADATA_LEN {
    return Err(SfError::InvalidMetadataSize.into())
}
```

Line number: 152

```
let data = acc.metadata.try_borrow_mut_data()?;
match solana_borsh::try_from_slice_unchecked::<Contract>(&data.to_vec()) {
    Ok(v) => {
        // state already initialized
        if v.created_at > 0 {
            return Err(SfError::InvalidMetadata.into())
        }
    },
    Err(_) => {},
};
```

Severity and Impact Summary

If the account given in this instruction does not have a rent exemption, it could get deleted at any time. Given that the metadata account is essential, this rent exemption should be enforced.

Recommendation

Make sure to check that the metadata account give has enough lamports to be rent exempt. Otherwise, add the required funds or fail.

CODE CLEANUP SUGGESTIONS

Finding ID: FYEO-SF-02
Severity: **Informational**
Status: **Remediated**

Description

The code would benefit from cleanup and the use of `cargo clippy`.

Proof of Issue

File name: programs/protocol/src/create.rs
Line number: 177

```
account_sanity_check(pid, acc.clone())?;
instruction_sanity_check(ix.clone(), now)?;
```

The use of `clone()` rather than using a reference `&` can be found in many other places (12+ over 6 files).

File name: programs/protocol/src/pause.rs
Line number: 71

```
if now >= metadata.end_time {
    return Err(SfError::InvalidMetadata.into())
}

if !metadata.ix.pausable {
    return Err(SfError::Unauthorized.into())
}

metadata.pause(now)?;
```

These checks are done in the handler - otherwise they are done in `metadata_sanity_check()`. It would be better to follow a consistent coding style to keep the code base more readable.

The check if `!metadata.ix.pausable` is also done by the function called in the next line `metadata.pause(now)?;`.

```
pub fn pause(&mut self, now: u64) -> Result<(), ProgramError> {
    if !self.ix.pausable {
        return Err(SfError::Unauthorized.into())
    }
    self.current_pause_start = now;
    Ok(())
}
```

The only other place calling `metadata.pause()` does so from an already paused state so presumably it was pausable.

File name: programs/protocol/src/topup.rs
Line number: 106

```
if !metadata.ix.can_topup {
    return Err(SfError::InvalidMetadata.into())
}
```

This check should be in `metadata_sanity_check()`.

File name: programs/protocol/src/create.rs
Line number: 136

```
if !(ix.amount_per_period > 0) {
```

This can be simplified as `ix.amount_per_period == 0`.

File name: programs/protocol/src/create.rs
Line number: 144

```
if !(ix.period > 0) {
```

This can be simplified as `ix.period == 0`.

File name: programs/protocol/src/update.rs
Line number: 219

```
if metadata.ix.automatic_withdrawal == true
```

This can be simplified as `metadata.ix.automatic_withdrawal`.

Severity and Impact Summary

No security impact but a maintainability improvement.

Recommendation

When passing data for reading, consider using references instead of copying larger structs. Also, keep code organized in the same fashion throughout the codebase. Consider the following improvements suggested by `cargo clippy`.

```
warning: question mark operator is useless here
   --> programs/protocol/src/state.rs:182:17
    |
182 |                 Ok(start.try_add(pause_cumulative)?.try_add(seconds_left)?)
    |                 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
help: try removing question mark and `Ok()`:
`start.try_add(pause_cumulative)?.try_add(seconds_left)`
...
196 | impl_create_params!(CreateParams);
    | --------------------------------- in this macro invocation
    |
    = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#needless_question_mark
    = note: `#[warn(clippy::needless_question_mark)]` on by default
    = note: this warning originates in the macro `impl_create_params` (in Nightly
builds, run with -Z macro-backtrace for more info)

warning: question mark operator is useless here
   --> programs/protocol/src/state.rs:182:17
```

```
    |
182 |                  Ok(start.try_add(pause_cumulative)?.try_add(seconds_left)?)
    |                  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
help: try removing question mark and `Ok()`:
`start.try_add(pause_cumulative)?.try_add(seconds_left)`
...
197 | impl_create_params!(CreateParamsUnchecked);
    | ------------------------------------------ in this macro invocation
    |
    = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#needless_question_mark
    = note: this warning originates in the macro `impl_create_params` (in Nightly
builds, run with -Z macro-backtrace for more info)

warning: unneeded `return` statement
   --> programs/protocol/src/state.rs:245:9
    |
245 | /         return Participants {
246 | |             sender: *sender,
247 | |             sender_tokens: *sender_tokens,
248 | |             recipient: *recipient,
...   |
253 | |             partner_tokens,
254 | |         }
    | |_________^
    |
    = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#needless_return
    = note: `#[warn(clippy::needless_return)]` on by default
    = help: remove `return`

warning: this function has too many arguments (8/7)
   --> programs/protocol/src/state.rs:416:5
    |
416 | /     pub fn new_with_participants(
417 | |         now: u64,
418 | |         acc: create_unchecked::CreateAccounts,
419 | |         ix: CreateParamsUnchecked,
...   |
424 | |         participants: Participants,
425 | |     ) -> Result<Self, ProgramError> {
    | |___________________________________^
    |
    = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#too_many_arguments
    = note: `#[warn(clippy::too_many_arguments)]` on by default

warning: unneeded `return` statement
   --> programs/protocol/src/state.rs:490:9
    |
490 | /         return match self.state()? {
491 | |             ContractState::Paused => {
492 | |
Ok(self.pause_cumulative.try_add(now)?.try_sub(self.current_pause_start)?)
493 | |             }
494 | |             _ => Ok(self.pause_cumulative),
495 | |         }
    | |_________^
    |
    = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#needless_return
    = help: remove `return`

warning: question mark operator is useless here
   --> programs/protocol/src/state.rs:672:9
    |
672 |
Ok(effective_start.try_add(seconds_left)?.try_add(self.pause_cumulative)?)
    |
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ help: try
removing question mark and `Ok()`:
`effective_start.try_add(seconds_left)?.try_add(self.pause_cumulative)`
    |
    = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#needless_question_mark

warning: question mark operator is useless here
  --> programs/protocol/src/utils.rs:81:5
   |
81 |     Ok(sum_available.try_sub(withdrawn)?)
   |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ help: try removing question mark and
`Ok()`: `sum_available.try_sub(withdrawn)`
   |
   = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#needless_question_mark

