

# Solend Crypto & Digital Asset Assessment

Findings and Recommendations Report Presented to:

**Solend Project & Solana Foundation**

August 06, 2021  
Version: 0.1

Presented by:

Kudelski Security, Inc.  
5090 North 40th Street, Suite 450  
Phoenix, Arizona 85018

STRICTLY CONFIDENTIAL

## TABLE OF CONTENTS

TABLE OF CONTENTS .....	2
LIST OF FIGURES .....	3
LIST OF TABLES .....	3
EXECUTIVE SUMMARY .....	4
Overview .....	4
Key Findings.....	4
Scope and Rules Of Engagement.....	5
TECHNICAL ANALYSIS & FINDINGS .....	6
Findings.....	7
Technical analysis .....	8
Conclusion.....	17
Technical Findings .....	18
Loss of precision causing miscalculation of interest rate .....	18
Structs implementing bytemuck::Pod contains non-Pod fields.....	20
Pyth product parsing may cause index-out-of-bounds .....	23
Low test coverage creates a risk for maintenance .....	26
Disabled lint checks may introduce bad code practices.....	27
Oracle (Pyth) program id stored in LendingMarket is not validated .....	28
SPL Token program id stored in LendingMarket is superfluous.....	30
METHODOLOGY .....	33
Kickoff.....	33
Ramp-up.....	33
Review.....	33
Code Safety.....	34
Cryptography.....	34
Technical Specification Matching .....	34
Reporting .....	35
Verify .....	35
Additional Note .....	35
The Classification of identified problems and vulnerabilities .....	36
Critical – vulnerability that will lead to loss of protected assets.....	36
High - A vulnerability that can lead to loss of protected assets .....	36
Medium - a vulnerability that hampers the uptime of the system or can lead to other problems ....	36
Low - Problems that have a security impact but does not directly impact the protected assets .....	36
Informational.....	36
Tools	37

RustSec.org.....	37
KUDELSKI SECURITY CONTACTS.....	39

## LIST OF FIGURES

Figure 1: Findings by Severity .....	6
Figure 2: Account reference graph for BorrowObligationLiquidity .....	8
Figure 3: Account reference graph for DepositObligationCollateral .....	8
Figure 4: Account reference graph for DepositReserveLiquidity .....	9
Figure 5: Account reference graph for FlashLoan .....	10
Figure 6: Account reference graph for InitLendingMarket .....	10
Figure 7: Account reference graph for InitObligation .....	11
Figure 8: Account reference graph for InitReserve .....	11
Figure 9: Account reference graph for LiquidateObligation .....	12
Figure 10: Account reference graph for RedeemReserveCollateral.....	13
Figure 11: Account reference graph for RefreshObligation .....	14
Figure 12: Account reference graph for RefreshReserve .....	14
Figure 13: Account reference graph for RepayObligationLiquidity .....	15
Figure 14: Account reference graph for SetLendingMarketOwner .....	15
Figure 15: Account reference graph for WithdrawObligationCollateral .....	16
Figure 16: Methodology Flow .....	33

## LIST OF TABLES

Table 1: Scope .....	5
Table 2: Findings Overview .....	7

## EXECUTIVE SUMMARY

### Overview

Solend Project & Solana Foundation engaged Kudelski Security to perform a Solend Crypto & Digital Asset Assessment.

The assessment was conducted remotely by the Kudelski Security Team. Testing took place on July 12-August 3, 2021, and focused on the following objectives:

- 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 result of our tests.

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

### Key Findings

The following are the major themes and issues identified during the testing period. These, along with other items, within the findings section, should be prioritized for remediation to reduce to the risk they pose.

- KS-SOLEND-F-01 – Loss of precision causing miscalculation of interest rate
- KS-SOLEND-F-02 – Structs implementing bytemuck::Pod contains non-Pod fields
- KS-SOLEND-F-03 – Pyth product parsing may cause index-out-of-bounds

During the test, the following positive observations were noted regarding the scope of the engagement:

- The code was very well documented and had a really high production standard
- The development team was very good at explaining and engaging in discussions

Based on the call graphs and the formal verification we can conclude that the reviewed code implements the documented functionality.

## Scope and Rules Of Engagement

Kudelski performed an Solend Crypto & Digital Asset Assessment for Solend Project & Solana Foundation. 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 public repository at <https://github.com/solana-labs/solana-program-library/tree/master/token-lending> with the commit hash ba0c0e007f99857894f238638b60cacb41281114

Files included in the code review	
program/src/processor.rs	The main program file
program/src/entrypoint.rs	Program entrypoint definitions
program/src/instruction.rs	Instruction types
program/src/state/reserve.rs	Lending market reserve
program/src/state/obligation.rs	Obligations definition and utils
program/src/state/lending_market.rs	The lending market
program/src/state/last-update.rs	Utils for slots and updates
program/src/math/common.rs	Common math utilities definitions
program/src/math/decimal.rs	Decimal utilities and definitions
program/src/math/mod.rs	Include file for math
program/src/math/rate.rs	Utilities for ratios and percentages

Table 1: Scope

## TECHNICAL ANALYSIS & FINDINGS

During the Solend Crypto & Digital Asset Assessment, we discovered 1 findings that had a HIGH severity rating, as well as 1 MEDIUM

The following chart displays the findings by severity.

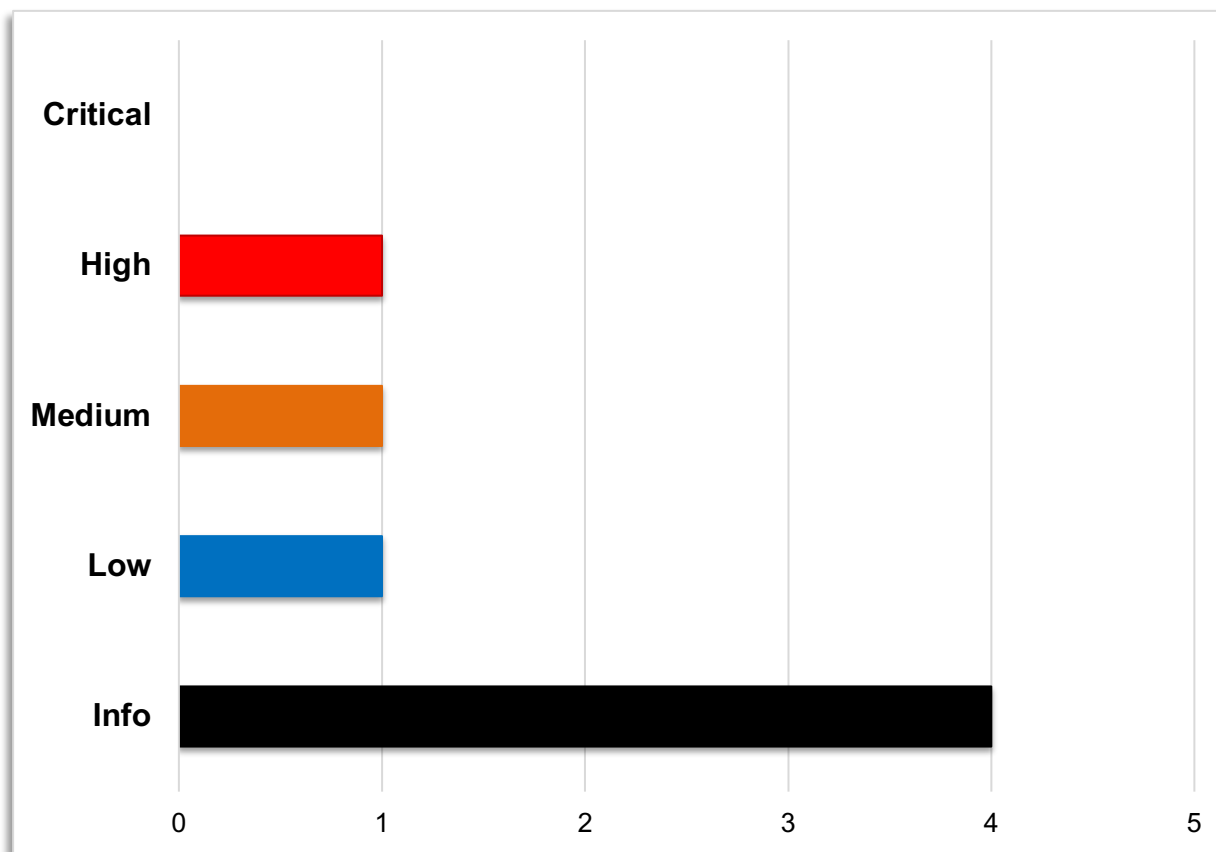


Figure 1: Findings by Severity

## 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.

#	Severity	Description
KS-SOLEND-F-01	High	Loss of precision causing miscalculation of interest rate
KS-SOLEND-F-02	Medium	Structs implementing bytemuck::Pod contains non-Pod fields
KS-SOLEND-F-03	Low	Pyth product parsing may cause index-out-of-bounds
KS-SOLEND-F-04	Informational	Low test coverage creates a risk for maintenance
KS-SOLEND-F-05	Informational	Disabled lint checks may introduce bad code practices
KS-SOLEND-F-06	Informational	Oracle (Pyth) program id stored in LendingMarket is not validated
KS-SOLEND-F-07	Informational	SPL Token program id stored in LendingMarket is superfluous

Table 2: Findings Overview

## Technical analysis

Based on the source code the following call graphs was made to verify the validity of the code as well as confirming that the intended functionality was implemented correctly and to the extent that the state of the repository allowed.