warning: question mark operator is useless here
  --> programs/protocol/src/utils.rs:74:16
   |
74 |         return Ok(total.try_sub(withdrawn)?)
   |                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ help: try removing question mark and
`Ok()`: `total.try_sub(withdrawn)`
   |
   = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#needless_question_mark

warning: question mark operator is useless here
  --> programs/protocol/src/utils.rs:93:5
   |
93 |     Ok(balance.try_sub(deposited.try_sub(withdrawn)?)?)
   |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ help: try removing
question mark and `Ok()`: `balance.try_sub(deposited.try_sub(withdrawn)?)`
   |
   = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#needless_question_mark

warning: redundant clone
   --> programs/protocol/src/create.rs:218:11
    |
218 |         ix.clone(),
    |           ^^^^^^^^ help: remove this
    |
note: this value is dropped without further use
   --> programs/protocol/src/create.rs:218:9
    |
218 |         ix.clone(),
    |         ^^
    = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#redundant_clone
    = note: `#[warn(clippy::redundant_clone)]` on by default

warning: this boolean expression can be simplified
   --> programs/protocol/src/create_unchecked.rs:119:8
    |
119 |     if !(ix.amount_per_period > 0) {
    |        ^^^^^^^^^^^^^^^^^^^^^^^^^^^ help: try: `ix.amount_per_period <= 0`
    |
    = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#nonminimal_bool

warning: this boolean expression can be simplified
   --> programs/protocol/src/create_unchecked.rs:127:8
    |
127 |     if !(ix.period > 0) {
    |        ^^^^^^^^^^^^^^^^ help: try: `ix.period <= 0`
    |
    = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#nonminimal_bool

warning: redundant clone
   --> programs/protocol/src/create_unchecked.rs:222:11
    |
222 |         ix.clone(),
    |           ^^^^^^^^ help: remove this
    |
note: this value is dropped without further use
   --> programs/protocol/src/create_unchecked.rs:222:9
    |
222 |         ix.clone(),
    |         ^^
    = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#redundant_clone

warning: you seem to be trying to use `match` for destructuring a single pattern.
Consider using `if let`
   --> programs/protocol/src/create_unchecked.rs:153:9
    |
153 | / ...   match solana_borsh::try_from_slice_unchecked::<Contract>(&data.to_vec())
{
154 | | ...       Ok(v) => {
155 | | ...           // state already initialized
156 | | ...           if v.created_at > 0 {
...   |
160 | | ...       Err(_) => {},
161 | | ...   };
    | |_______^
    |
    = help: for further information visit https://rust-lang.github.io/rust-
```

```
clippy/master/index.html#single_match
    = note: `#[warn(clippy::single_match)]` on by default
help: try this
    |
153 ~          if let Ok(v) =
solana_borsh::try_from_slice_unchecked::<Contract>(&data.to_vec()) {
154 +              // state already initialized
155 +              if v.created_at > 0 {
156 +                  return Err(SfError::InvalidMetadata.into())
157 +              }
158 ~          };
    |

warning: unnecessary closure used to substitute value for `Option::None`
  --> programs/protocol/src/update.rs:83:9
   |
83 |          params.withdraw_frequency.unwrap_or_else(||
metadata.ix.withdraw_frequency);
   |          ^^^^^^^^^^^^^^^^^^^^^^^^^^-----------------------------------------------
--
   |                                    |
   |                                    help: use `unwrap_or(..)` instead:
`unwrap_or(metadata.ix.withdraw_frequency)`
   |
   = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#unnecessary_lazy_evaluations
   = note: `#[warn(clippy::unnecessary_lazy_evaluations)]` on by default

warning: this expression creates a reference which is immediately dereferenced by the
compiler
   --> programs/protocol/src/update.rs:198:26
    |
198 |     account_sanity_check(&pid, &acc)?;
    |                          ^^^^ help: change this to: `pid`
    |
    = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#needless_borrow
    = note: `#[warn(clippy::needless_borrow)]` on by default

warning: equality checks against true are unnecessary
   --> programs/protocol/src/update.rs:219:12
    |
219 |          if metadata.ix.automatic_withdrawal == true {
    |             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ help: try simplifying it as
shown: `metadata.ix.automatic_withdrawal`
    |
    = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#bool_comparison
    = note: `#[warn(clippy::bool_comparison)]` on by default

warning: `protocol` (lib) generated 21 warnings
    Checking wrapper v0.1.0 (streamflow/programs/wrapper)
warning: this function has too many arguments (17/7)
  --> programs/wrapper/src/lib.rs:17:5
   |
17 | /     pub fn create(
18 | |         ctx: Context<Create>,
19 | |         start_time: u64,
20 | |         net_amount_deposited: u64,
...  |
34 | |         can_update_rate: Option<bool>,
35 | |     ) -> ProgramResult {
   | |______________________^
   |
   = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#too_many_arguments
   = note: `#[warn(clippy::too_many_arguments)]` on by default

warning: unnecessary closure used to substitute value for `Option::None`
  --> programs/wrapper/src/lib.rs:36:26
   |
36 |         let pause_flag = pausable.unwrap_or_else(|| false);
   |                          ^^^^^^^^^------------------------
   |                                   |
   |                                   help: use `unwrap_or(..)` instead:
`unwrap_or(false)`
   |
   = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#unnecessary_lazy_evaluations
   = note: `#[warn(clippy::unnecessary_lazy_evaluations)]` on by default

warning: unnecessary closure used to substitute value for `Option::None`
  --> programs/wrapper/src/lib.rs:37:31
   |
37 |         let can_update_rate = can_update_rate.unwrap_or_else(|| false);
   |                               ^^^^^^^^^^^^^^^^------------------------
   |                                                |
   |                                                help: use `unwrap_or(..)` instead:
`unwrap_or(false)`
   |
   = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#unnecessary_lazy_evaluations

warning: this function has too many arguments (19/7)
   --> programs/wrapper/src/lib.rs:80:5
    |
80  | /     pub fn create_unchecked(
81  | |         ctx: Context<CreateUnchecked>,
82  | |         start_time: u64,
83  | |         net_amount_deposited: u64,
...   |
99  | |         can_update_rate: bool,
100 | |     ) -> ProgramResult {
    | |______________________^
    |
    = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#too_many_arguments