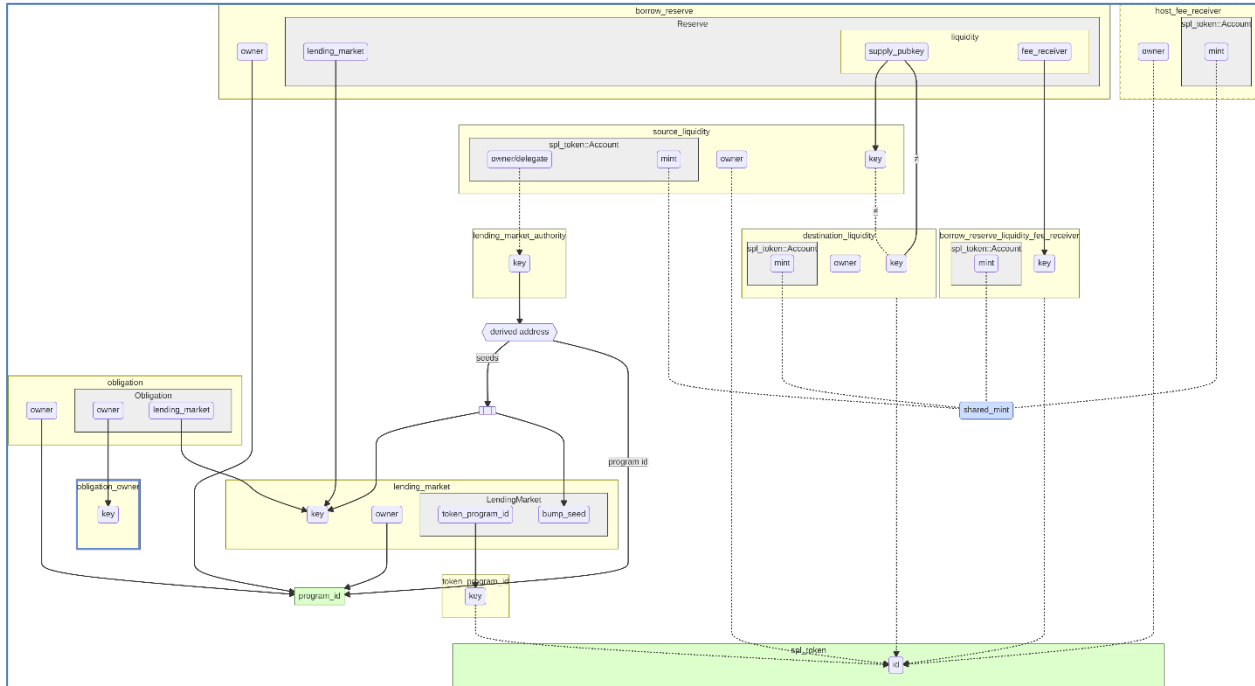


Figure 2: Account reference graph for BorrowObligationLiquidity

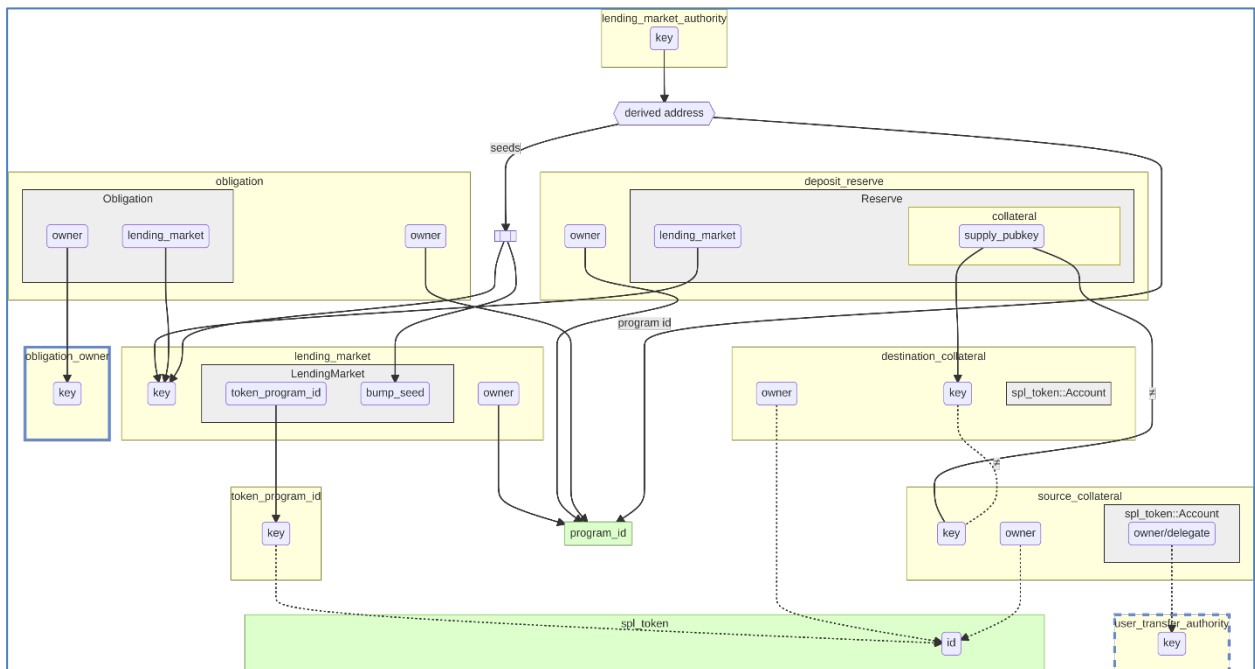


Figure 3: Account reference graph for DepositObligationCollateral



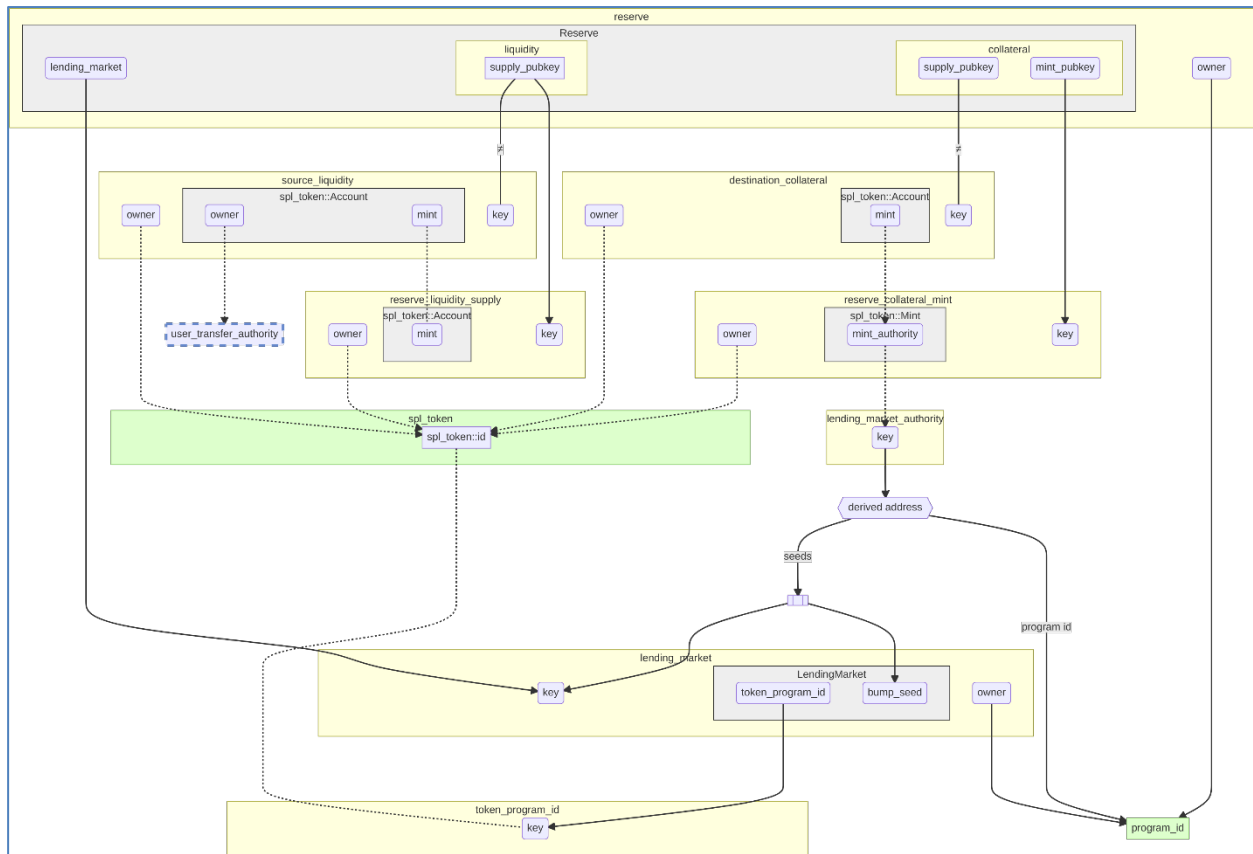