warning: this function has too many arguments (19/7)
   --> programs/wrapper/src/lib.rs:140:5
    |
140 | /     pub fn create_unchecked_with_payer(
141 | |          ctx: Context<CreateUncheckedWithPayer>,
142 | |          start_time: u64,
143 | |          net_amount_deposited: u64,
...   |
159 | |          can_update_rate: bool,
160 | |      ) -> ProgramResult {
    | |______________________^
    |
    = help: for further information visit https://rust-lang.github.io/rust-
clippy/master/index.html#too_many_arguments
```

ERROR VARIANT IS NEVER CONSTRUCTED

Finding ID: FYEO-SF-03
Severity: **Informational**
Status: **Remediated**

Description

The function never constructs a `ProgramError` and can therefore be simplified.

Proof of Issue

File name: programs/protocol/src/state.rs
Line number: 479

```
pub fn state(&self) -> Result<ContractState, ProgramError> {
    if self.closed {
        return Ok(ContractState::Closed)
    }
    if self.current_pause_start > 0 {
        return Ok(ContractState::Paused)
    }
    Ok(ContractState::Scheduled)
}
```

Severity and Impact Summary

No security impact. Just improves code clarity.

Recommendation

Return `ContractState`.

Partner is a Vec, is limited to 250 entries

Finding ID: FYEO-SF-04
Severity: **Informational**
Status: **Open**

Description

The partner oracle programs' `Partner` account stores its data in a `Vec` that limits the number of possible Partners to the size the account was initialized with - currently to 250 entries at 10000 bytes / (32+4+4).

Proof of Issue

File name: partner-oracle/src/entrypoint.rs
Line number: 42

```
// The data bytes we'll allocate for the new account.
let metadata_alloc = 10000;
let cluster_rent = Rent::get()?;
let metadata_rent = cluster_rent.minimum_balance(metadata_alloc);
```

Severity and Impact Summary

The number of partners is limited to 250 entries.

Recommendation

Consider using PDAs based on the partner's key to increasing this limit.

Payment period of 4 bytes would suffice

Finding ID: FYEO-SF-05
Severity: **Informational**
Status: **Open**

Description

The payment period would probably be fine with 4 bytes rather than 8. This would already allow a payment period of "pay every 130+ years".

Proof of Issue

File name: programs/protocol/src/state.rs
Line number: 32

```
#[derive(BorshDeserialize, BorshSerialize, Clone, Debug)]
#[repr(C)]
pub struct CreateParamsUnchecked {
    /// Timestamp when the tokens start vesting
    pub start_time: u64,
    /// Deposited amount of tokens
    pub net_amount_deposited: u64,
    /// Time step (period) in seconds per which the vesting/release occurs
    pub period: u64,
```

Severity and Impact Summary

While not a large amount, this can reduce the contract size by 4 bytes.

Recommendation

Consider using 4 bytes.

SUPERFLUOUS RETURN STATEMENTS

Finding ID: FYEO-SF-06
Severity: **Informational**
Status: **Remediated**

Description

The `process_instruction` uses a `match` which has 10x return statements that are superfluous. The word "return" can be removed in every line.

Proof of Issue

File name: programs/protocol/src/entrypoint.rs
Line number: 26

```
match instruction {
    StreamInstruction::Create { create_params } => {
        ...
        return create(pid, ia, create_params)
    }

    StreamInstruction::CreateUnchecked { create_params } => {
        ...
        return create_unchecked::create(pid, ia, create_params)
    }
    StreamInstruction::Withdraw { amount } => {
        ...
        return withdraw(pid, ia, amount)
    }
    StreamInstruction::Cancel {} => {
        ...
        return cancel(pid, ia)
    }

    ...
```

Severity and Impact Summary

No security impact.

Recommendation

Remove unnecessary code.

TWO CREATE INSTRUCTIONS IMPLY THAT ONE HAS OPTIONAL CHECKS

Finding ID: FYEO-SF-07
Severity: **Informational**
Status: **Open**

Description

There are two create instructions, one of which has less checks than the other. Meaning all extra checks in the 'checked"' one become optional. Also, this leads to a lot of duplicate code and makes maintenance harder. The unchecked version could instead do additional checks if requested using some instruction parameter.

Proof of Issue

File name: programs/protocol/src/entrypoint.rs
Line number: 26

```
match instruction {
    StreamInstruction::Create { create_params } => {
        ...
        return create(pid, ia, create_params)
    }

    StreamInstruction::CreateUnchecked { create_params } => {
        ...
        return create_unchecked::create(pid, ia, create_params)
    }
    ...
```

Severity and Impact Summary

Having two instructions with the same outcome makes all additional checks optional. But does so at a loss of code maintainability by increasing complexity.

Recommendation

Use the unchecked version as the base and implement some parameter that triggers the additional optional checks.

UNNECESSARY WHILE LOOP CALCULATION

Finding ID: FYEO-SF-08
Severity: **Informational**
Status: **Remediated**

Description

This code pads the size to make it divisible by 8. The while loop can be simplified as `+= 8 – (metadata_struct_size % 8)`

Proof of Issue

File name: programs/protocol/src/create.rs
Line number: 228