Figure 4: Account reference graph for DepositReserveLiquidity

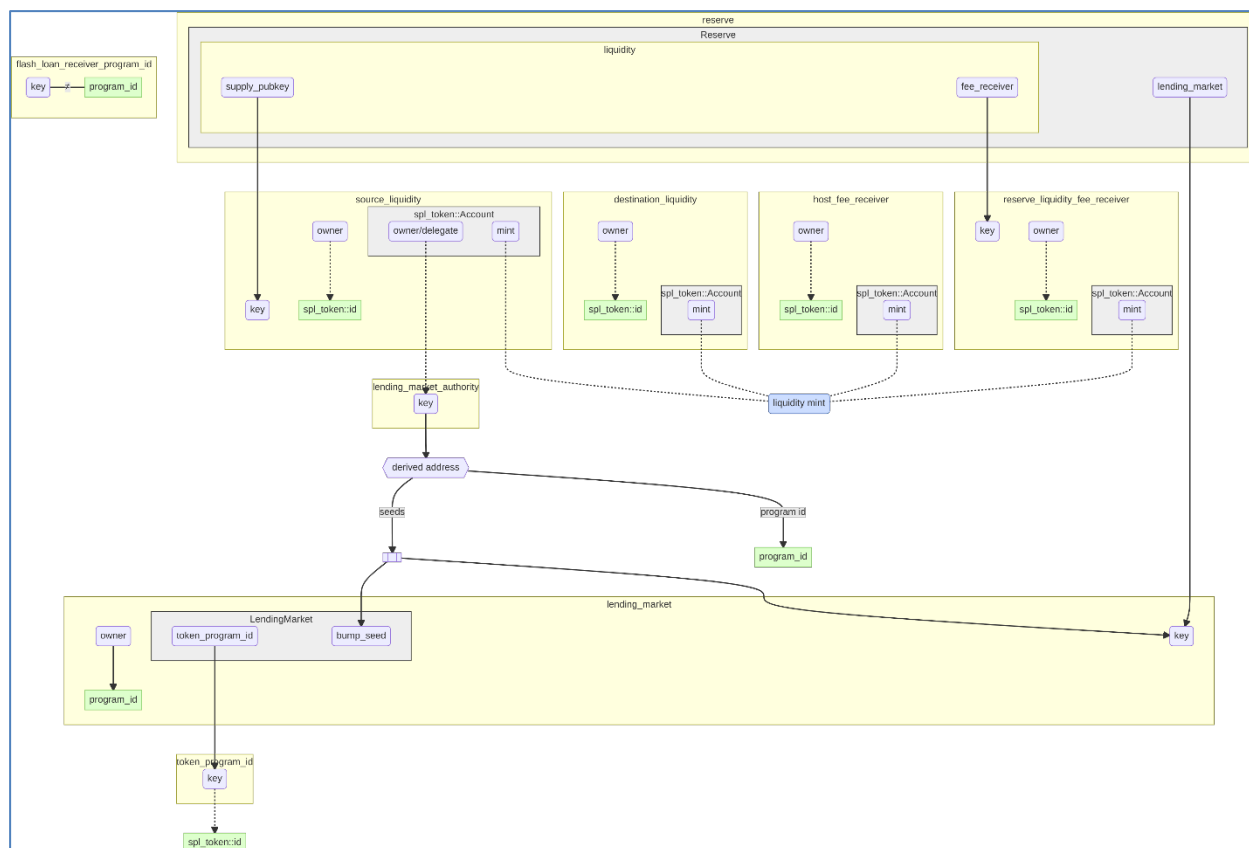


Figure 5: Account reference graph for FlashLoan

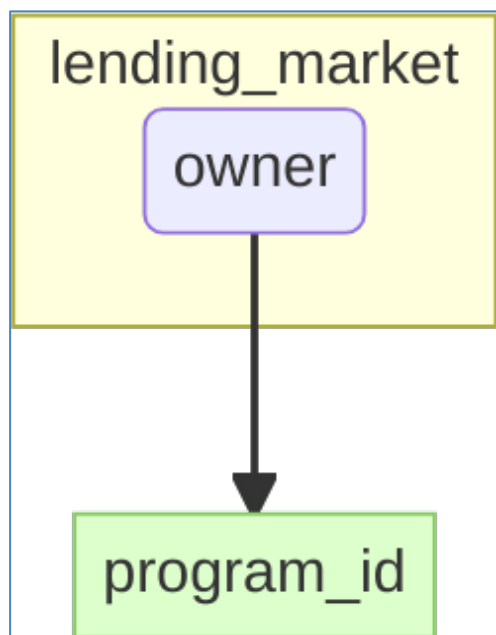
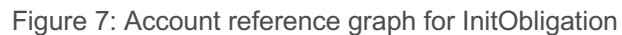