```
while metadata_struct_size % 8 > 0 {
    metadata_struct_size += 1;
}
```

Severity and Impact Summary

No security impact. It is code optimization.

Recommendation

Calculate this without using a loop.

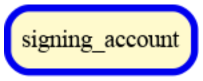
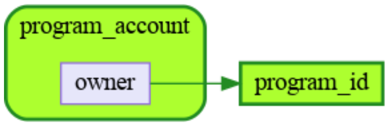
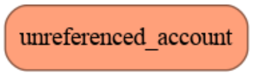
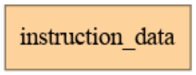
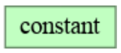
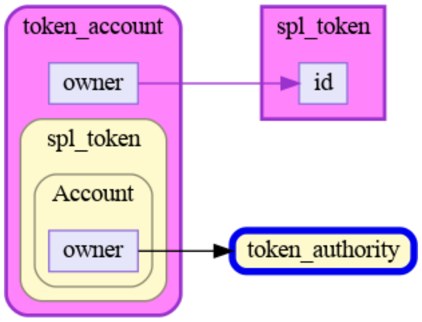
Relationship Graphs

A relationship graph shows relational requirements to the input of an instruction. Notice, that it does not show outcome of the instruction.

Relationship graphs are used to analyze insufficient authorization and unchecked account relations. Insufficient authorization allows unintended access. Unchecked account relations may allow account and data injection resulting in unintended behavior and access.

Various styles are used to highlight special properties. Accounts are shown as boxes with round corners. An account box may contain smaller boxes indicating relevant account data.

The following table shows the basic styles for the relationship graphs.

	Round boxes with thick blue borders indicate accounts required to sign the transaction.
	Green round boxes highlight accounts required to be owned by the program itself.
	Red round boxes indicate accounts where ownership is not validated. Further analysis should be made to ensure this does not allow account injection attacks.
	Orange boxes indicate instruction data. As instruction data does not originate from the blockchain, it may open for data injection attacks. Caution must be taken if used for account validation.
	Green boxes indicate constants. It may refer to either hardcoded values or library code.
 token_account owner spl_token Account owner spl_token id token_authority	Inner boxes are used to indicate data structures to ease readability. Additional colors may also be used to highlight account ownership from different programs.

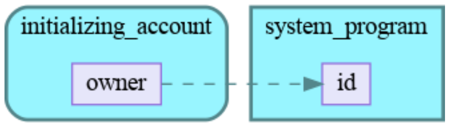
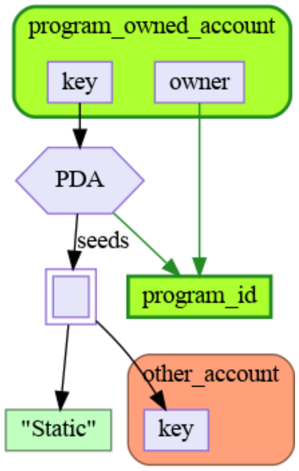
	Dashed lines indicate implicitly required relations. For example, relations required by another program during a cross program invocation. This is emphasized as the program cannot guarantee the behavior of external programs.
 program_owned_account key owner PDA seeds program_id other_account key "Static"	Program derived addresses are calculated using the `find_program_address` or `create_program_address` functions. To illustrate the relations to the input used for the PDA calculation, a diamond shaped box is used to show the PDA and a double-edged box to gather the seeds. As seed injection may lead to account injection, it is important that all input to the PDA is verified.