Figure 6: Account reference graph for InitLendingMarket



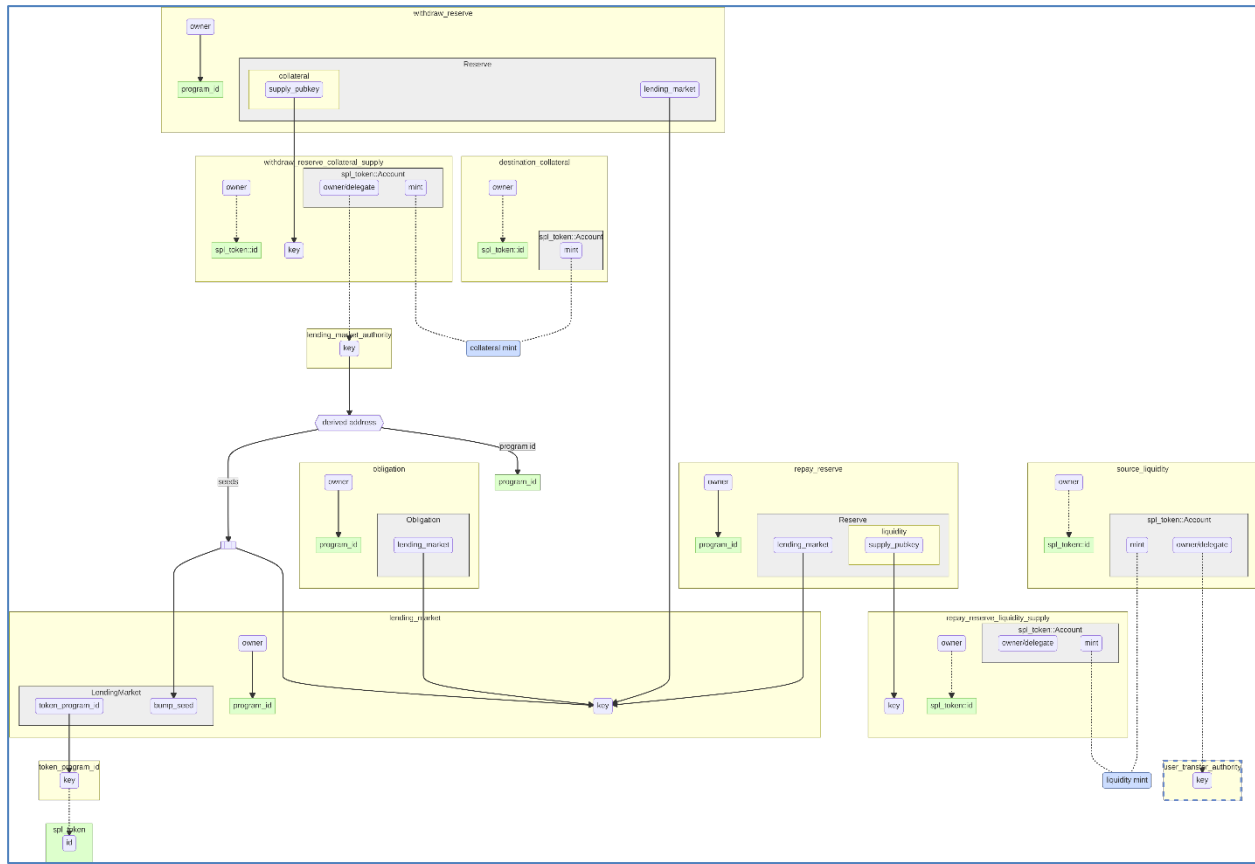


Figure 9: Account reference graph for LiquidateObligation

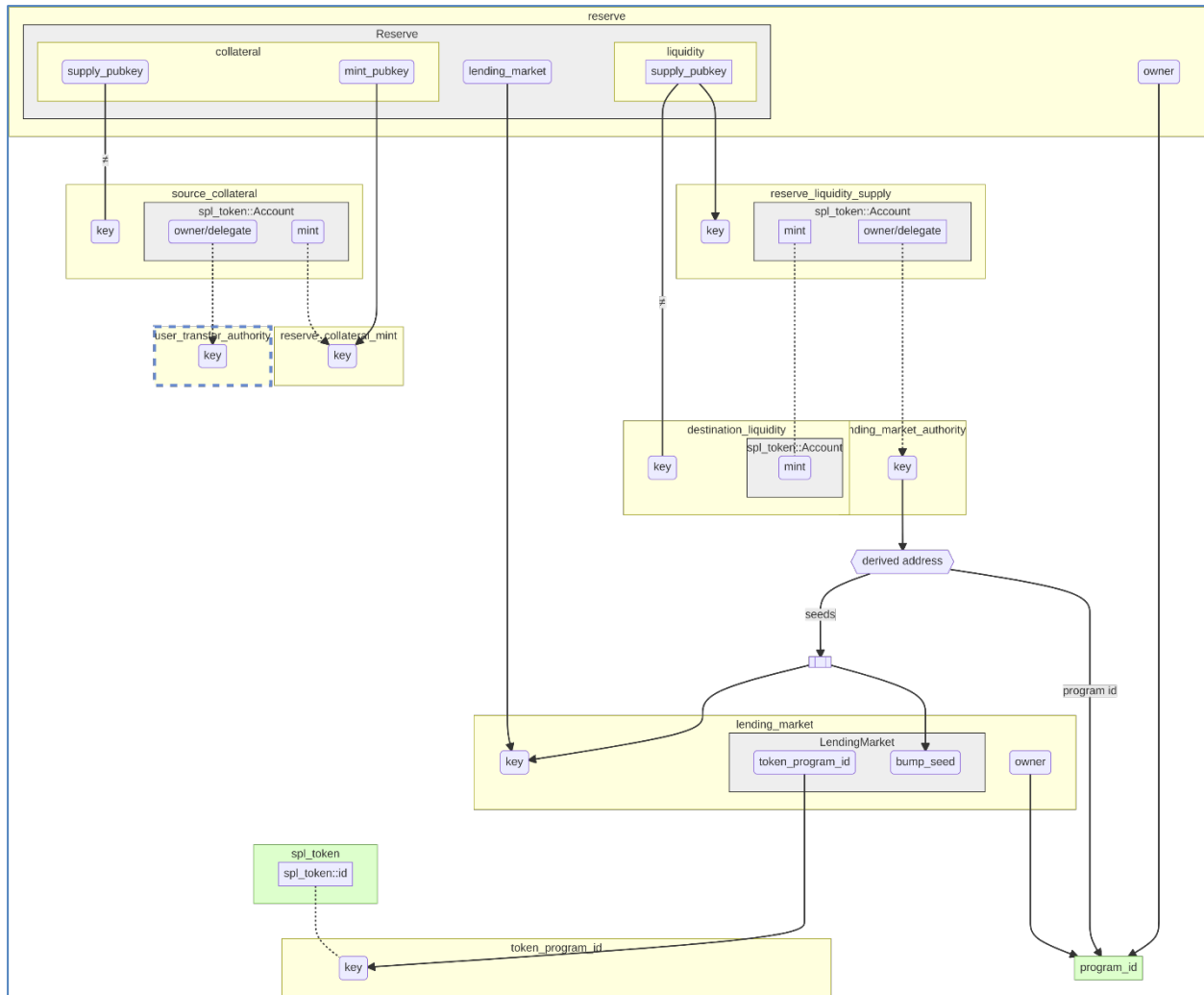


Figure 10: Account reference graph for RedeemReserveCollateral

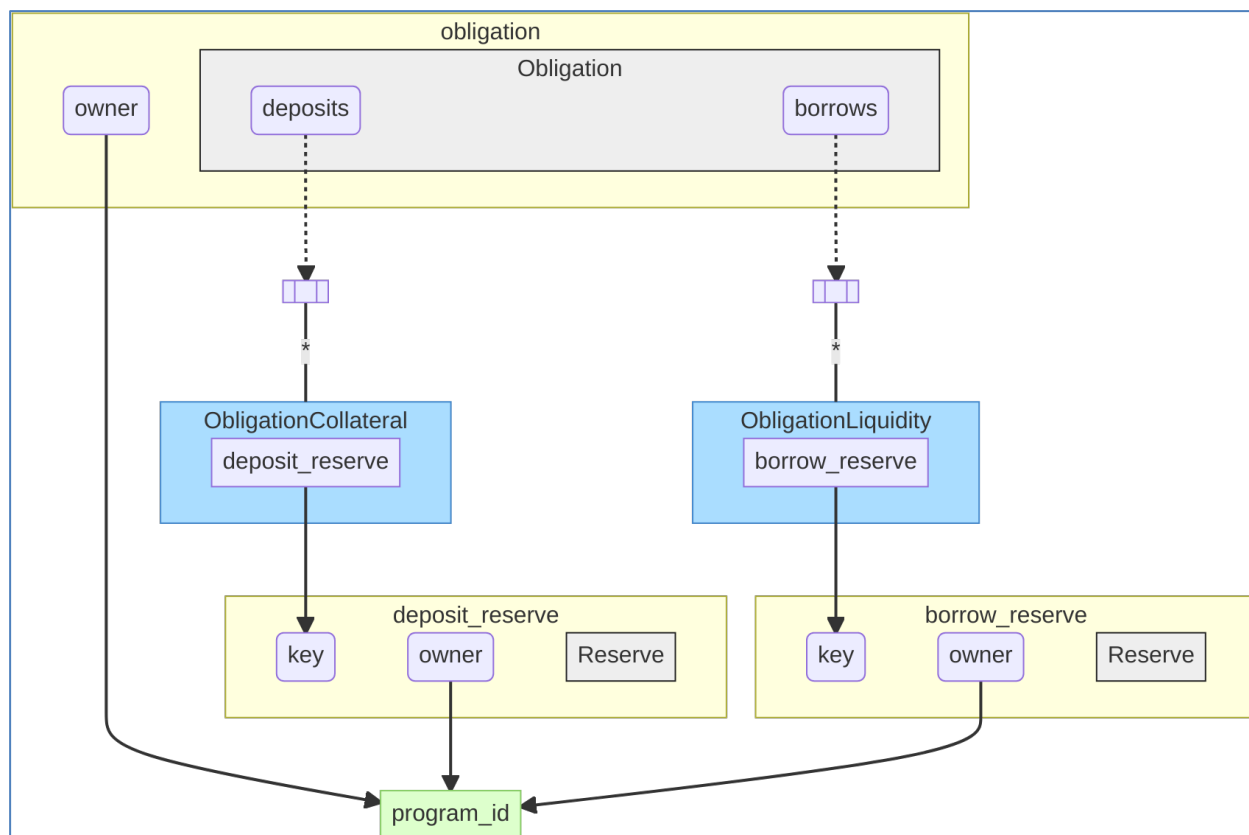


Figure 11: Account reference graph for RefreshObligation

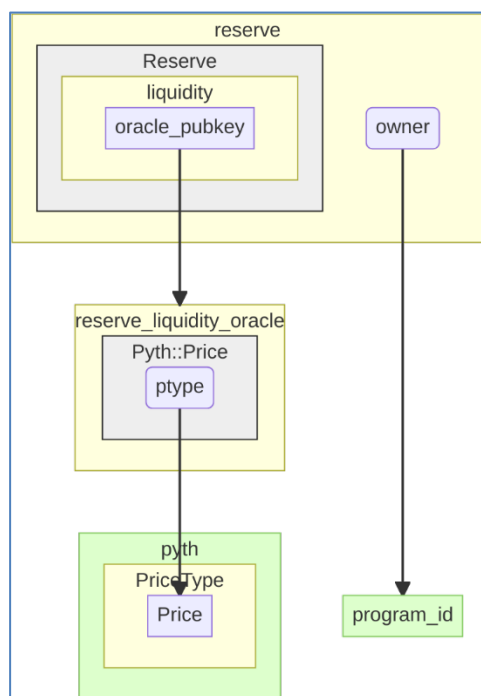


Figure 12: Account reference graph for RefreshReserve

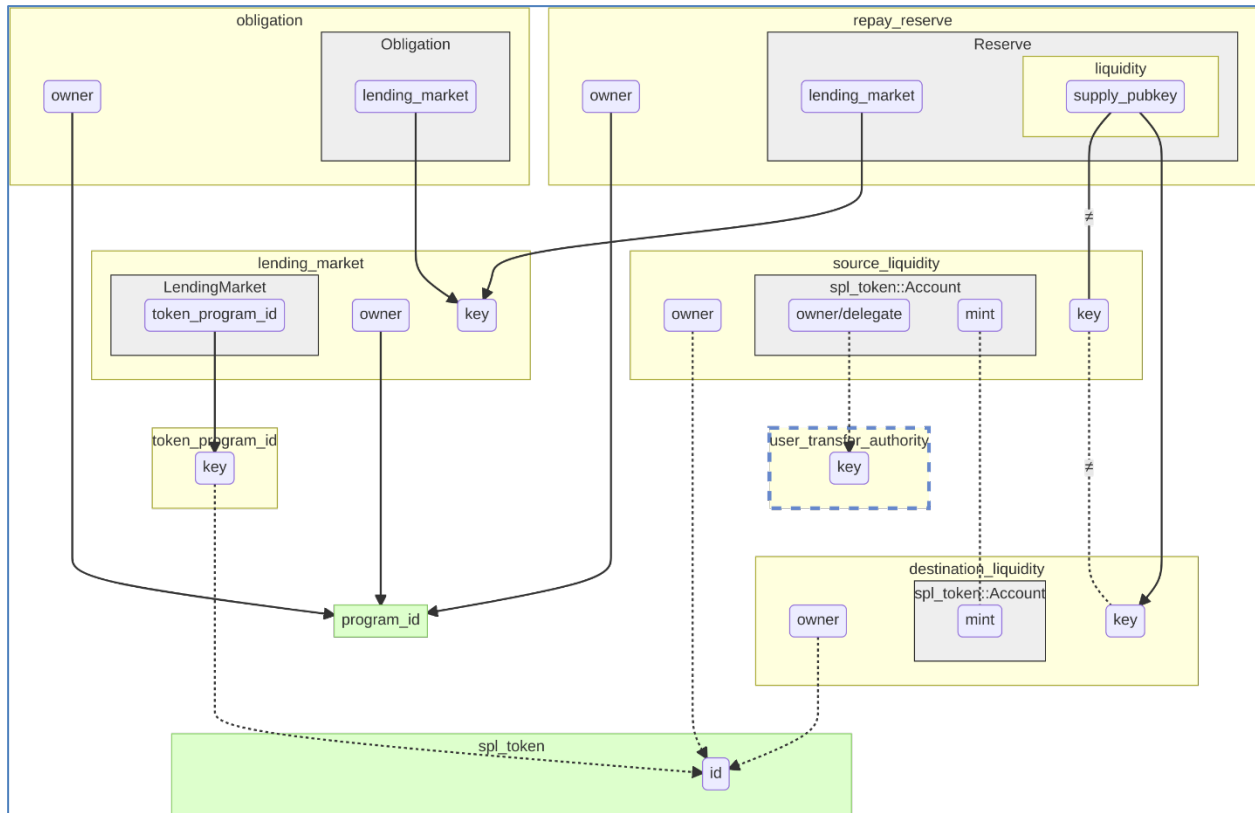


Figure 13: Account reference graph for RepayObligationLiquidity

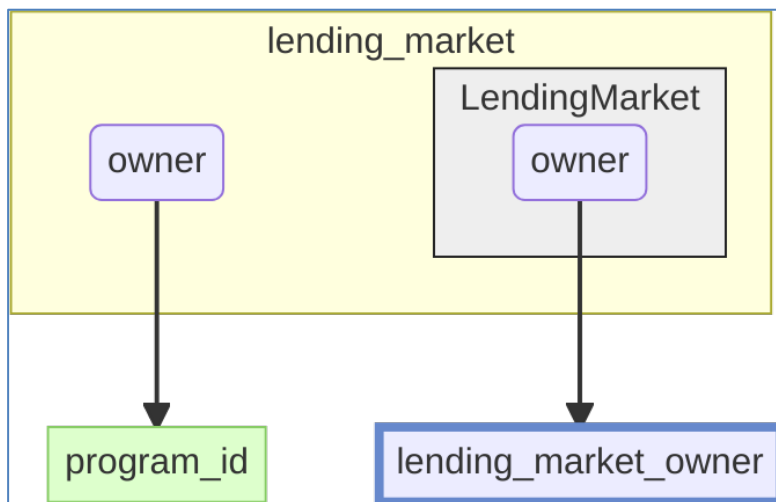


Figure 14: Account reference graph for SetLendingMarketOwner

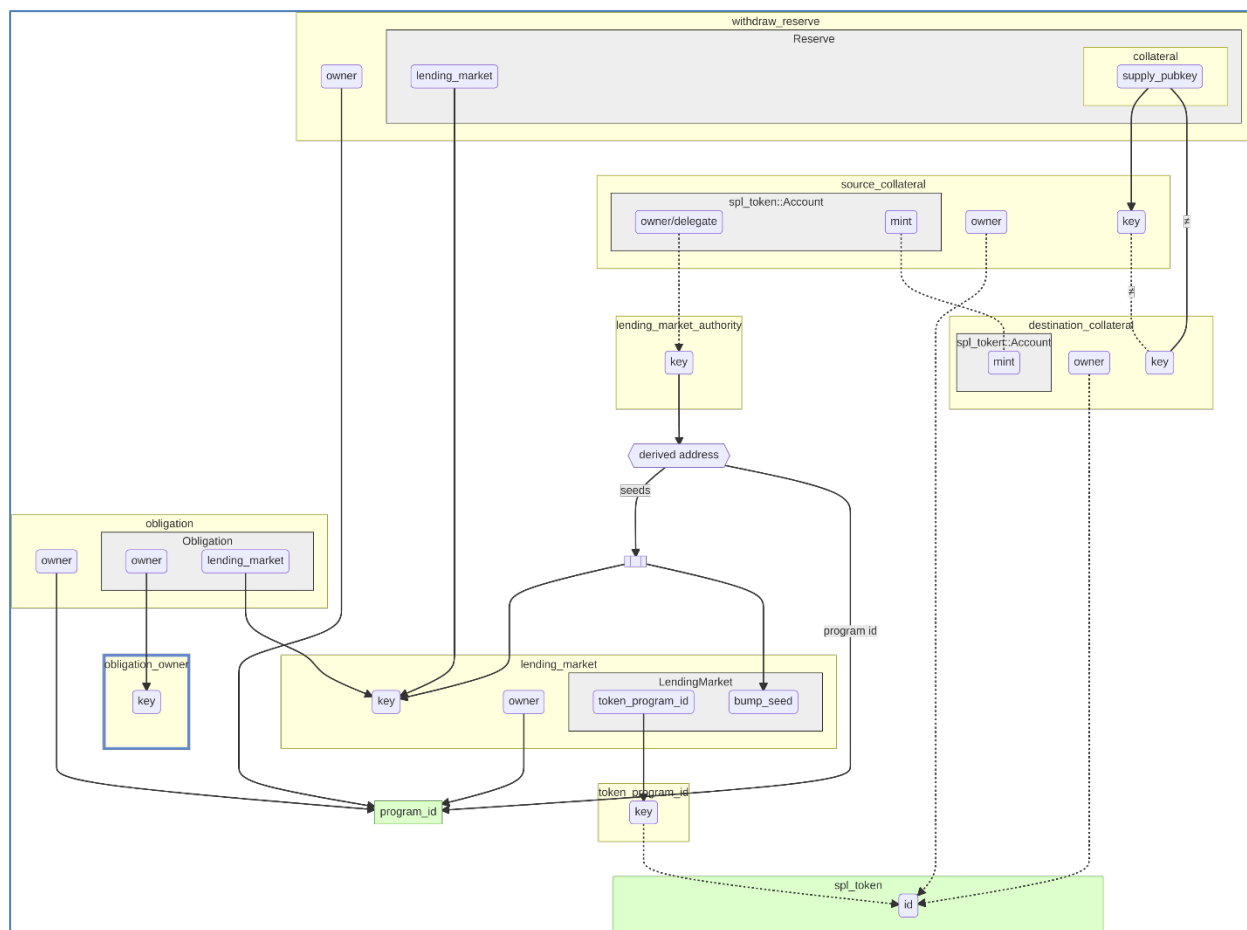


Figure 15: Account reference graph for WithdrawObligationCollateral

A number of further investigations were made which concluded that they did not pose a risk to the application. There were

- Authorization for FlashLoan is soundly validated
- Authorization for LiquidateObligation is soundly validated
- Authorization for RepayObligationLiquidity is soundly validated
- Authorization for BorrowObligationLiquidity is soundly validated
- Authorization for WithdrawObligationCollateral is soundly validated
- Authorization for DepositObligationCollateral is soundly validated
- Authorization for InitObligation is soundly validated
- Authorization for RedeemReserveCollateral is soundly validated
- Authorization for DepositReserveLiquidity is soundly validated



- Authorization for RefreshReserve is soundly validated
- Authorization for InitReserve is soundly validated
- Authorization for SetLendingMarketOwner is soundly validated
- Dependency tokio 1.5.0 is vulnerable according to the RustSec Advisory Database

## Conclusion

Based on the call graphs and the formal verification we can conclude that the code implements the documented functionality to the extent of the code reviewed.

## Technical Findings

### Loss of precision causing miscalculation of interest rate

Finding ID: KS-SOLEND-F-01

Severity: **[High]**

Status: **[Open]**

#### Description

In calculations involving integer division unintended loss of precision may occur if the remainder is not zero and further operations is performed afterwards.

This occurs in the calculation of the constant SLOTS\_PER\_YEAR

#### Proof of Issue

Filename: state/mod.rs

Beginning Line Number: 33

```
pub const SLOTS_PER_YEAR: u64 =
    DEFAULT_TICKS_PER_SECOND / DEFAULT_TICKS_PER_SLOT * SECONDS_PER_DAY * 365;
```

The constants used to calculate the value of SLOTS\_PER\_YEAR are defined in solana\_program::clock as follows

```
pub const DEFAULT_TICKS_PER_SECOND: u64 = 160;
pub const DEFAULT_TICKS_PER_SLOT: u64 = 64;
pub const SECONDS_PER_DAY: u64 = 24 * 60 * 60;
```

Thus the implementation evaluates SLOTS\_PER\_YEAR as follows

```
DEFAULT_TICKS_PER_SECOND / DEFAULT_TICKS_PER_SLOT * SECONDS_PER_DAY * 365
= 160 / 64 * 24 * 60 * 60 * 365
= 2 * 24 * 60 * 60 * 365
= 63_072_000
```

The loss of precision lies in the evaluation of  $160 / 64$  which will result in the integer value 2 and not the decimal number 2.5.

SLOTS\_PER\_YEAR is used as follow

```
SLOTS_PER_YEAR (state/mod.rs:33)
at ReserveLiquidity::compound_interest (state/reserve.rs:456)
at ReserveLiquidity::accrue_interest (state/reserve.rs:133)
at process_refresh_reserve (processor.rs:427)
at process_instruction (processor.rs:60)
at process_instruction (entrypoint.rs:12)
```

The call from processor::process\_instruction line 60 is in the match statement for processing the RefreshReserve instruction which accrues the interest on a reserve.

With the current implementation the calculated interest will be 20% lower than expected.

### **Severity and Impact Summary**

The `SLOTS_PER_YEAR` constant is used to calculate interest rates. As the constant is exposed to a serious loss of precision all calculated interests will be off by 20%!

This is a programming error with immediate consequences to all interest calculations if put into production.

### **Recommendation**

Fix the calculation of the `SLOTS_PER_YEAR` constant to apply multiplication before division to avoid loss of precision.

Furthermore, it is highly recommended to implement unit tests to validate the output of functions with critical responsibilities such as the calculation of interests.

The desired implementation should be the following

```
pub const SLOTS_PER_YEAR: u64 =  
    DEFAULT_TICKS_PER_SECOND * SECONDS_PER_DAY * 365 / DEFAULT_TICKS_PER_SLOT;
```

which evaluates to

```
DEFAULT_TICKS_PER_SECOND * SECONDS_PER_DAY * 365 / DEFAULT_TICKS_PER_SLOT  
= 160 * 24 * 60 * 60 * 365 / 64  
= 78_840_000
```

### **References**

- N/A

## Structs implementing bytemuck::Pod contains non-Pod fields

Finding ID: KS-SOLEND-F-02

Severity: [Medium]

Status: [Open]

### Description

The trait `bytemuck::Pod` is applied to the types `Price` and `Product`.

### Proof of Issue

Filename: `pyth.rs`

Beginning Line Number: 103

```
#[cfg(target_endian = "little")]
unsafe impl Pod for Price {}
```

Filename: `pyth.rs`

Beginning Line Number: 120

```
#[cfg(target_endian = "little")]
unsafe impl Pod for Product {}
```

The documentation for `bytemuck::Pod` specifies 5 safety requirements:

- The type must be inhabited (eg: no `Infallible`).
- The type must allow any bit pattern (eg: no `bool` or `char`, which have illegal bit patterns).
- The type must not contain any padding bytes, either in the middle or on the end (eg: no `#[repr(C)] struct Foo(u8, u16)`, which has padding in the middle, and also no `#[repr(C)] struct Foo(u16, u8)`, which has padding on the end).
- The type needs to have all fields also be `Pod`.
- The type needs to be `repr(C)` or `repr(transparent)`. In the case of `repr(C)`, the `packed` and `align` `repr` modifiers can be used as long as all other rules end up being followed.

Let's have a look at the implementation of the `Price` struct

Filename: `pyth.rs`

Beginning Line Number: 72

```
#[derive(Copy, Clone)]
#[repr(C)]
pub struct Price {
    pub magic: u32,        // pyth magic number
    pub ver: u32,          // program version
    pub atype: u32,         // account type
    pub size: u32,         // price account size
    pub ptype: PriceType,  // price or calculation type
    pub expo: i32,         // price exponent
    pub num: u32,          // number of component prices
    pub unused: u32,
    pub curr_slot: u64,     // currently accumulating price slot
    pub valid_slot: u64,   // valid slot-time of agg. price
    pub twap: i64,         // time-weighted average price
    pub avol: u64,         // annualized price volatility
    pub drv0: i64,         // space for future derived values
    pub drv1: i64,         // space for future derived values
    pub drv2: i64,         // space for future derived values
    pub drv3: i64,         // space for future derived values
    pub drv4: i64,         // space for future derived values
    pub drv5: i64,         // space for future derived values
    pub prod: AccKey,      // product account key
    pub next: AccKey,      // next Price account in linked list
    pub agg_pub: AccKey,   // quoter who computed last aggregate price
    pub agg: PriceInfo,    // aggregate price info
    pub comp: [PriceComp; 32], // price components one per quoter
}
```

According to the documentation "the type needs to have all fields also be Pod."

The primitive integer types are supported out-of-the-box by the Pod trait. But the custom types AccKey, PriceInfo, PriceType, and [PriceComp; 32] are not!

The same goes for the Product struct

**Filename:** pyth.rs

**Beginning Line Number:** 106

```
#[derive(Copy, Clone)]
#[repr(C)]
pub struct Product {
    pub magic: u32,        // pyth magic number
    pub ver: u32,          // program version
    pub atype: u32,         // account type
    pub size: u32,         // price account size
    pub px_acc: AccKey,     // first price account in list
    pub attr: [u8; PROD_ATTR_SIZE], // key/value pairs of reference attr.
}
```

Here the AccKey type is not a Pod!

**Severity and Impact Summary**

Not following the safety requirements when using unsafe code may result in memory corruption and unexpected behavior of the program.

**Recommendation**

- The types `AccKey`, `PriceInfo`, `PriceType`, and `PriceComp` needs to be `Pods` for `Price` and `Product` to be implemented safely.
- The types `PriceStatus` and `CorpAction` needs to be `Pods` for `PriceInfo`, `PriceComp` and `Price` to be implemented safely.

Furthermore, `bytemock_derive` should be used when implementing the `Pod` and `Zeroable` traits as the derive macro checks the field types. This is highly recommended as the `Pod` and `Zeroable` traits involve unsafe code!

Example:

```
use bytemuck_derive::{Pod, Zeroable};

#[derive(Copy, Clone, Pod, Zeroable)]
#[repr(C)]
pub struct Product {
    pub magic: u32,           // pyth magic number
    pub ver: u32,             // program version
    pub atype: u32,           // account type
    pub size: u32,            // price account size
    pub px_acc: AccKey,       // first price account in list
    pub attr: [u8; PROD_ATTR_SIZE], // key/value pairs of reference attr.
}
```

**References**

- <https://docs.rs/bytemuck/1.5.1/bytemuck/trait.Pod.html>
- [https://docs.rs/bytemuck\\_derive/1.0.1/bytemuck\\_derive/derive.Pod.html](https://docs.rs/bytemuck_derive/1.0.1/bytemuck_derive/derive.Pod.html)

## Pyth product parsing may cause index-out-of-bounds

Finding ID: KS-SOLEND-F-03

Severity: [\[Low\]](#)

Status: [\[Open\]](#)

### Description

The function for extracting the `quote_currency` entry of the key/value pairs in `pyth::Product::attr` may cause index-out-of-bounds.

### Proof of Issue

Filename: processor.rs

Beginning Line Number: 1734

```
fn get_pyth_product_quote_currency(pyth_product: &pyth::Product) -> Result<[u8; 32], ProgramError> {
    const LEN: usize = 14;
    const KEY: &[u8; LEN] = b"quote_currency";

    let mut start = 0;
    while start < pyth::PROD_ATTR_SIZE {
        let mut length = pyth_product.attr[start] as usize;
        start += 1;

        if length == LEN {
            let mut end = start + length;
            if end > pyth::PROD_ATTR_SIZE {
                msg!("Pyth product attribute key length too long");
                return Err(LendingError::InvalidOracleConfig.into());
            }

            let key = &pyth_product.attr[start..end];
            if key == KEY {
                start += length;
                length = pyth_product.attr[start] as usize;
                start += 1;

                end = start + length;
                if length > 32 || end > pyth::PROD_ATTR_SIZE {
                    msg!("Pyth product quote currency value too long");
                    return Err(LendingError::InvalidOracleConfig.into());
                }

                let mut value = [0u8; 32];
                value[0..length].copy_from_slice(&pyth_product.attr[start..end]);
                return Ok(value);
            }
        }

        start += length;
        start += 1 + pyth_product.attr[start] as usize;
    }

    msg!("Pyth product quote currency not found");
    Err(LendingError::InvalidOracleConfig.into())
}
```

The values from the `pyth_product.attr` array itself is used as indices to array without validating if it is out of bounds. Unexpected data will cause index-out-of-bounds resulting in panic

Consider that the `attr` is an array containing only bytes of 230

```
pyth_product.attr = [230_u8; pyth::PROD_ATTR_SIZE];
```

A call to `processor::get_pyth_product_quote_currency` will then go like this

The first iteration of the while loop will go like this

```
const LEN: usize = 14;
const KEY: &[u8; LEN] = b"quote_currency";

let mut start = 0;
while start < pyth::PROD_ATTR_SIZE {           // True as 0 < 464
    let mut length = pyth_product.attr[start] as usize; // length = 230
    start += 1;                                     // start = 1
    if length == LEN {                             // False as 230 != 14
        // ...
    }

    start += length;                               // start = 1 + 230 = 231
    start += 1 + pyth_product.attr[start] as usize; // start = 231 + 1 + 230 = 462
}
```

Then the second iteration of the while loop will go like this

```
while start < pyth::PROD_ATTR_SIZE {           // True as 462 < 464
    let mut length = pyth_product.attr[start] as usize; // length = 230
    start += 1;                                     // start = 463
    if length == LEN {                             // False as 230 != 14
        // ...
    }

    start += length;                               // start = 463 + 230 = 693
    start += 1 + pyth_product.attr[start] as usize; // Panic!
}
```

As `pyth_product.attr` has length 464 indexing into 693 will cause panic due to array-out-of-bounds.

### **Severity and Impact Summary**

The values from the `pyth_product.attr` array itself is used as indices to array without validating if it is out of bounds. Unexpected data will cause index-out-of-bounds resulting in panic

### **Recommendation**

Check array indexes or use `std::slice::get` to avoid panic due to array-out-of-bounds.

### **References**

- <https://doc.rust-lang.org/std/primitive.slice.html#method.get>





## Low test coverage creates a risk for maintenance

Finding ID: KS-SOLEND-F-04

Severity: [Informational]

Status: [Open]

### Description

Only 405 of 2845 lines of code or 14.2% is covered by tests

### Proof of Issue

#### LCOV - code coverage report

Current view: top level		Hit	Total	Coverage
Test: token-lending.lcov	Lines:	22316	243338	9.2 %
Date: 2021-07-12 12:08:34	Functions:	0	0	-

Directory	Line Coverage	Functions
/home/sgm/Projects/btblock/solend/solana-program-library/token-lending/program/src	0.3 % 5 / 1738	0 / 0
/home/sgm/Projects/btblock/solend/solana-program-library/token-lending/program/src/math	74.4 % 157 / 211	0 / 0
/home/sgm/Projects/btblock/solend/solana-program-library/token-lending/program/src/state	27.0 % 243 / 899	0 / 0
/home/sgm/Projects/btblock/solend/solana-program-library/token-lending/program/tests	100.0 % 15 / 15	0 / 0
/home/sgm/Projects/btblock/solend/solana-program-library/token/program/src	0.0 % 0 / 1081	0 / 0

### Severity and Impact Summary

The low test coverage is far from convincing when it comes to verifying the behavior of the program.

The low test coverage may introduce security issues in the long run as it makes the code unmaintainable. Thus, the maintainers will not be able to rely on the intention of the code or be sure that even simple updates do not break functionality. And if updates cannot be implemented reliably then it becomes a high security concern.

### Recommendation

Implement both unit tests and integration tests to verify the intention of the code.

This will make a base for regression testing ensuring that functionality is not been broken when future updates such as security patches are implemented.

### References

- N/A

## Disabled lint checks may introduce bad code practices

Finding ID: KS-SOLEND-F-05

Severity: [Informational]

Status: [Open]

### Description

Lint checks were disabled in the code.

### **Proof of Issue**

The following lint checks are disabled in the code:

- `clippy::assign_op_pattern`
- `clippy::manual_range_contains`
- `clippy::ptr_offset_with_cast`
- `clippy::reversed_empty_ranges`
- `clippy::too_many_arguments`
- `clippy::wrong_self_convention`

As clippy's lint checks warn about bad code practices, ignoring the warnings allows bad practices in the code.

As an example the `clippy::reversed_empty_ranges` lint check verifies the following

Checks for range expressions `x..y` where both `x` and `y` are constant and `x` is greater or equal to `y`.

### Severity and Impact Summary

Disabling the `clippy::reversed_empty_ranges` lint check will allow such range expressions in the code which will not warn on reversed range such as `3..0` which will cause panic at runtime!

### Recommendation

Do not disable lint checks unless you have a really good reason and always document that reason in the code where the check is disabled.

### References

- Clippy Lints: `reversed_empty_ranges`  
[https://rust-lang.github.io/rust-clippy/master/index.html#reversed\\_empty\\_ranges](https://rust-lang.github.io/rust-clippy/master/index.html#reversed_empty_ranges)

## Oracle (Pyth) program id stored in LendingMarket is not validated

Finding ID: KS-SOLEND-F-06

Severity: [Informational]

Status: **[Open]**

### Description

The `LendingMarket` struct contains ids for the Oracle (Pyth) program (`oracle_program_id`)

## Proof of Issue

**Filename:** state/lending\_market.rs

**Beginning Line Number: 11**

```
#[derive(Clone, Debug, Default, PartialEq)]
pub struct LendingMarket {
    /// Version of lending market
    pub version: u8,
    /// Bump seed for derived authority address
    pub bump_seed: u8,
    /// Owner authority which can add new reserves
    pub owner: Pubkey,
    /// Currency market prices are quoted in
    /// e.g. "USD" null padded ("b"USD\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0") or a SPL token mint pubkey
    pub quote_currency: [u8; 32],
    /// Token program id
    pub token_program_id: Pubkey,
    /// Oracle (Pyth) program id
    pub oracle_program_id: Pubkey,
}
```

The `oracle_program_id` is written to the `LendingMarket` during processing of the `InitLendingMarket` instruction. No instructions allows it to be changed after initialization.

The `oracle_program_id` is used to validate the price and product accounts in the `InitReserve` instruction. The price account from is used again in the `RefreshReserve` instruction to update the current market price which again is used in calculations for the `RefreshObligation` and the `BorrowObligationLiquidity` instruction.

### Severity and Impact Summary

As the `oracle_program_id` is not validated, it is possible to initialize a lending market with an oracle program id referring to a program controlled by the someone else.

Off course, this would create a lending market which is untrustworthy. But how can the end-user verify that?

Misbehaving lending markets may have no direct consequence for the `spl-token-lending` program as this is a configuration issue and not an implementation issue. But the result may backfire resulting in mistrust for the whole `spl-token-lending` and all of its lending markets.

**Recommendation**

If it is possible at all, the `oracle_program_id` should be validated against a whitelist of trusted Oracle (Pyth) programs. If implemented it should be considered how to handle lending markets based on Oracle programs that have been removed from the whitelist.

**References**

- N/A

## SPL Token program id stored in LendingMarket is superfluous

Finding ID: KS-SOLEND-F-07

Severity: [Informational]

Status: **[Open]**

### Description

The `LendingMarket` struct contains ids for the SPL Token program (`token_program_id`).

## Proof of Issue

**Filename:** state/lending\_market.rs

**Beginning Line Number: 11**

```
#[derive(Clone, Debug, Default, PartialEq)]
pub struct LendingMarket {
    /// Version of lending market
    pub version: u8,
    /// Bump seed for derived authority address
    pub bump_seed: u8,
    /// Owner authority which can add new reserves
    pub owner: Pubkey,
    /// Currency market prices are quoted in
    /// e.g. "USD" null padded ("*b"USD\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0") or a SPL token mint pubkey
    pub quote_currency: [u8; 32],
    /// Token program id
    pub token_program_id: Pubkey,
    /// Oracle (Pyth) program id
    pub oracle_program_id: Pubkey,
}
```

The `token_program_id` is written to the `LendingMarket` during processing of the `InitLendingMarket` instruction. No instructions allows it to be changed after initialization.

The `token_program_id` is used to validate the SPL Token program account given as input to the following instructions:

- InitReserve
- DepositReserveLiquidity
- RedeemReserveCollateral
- InitObligation
- DepositObligationCollateral
- WithdrawObligationCollateral
- BorrowObligationLiquidity
- RepayObligationLiquidity
- LiquidateObligation
- FlashLoan

In the processing of the `InitReserve` instruction the `token_program_id` account input is validated against the SPL Token program id from the lending market

**Filename:** processor.rs

**Beginning Line Number: 252**

```
if &lending_market.token_program_id != token_program_id.key {
    msg!("Lending market token program does not match the token program provided");
    return Err(LendingError::InvalidTokenProgram.into());
}
```

Furthermore, the owner of the reserve liquidity mint is also validated against the SPL Token program id from the lending market

**Filename:** processor.rs

**Beginning Line Number: 252**

```
if reserve_liquidity_mint_info.owner != token_program_id.key {
    msg!("Reserve liquidity mint is not owned by the token program provided");
    return Err(LendingError::InvalidTokenOwner.into());
}
```

But then the function `spl_token_init_account` is called

**Filename:** processor.rs

**Beginning Line Number: 348**

```
spl_token_init_account(TokenInitializeAccountParams {
    account: reserve_liquidity_supply_info.clone(),
    mint: reserve_liquidity_mint_info.clone(),
    owner: lending_market_authority_info.clone(),
    rent: rent_info.clone(),
    token_program: token_program_id.clone(),
})?;
```

The `spl_token_init_account` inline function creates an instruction by calling the constructor function `spl_token::instruction::initialize_account`. Here, `token_program_id` account key is passed as `token_program` argument to `initialize_account`

**Filename:** processor.rs

**Beginning Line Number: 1828**

```
#[inline(always)]
fn spl_token_init_account(params: TokenInitializeAccountParams<'_>) -> ProgramResult {
    let TokenInitializeAccountParams {
        account,
        mint,
        owner,
        rent,
        token_program,
    } = params;
    let ix = spl_token::instruction::initialize_account(
        token_program.key,
        account.key,
        mint.key,
        owner.key,
    )?;
    let result = invoke(&ix, &[account, mint, owner, rent, token_program]);
    result.map_err(|_| LendingError::TokenInitializeAccountFailed.into())
}
```

The `spl_token::instruction::initialize_account` function (in the `spl-token` crate) calls the `spl_token::check_program_account` function

**Filename:** token/program/src/instruction.rs

**Beginning Line Number:** 645

```
pub fn initialize_account(  
    token_program_id: &Pubkey,  
    account_pubkey: &Pubkey,  
    mint_pubkey: &Pubkey,  
    owner_pubkey: &Pubkey,  
) -> Result<Instruction, ProgramError> {  
    check_program_account(token_program_id?);  
}
```

Finally, the `spl_token::check_program_account` function validated that the key of the `token_program_id` account given as input to the `InitReserve` instruction is `spl_token::id()`

**Filename:** token/program/src/lib.rs

**Beginning Line Number:** 30

```
solana_program::declare_id!("TokenkegQfeZYiNhWAjnbGKPFXCWuBvrf9Ss623VQ5DA");  
  
/// Checks that the supplied program ID is the correct one for SPL-token  
pub fn check_program_account(spl_token_program_id: &Pubkey) -> ProgramResult {  
    if spl_token_program_id != &id() {  
        return Err(ProgramError::IncorrectProgramId);  
    }  
    Ok(())  
}
```

The validations done during the instruction processing are sane and must take place one way or the other. But storing the SPL Token program id as part of the lending market account is superfluous as the call from the processing of the `InitReserve` instruction to `spl_token::instruction::initialize_account` requires the program id to be `spl_token::id()`.

So even though `LendingMarket::token_program_id` can be refer to another program than the official SPL Token program, it will not be possible to initialize a reserve for such a lending market...

### **Severity and Impact Summary**

`LendingMarket::token_program_id` can be refer to another program than the official SPL Token program, it will not be possible to initialize a reserve for such a lending market.

### **Recommendation**

To simplify the code and the instruction arguments it is recommended to remove the `token_program_id` from the `LendingMarket` struct and update all validations to check against the `spl_token::ID` constant instead.

### **References**

- N/A



## METHODOLOGY

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

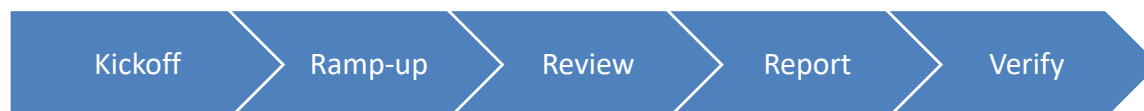


Figure 16: Methodology Flow

### Kickoff

The project is kicked all of 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 particular 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 a majority 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 list and not comprehensive, meant only to give an understanding of the issues we are looking for.

## Cryptography

We analyzed the cryptographic primitives and components as well as their implementation. We checked in particular:

- Matching of the proper cryptographic primitives to the desired cryptographic functionality needed
- Security level of cryptographic primitives and their respective parameters (key lengths, etc.)
- Safety of the randomness generation in general as well as in the case of failure
- Safety of key management
- Assessment of proper security definitions and compliance to use cases
- Checking for known vulnerabilities in the primitives used

## 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

Kudelski Security delivers a preliminary report in PDF format 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 as a whole. 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 not only report security issues identified but also informational findings for improvement categorized into several buckets:

- 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. These are a solid baseline for severity determination.

## The Classification of identified problems and vulnerabilities

There are four severity levels of an identified security vulnerability.

### **Critical – vulnerability that will lead to 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 - A vulnerability that can lead to loss of protected assets**

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

### **Medium - a vulnerability that 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 write sensitive data to log files

### **Low - Problems that have a security impact but does not directly impact the protected assets**

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

### **Informational**

- General recommendations

## TOOLS

The following tools were used during this portion of the test. A link for more information about the tool is provided as well.

Tools used during the code review and assessment

- Rust – cargo tools
- IDE modules for Rust and analysis of source code
- Cargo audit which uses <https://rustsec.org/advisories/> to find vulnerabilities cargo.

### RustSec.org

#### About RustSec

The RustSec Advisory Database is a repository of security advisories filed against Rust crates published and maintained by the Rust Secure Code Working Group.

#### The RustSec Tool-set used in projects and CI/CD pipelines

‘cargo-audit’ - audit Cargo.lock files for crates with security vulnerabilities.

‘cargo-deny’ - audit ‘Cargo.lock’ files for crates with security vulnerabilities, limit the usage of particular dependencies, their licenses, sources to download from, detect multiple versions of same packages in the dependency tree and more.



## KUDELSKI SECURITY CONTACTS

NAME	POSITION	CONTACT INFORMATION
Mikael Björn	Technical Lead	<a href="mailto:mikael.bjorn@kudelskisecurity.com">mikael.bjorn@kudelskisecurity.com</a>