Table 3: Legend for relationship graphs

cancel

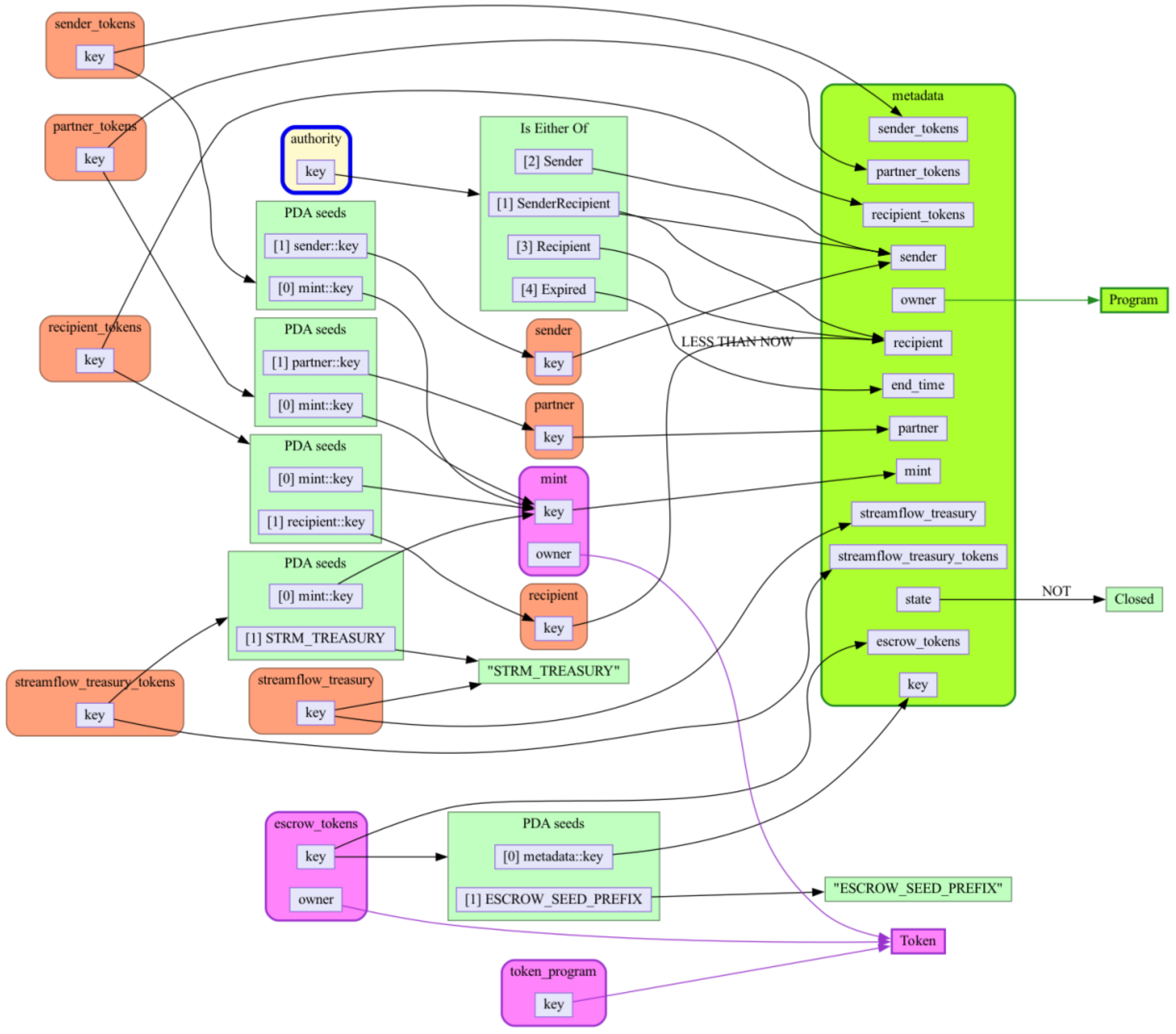


Figure 2: cancel

This instruction closes a payment stream, pending tokens will be sent to the recipient and refunds to the sender.

create

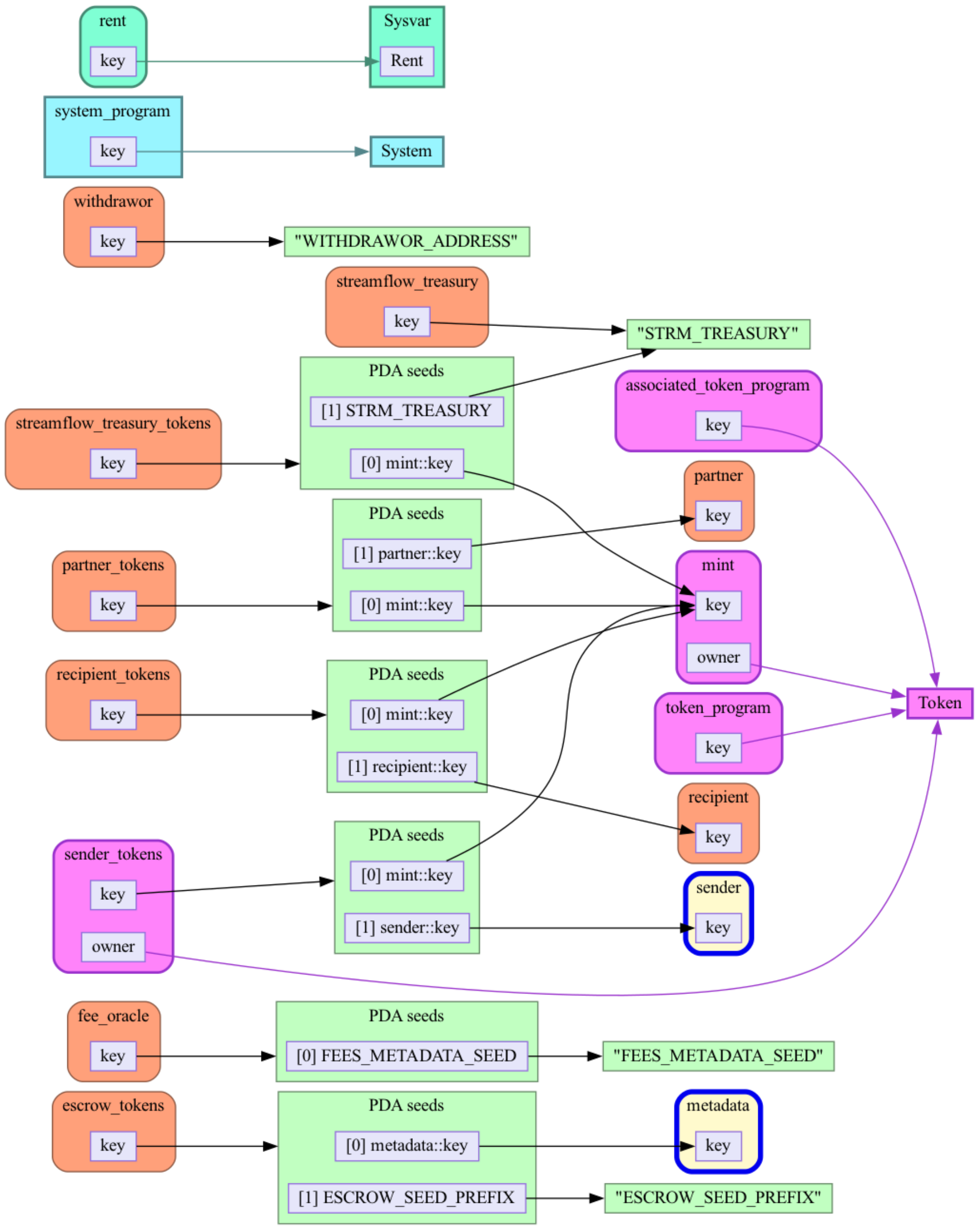


Figure 3: create

This instruction creates a payment stream between two parties. The funds are transferred into an escrow account.

create_unchecked

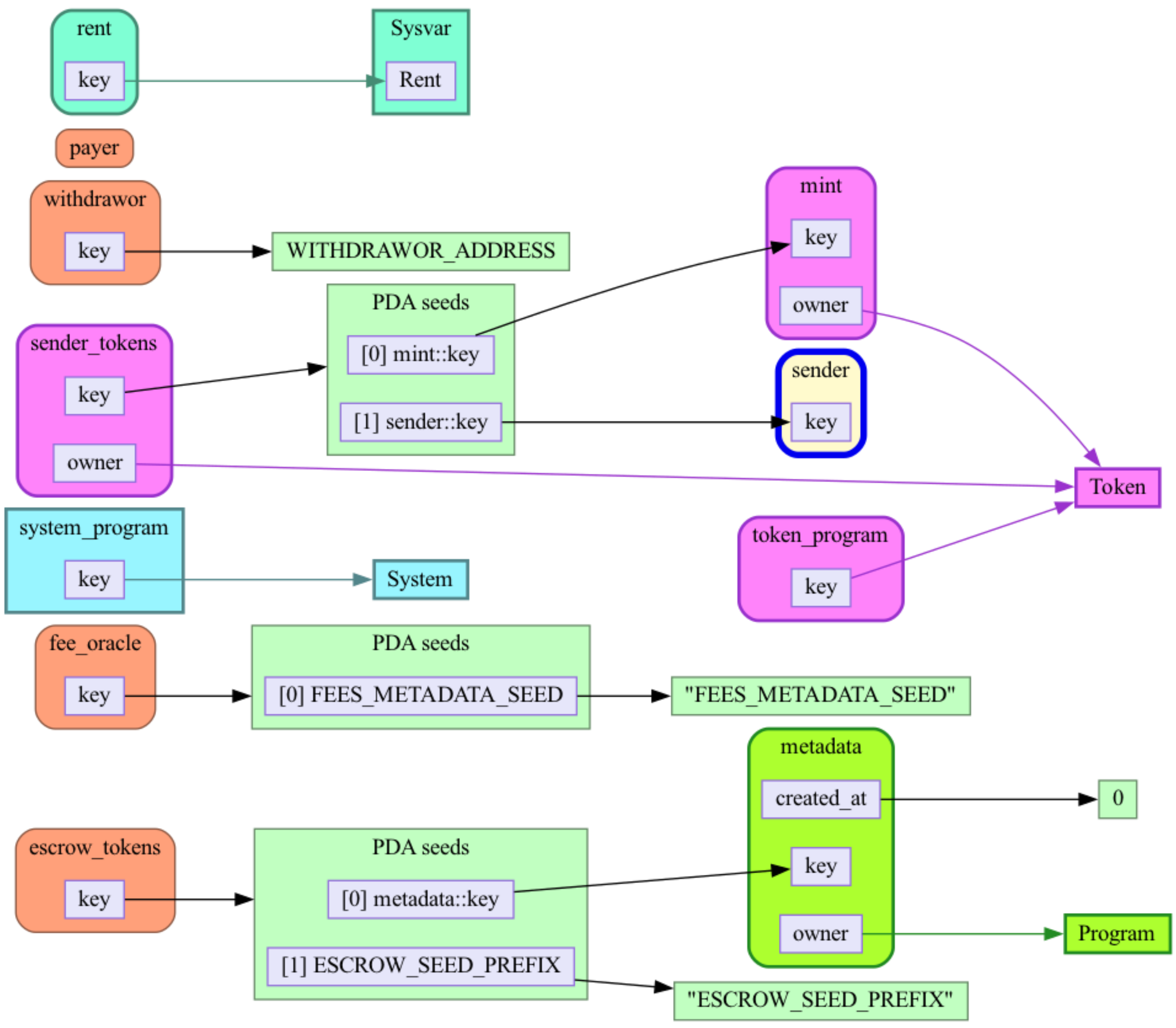


Figure 4: create_unchecked

This instruction creates a payment stream between two parties. The funds are transferred into an escrow account.

pause

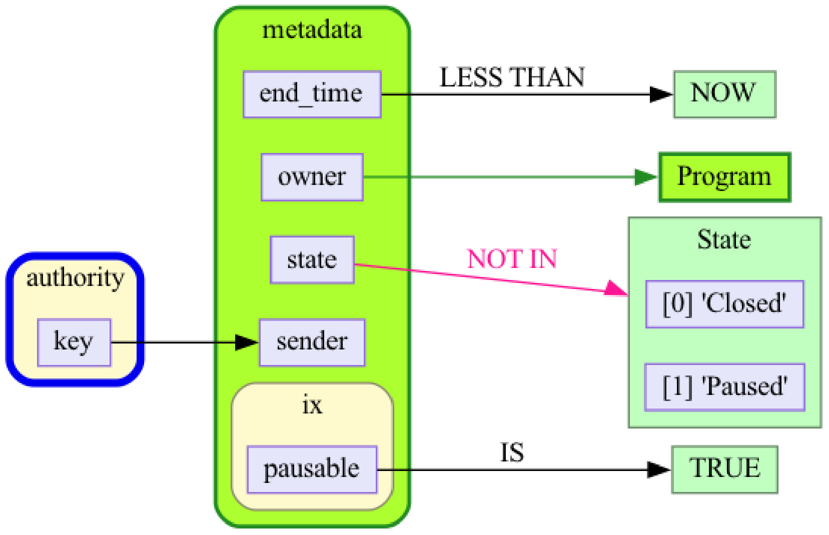


Figure 5: pause

This instruction pauses a payment stream.

topup

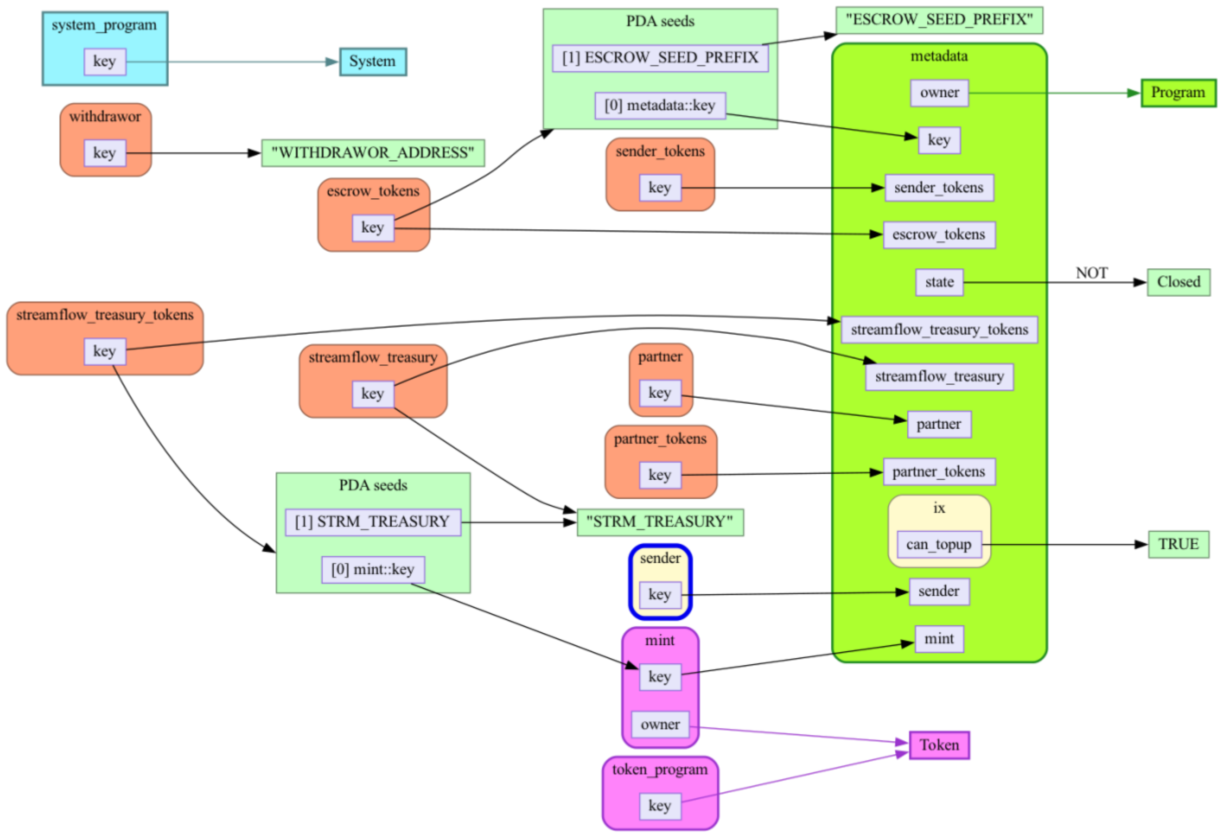


Figure 6: topup

This instruction adds funds to a payment stream.

transfer

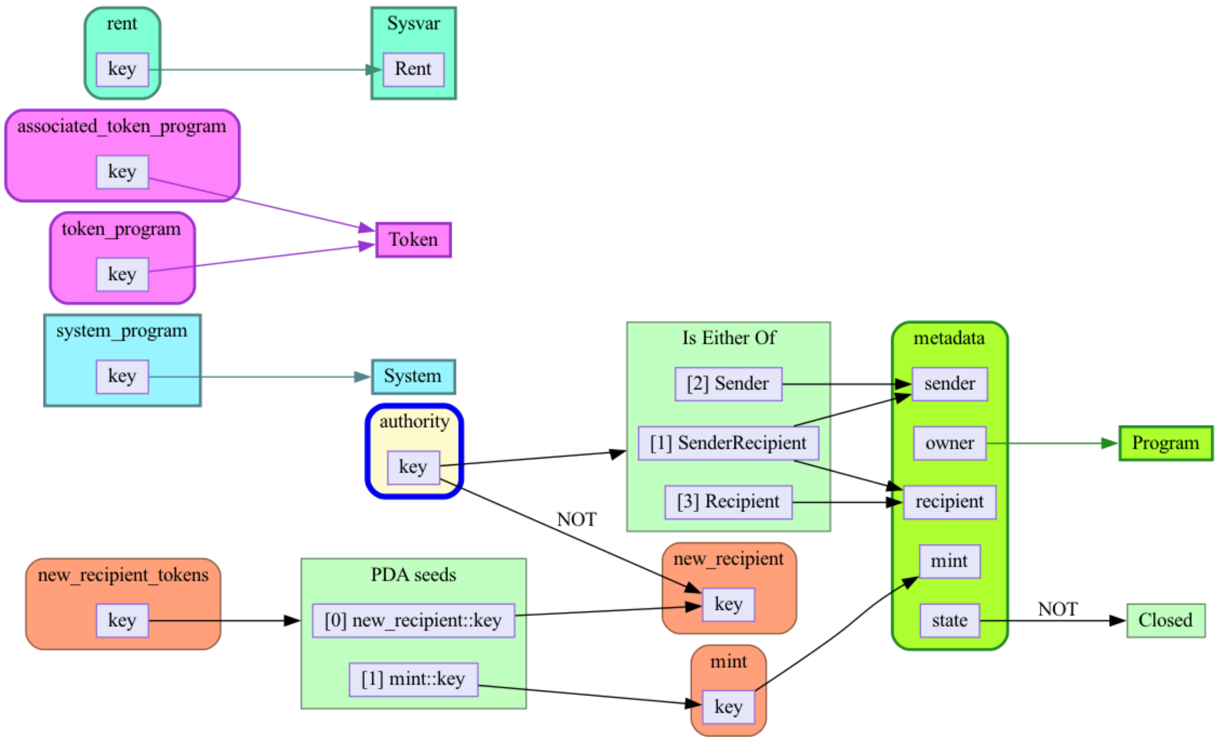


Figure 7: transfer

Replaces the recipient with another.

unpause

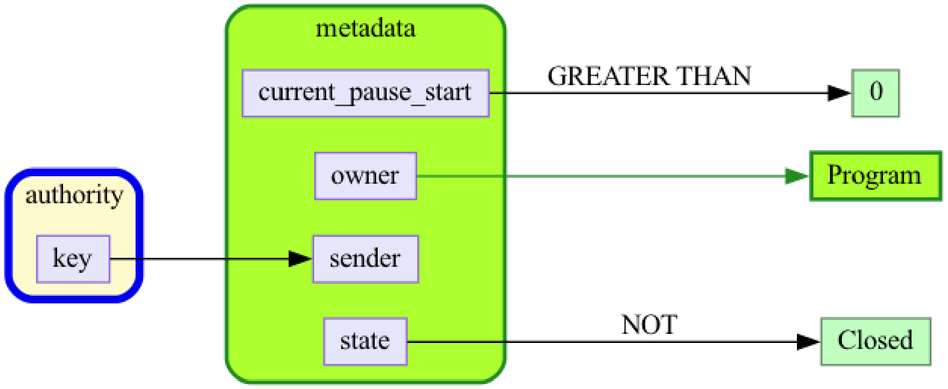


Figure 8: unpause

This instruction unpauses a payment stream.

update

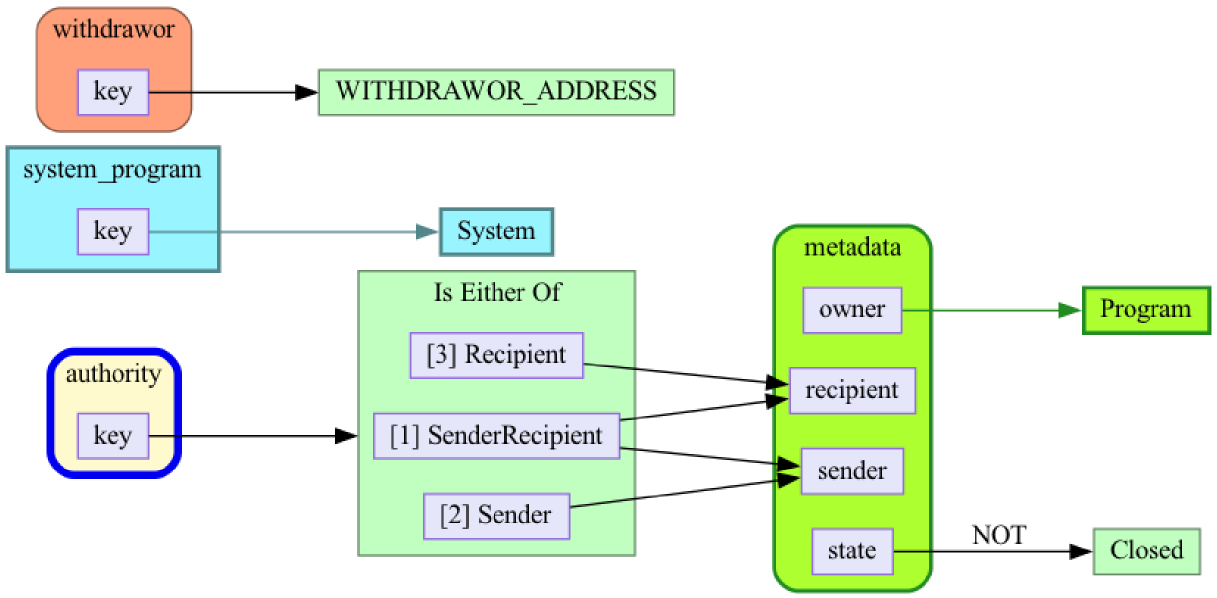


Figure 9: update

This instruction reconfigures a payment stream.

withdraw

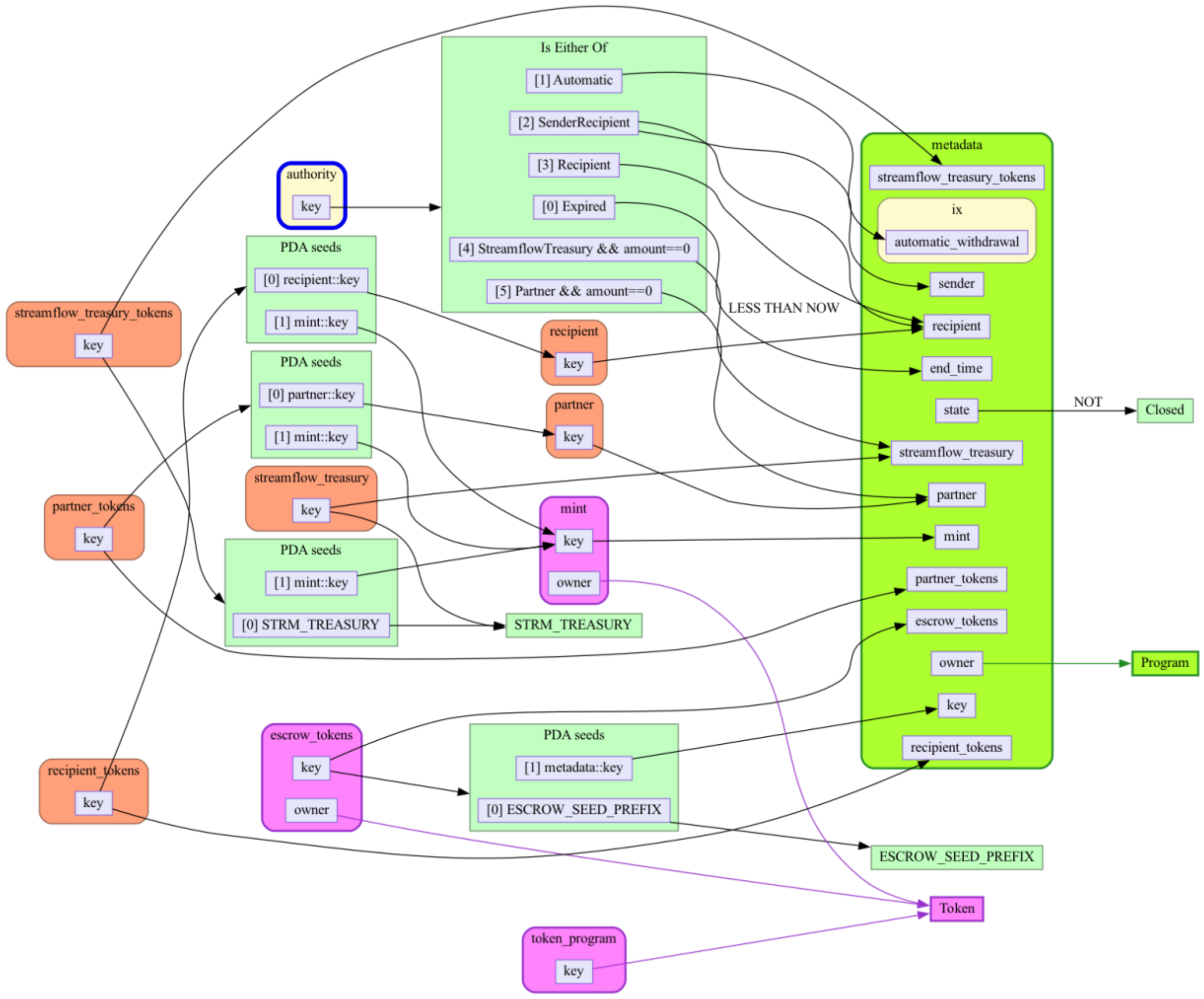


Figure 10: withdraw

This instruction withdraws unlocked funds from the escrow account to the recipient or fee collectors.

OUR PROCESS

Methodology

FYEO Inc. uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

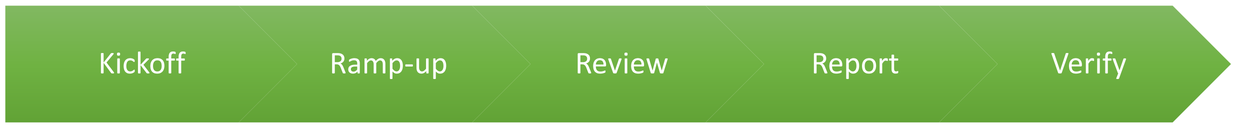


Figure 11: Methodology Flow

Kickoff

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

FYEO Inc. delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.

THE CLASSIFICATION OF VULNERABILITIES

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations