



Solana Token-2022 Program

Security Assessment (Summary Report)

October 25, 2022

Prepared for:

Sam Kim

Solana

Prepared by: **Anders Helsing**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at <https://github.com/trailofbits/publications>, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow [@trailofbits](#) on Twitter and explore our public repositories at <https://github.com/trailofbits>. To engage us directly, visit our "Contact" page at <https://www.trailofbits.com/contact>, or email us at info@trailofbits.com.

Trail of Bits, Inc.

228 Park Ave S #80688

New York, NY 10003

<https://www.trailofbits.com>

info@trailofbits.com

Notices and Remarks

Copyright and Distribution

© 2022 by Trail of Bits, Inc.

All rights reserved. Trail of Bits hereby asserts its right to be identified as the creator of this report in the United Kingdom.

This report is considered by Trail of Bits to be public information; it is licensed to Solana under the terms of the project statement of work and has been made public at Solana's request. Material within this report may not be reproduced or distributed in part or in whole without the express written permission of Trail of Bits.

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

About Trail of Bits	1
Notices and Remarks	2
Table of Contents	3
Executive Summary	5
Project Summary	6
Project Goals	7
Project Targets	8
Project Coverage	9
Summary of Findings	10
Detailed Findings	11
1. Ok returned for malformed extension data	11
2. Missing account ownership checks	12
3. Use of a vulnerable dependency	13
4. Large extension sizes can cause panics	14
5. Unexpected function behavior	15
6. Out of bounds access in the get_extension instruction	16
7. Iteration over empty data	17
8. Missing check in UpdateMint instruction could result in inoperable mints	18
9. Incorrect test data description	20
10. The Transfer and TransferWithFee instructions are identical	21
11. Some instructions operate only on the lo bits of balances	23
12. Instruction susceptible to front-running	25

A. Vulnerability Categories	26
B. Non-Security-Related Findings	28
C. Fix Review Results	29
Detailed Fix Review Results	31

Executive Summary

Engagement Overview

Solana engaged Trail of Bits to review the security of its Token-2022 Program. From September 26 to September 30, 2022, one consultant conducted a security review of the client-provided source code, with one person-week of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

Project Scope

Our testing efforts were focused on the identification of flaws that could result in a compromise of confidentiality, integrity, or availability of the target system. We conducted this audit with full knowledge of the system, including access to the source code and documentation. We performed static and dynamic testing of the target system and its codebase, using both automated and manual processes.

Summary of Findings

The audit uncovered flaws that could impact system confidentiality, integrity, or availability. A summary of the findings is provided below.

EXPOSURE ANALYSIS

<i>Severity</i>	<i>Count</i>
High	0
Medium	0
Low	2
Informational	7
Undetermined	3

CATEGORY BREAKDOWN

<i>Category</i>	<i>Count</i>
Access Controls	1
Data Validation	1
Denial of Service	1
Patching	1
Testing	1
Undefined Behavior	7

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

Jeff Braswell, Project Manager
jeff.braswell@trailofbits.com

The following engineer was associated with this project:

Anders Helsing, Consultant
anders.helsing@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
September 22, 2022	Pre-project kickoff call
October 3, 2022	Delivery of report draft
October 3, 2022	Report readout meeting
October 25, 2022	Delivery of final report
January 26, 2023	Delivery of fix review addendum

Project Goals

The engagement was scoped to provide a security assessment of the Solana Token-2022 Program. Specifically, we sought to answer the following non-exhaustive list of questions:

- Does the system securely store extensions in type-length-value (TLV) records?
- Is it possible to bypass the checks of accounts used by instructions?
- Could instructions use the wrong types of accounts?
- Could tokens become lost or frozen?

Project Targets

The engagement involved a review and testing of the following target.

Solana

Repository	<code>solana-labs/solana-program-library/token/program-2022</code>
Version	<code>50abadd819df2e406567d6eca31c213264c1c7cd</code>
Type	Rust
Platform	Solana

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- We ran the static analysis tools `cargo-audit`, `cargo-outdated`, and Clippy over the codebase. Specifically, we ran `cargo-audit` and `cargo-outdated` over the `Cargo.lock` file and Clippy over all of the Rust source files. We reviewed the results of each run.
- We manually reviewed the load/store mechanism used for storing extensions in TLV records.
- We manually reviewed the confidential transfers extension.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. During this project, we were unable to perform comprehensive testing of the following system elements, which may warrant further review:

- The updates to the legacy Token Program instructions
- The implementations of the instructions for the following extensions:
 - Transfer fees
 - Closing mint
 - Interest-bearing tokens
 - Non-transferable tokens
- The implementations of the following account extensions:
 - Memo required on incoming transfers
 - Immutable ownership
 - Default account state

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Ok returned for malformed extension data	Data Validation	Informational
2	Missing account ownership checks	Access Controls	Undetermined
3	Use of a vulnerable dependency	Patching	Undetermined
4	Large extension sizes can cause panics	Undefined Behavior	Informational
5	Unexpected function behavior	Undefined Behavior	Informational
6	Out of bounds access in the get_extension instruction	Undefined Behavior	Low
7	Iteration over empty data	Undefined Behavior	Informational
8	Missing check in UpdateMint instruction could result in inoperable mints	Denial of Service	Low
9	Incorrect test data description	Testing	Informational
10	The Transfer and TransferWithFee instructions are identical	Undefined Behavior	Informational
11	Some instructions operate only on the lo bits of balances	Undefined Behavior	Undetermined
12	Instruction susceptible to front-running	Undefined Behavior	Informational

Detailed Findings

1. Ok returned for malformed extension data

Severity: Informational

Difficulty: Low

Type: Data Validation

Finding ID: TOB-STK-1

Target: token/program-2022/src/extension/mod.rs

Description

In the `get_extension_types` function, if the account type-length-value (TLV) data is malformed and the TLV record data is truncated (i.e., the account data length is less than the start offset summed with the length of the TLV data), the function returns `Ok` rather than an error.

```
fn get_extension_types(tlv_data: &[u8]) -> Result<Vec<ExtensionType>, ProgramError>
{
    let mut extension_types = vec![];
    let mut start_index = 0;
    while start_index < tlv_data.len() {
        let tlv_indices = get_tlv_indices(start_index);
        if tlv_data.len() < tlv_indices.value_start {
            return Ok(extension_types);
        }
    }
}
```

Figure 1.1:

<https://github.com/solana-labs/solana-program-library/token/program-2022/src/extension/mod.rs#L127-L134>

Recommendations

Short term, modify the `get_extension_types` function so that it returns an error if the TLV data is corrupt. This will ensure that the Token Program will not continue processing if the provided account's extension data is corrupt.

2. Missing account ownership checks

Severity: **Undetermined**

Difficulty: **Medium**

Type: Access Controls

Finding ID: TOB-STK-2

Target: Various instructions

Description

Every account that the Token Program operates on should be owned by the Token Program, but several instructions lack account ownership checks. The functions lacking checks include `process_reallocate`, `process_withdraw_withheld_tokens_from_mint`, and `process_withdraw_withheld_tokens_from_accounts`. Many of these functions have an implicit check for this condition in that they modify the account data, which is possible only if the account is owned by the Token Program; however, future changes to the associated code could remove this protection.

For example, in the `process_withdraw_withheld_tokens_from_accounts` instruction, neither the `mint_account_info` nor `destination_account_info` parameter is checked to ensure the account is owned by the Token Program. While the mint account's data is mutably borrowed, the account data is never written. As a result, an attacker could pass a forged account in place of the mint account. Conversely, the `destination_account_info` account's data is updated by the instruction, so it must be owned by the Token Program. However, if an attacker can find a way to spoof an account public key that matches the mint property in the destination account, he could bypass the implicit check.

Recommendations

Short term, as a defense-in-depth measure, add explicit checks of account ownership for all accounts passed to instructions. This will both improve the clarity of the codebase and remove the dependence on implicit checks, which may no longer hold true when updates occur.

3. Use of a vulnerable dependency

Severity: **Undetermined**

Difficulty: **Low**

Type: Patching

Finding ID: TOB-STK-3

Target: Various files

Description

Running the `cargo audit` command uncovered the use of one crate with a known vulnerability (`time`).

```
> cargo audit
  Fetching advisory database from `https://github.com/RustSec/advisory-db.git`
    Loaded 458 security advisories (from /Users/andershelsing/.cargo/advisory-db)
    Updating crates.io index
  Scanning Cargo.lock for vulnerabilities (651 crate dependencies)
Crate:      time
Version:    0.1.44
Title:      Potential segfault in the time crate
Date:       2020-11-18
ID:         RUSTSEC-2020-0071
URL:        https://rustsec.org/advisories/RUSTSEC-2020-0071
Solution:   Upgrade to >=0.2.23
```

Figure 3.1: The result of running the `cargo audit` command

Recommendations

Short term, triage the use of the vulnerability in the `time` crate and upgrade the crate to a version in which the vulnerability is patched.

4. Large extension sizes can cause panics

Severity: Informational

Difficulty: Low

Type: Undefined Behavior

Finding ID: TOB-STK-4

Target: token/program-2022/src/extension/mod.rs

Description

The call to `try_from` in the `init_extension` function returns an error if the `length` of the given extension is larger than `u16::Max`, which causes the `unwrap` operation to panic.

```
let length = pod_get_packed_len::<V>();  
*length_ref = Length::try_from(length).unwrap();
```

Figure 4.1:

<https://github.com/solana-labs/solana-program-library/token/program-2022/src/extension/mod.rs#L493-L494>

Recommendations

Short term, add assertions to the program to catch extensions whose sizes are too large, and add relevant code to handle errors that could arise in the `try_from` function. This will ensure that the Token Program does not panic if any extension grows larger than `u16::Max`.

5. Unexpected function behavior

Severity: Informational

Difficulty: Low

Type: Undefined Behavior

Finding ID: TOB-STK-5

Target: token/program-2022/src/instruction.rs

Description

The `decode_instruction_data` function receives a byte slice representing the instruction data. Specifically, the function expects the first byte of the slice to contain the instruction type for an extension instruction; however, the function name does not clearly convey this intended behavior, and the behavior is not explained in code comments.

```
/// Utility function for decoding instruction data
pub fn decode_instruction_data<T: Pod>(input: &[u8]) -> Result<&T, ProgramError> {
    if input.len() != pod_get_packed_len:::<T>().saturating_add(1) {
        Err(ProgramError::InvalidInstructionData)
    } else {
        pod_from_bytes(&input[1..])
    }
}
```

Figure 5.1:

<https://github.com/solana-labs/solana-program-library/token/program-2022/src/instruction.rs#L1761-L1768>

Recommendations

Short term, change the `decode_instruction_data` function so that it operates only on instruction data, and remove the instruction type from the data passed prior to the call. This will ensure that the function's name is in line with the function's operation.

6. Out of bounds access in the get_extension instruction

Severity: Low

Difficulty: Low

Type: Undefined Behavior

Finding ID: TOB-STK-6

Target: token/program-2022/src/extension/mod.rs

Description

The `get_extension` function instantiates a type from a TLV record. However, the `get_extension_indices` function does not check that the account's data length is large enough for the `value_end` index.

```
fn get_extension<S: BaseState, V: Extension>(tlv_data: &[u8]) -> Result<&V,
ProgramError> {
    if V::TYPE.get_account_type() != S::ACCOUNT_TYPE {
        return Err(ProgramError::InvalidAccountData);
    }
    let TlvIndices {
        type_start: _,
        length_start,
        value_start,
    } = get_extension_indices::<V>(tlv_data, false)?;
    // get_extension_indices has checked that tlv_data is long enough to include
    these indices
    let length = pod_from_bytes::<Length>(&tlv_data[length_start..value_start])?;
    let value_end = value_start.saturating_add(usize::from(*length));
    pod_from_bytes::<V>(&tlv_data[value_start..value_end])
}
```

Figure 6.1:

<https://github.com/solana-labs/solana-program-library/token/program-2022/src/extension/mod.rs#L235-L248>

Recommendations

Short term, add a check to ensure that the TLV data for the account is large enough for the `value_end` index, and add relevant code to handle the error if it is not. This will ensure that the Token Program will not panic on accounts with truncated data.

7. Iteration over empty data

Severity: Informational

Difficulty: Low

Type: Undefined Behavior

Finding ID: TOB-STK-7

Target: token/program-2022/src/extension/mod.rs

Description

The `get_extension_indices` function returns either the indices for a given extension type or the first uninitialized slot. Because a TLV data record can never be deleted, the first zero-value entry of the slice should indicate that the iteration has reached the end of the used data space. However, if the `init` parameter is false, the `start_index` index is advanced by two, and the iteration continues, presumably iterating over empty data until it reaches the end of the TLV data for the account.

```
while start_index < tlv_data.len() {
    let tlv_indices = get_tlv_indices(start_index);
    if tlv_data.len() < tlv_indices.value_start {
        return Err(ProgramError::InvalidAccountData);
    }
    ...
    // got to an empty spot, can init here, or move forward if not initing
    if extension_type == ExtensionType::Uninitialized {
        if init {
            return Ok(tlv_indices);
        } else {
            start_index = tlv_indices.length_start;
        }
    }
    ...
}
```

Figure 7.1:

<https://github.com/solana-labs/solana-program-library/token/program-2022/src/extension/mod.rs#L96-L122>

Recommendations

Short term, modify the associated code so that it terminates the iteration when it reaches uninitialized data, which should indicate the end of the used TLV record data.

8. Missing check in UpdateMint instruction could result in inoperable mints

Severity: Low

Difficulty: Low

Type: Denial of Service

Finding ID: TOB-STK-8

Target:

token/program-2022/src/extension/confidential_transfer/processor.rs

Description

If a mint's `auto_approve_new_accounts` property is `false`, the `ApproveAccount` instruction needs the mint's authority to sign transactions approving Accounts for the mint. However, issuing an `update_mint` instruction with the new authority set to `Pubkey::default` and the `auto_approve_new_accounts` property set to `false` would prevent Accounts from being approved.

```
/// Processes an [UpdateMint] instruction.
fn process_update_mint(
    accounts: &[AccountInfo],
    new_confidential_transfer_mint: &ConfidentialTransferMint,
) -> ProgramResult {
    let account_info_iter = &mut accounts.iter();
    let mint_info = next_account_info(account_info_iter)?;
    let authority_info = next_account_info(account_info_iter)?;
    let new_authority_info = next_account_info(account_info_iter)?;

    check_program_account(mint_info.owner)?;
    let mint_data = &mut mint_info.data.borrow_mut();
    let mut mint = StateWithExtensionsMut::::unpack(mint_data)?;
    let confidential_transfer_mint =
        mint.get_extension_mut::()?;

    if authority_info.is_signer
        && confidential_transfer_mint.authority == *authority_info.key
        && (new_authority_info.is_signer || *new_authority_info.key ==
            Pubkey::default())
        && new_confidential_transfer_mint.authority == *new_authority_info.key
    {
        *confidential_transfer_mint = *new_confidential_transfer_mint;
        Ok(())
    } else {
        Err(ProgramError::MissingRequiredSignature)
    }
}
```

Figure 8.1:

https://github.com/solana-labs/solana-program-library/token/program-2022/src/extension/confidential_transfer/processor.rs#L64-L89

```
/// Processes an [ApproveAccount] instruction.
fn process_approve_account(accounts: &[AccountInfo]) -> ProgramResult {
    let account_info_iter = &mut accounts.iter();
    let token_account_info = next_account_info(account_info_iter)?;
    let mint_info = next_account_info(account_info_iter)?;
    let authority_info = next_account_info(account_info_iter)?;

    check_program_account(token_account_info.owner)?;
    let token_account_data = &mut token_account_info.data.borrow_mut();
    let mut token_account =
        StateWithExtensionsMut::<Account>::unpack(token_account_data)?;

    check_program_account(mint_info.owner)?;
    let mint_data = &mut mint_info.data.borrow_mut();
    let mint = StateWithExtensionsMut::<Mint>::unpack(mint_data)?;
    let confidential_transfer_mint =
        mint.get_extension::<ConfidentialTransferMint>()?;

    if authority_info.is_signer && *authority_info.key ==
        confidential_transfer_mint.authority {
        let mut confidential_transfer_state =
            token_account.get_extension_mut::<ConfidentialTransferAccount>()?;
        confidential_transfer_state.approved = true.into();
        Ok(())
    } else {
        Err(ProgramError::MissingRequiredSignature)
    }
}
```

Figure 8.2:

https://github.com/solana-labs/solana-program-library/token/program-2022/src/extension/confidential_transfer/processor.rs#L180-L204

Recommendations

Short term, add a check to ensure that the `auto_approve_new_accounts` property is not `false` when the new authority is `Pubkey::default`. This will ensure that contract users cannot accidentally disable the authorization of accounts for mints.

9. Incorrect test data description

Severity: Informational	Difficulty: Low
Type: Testing	Finding ID: TOB-STK-9
Target: token/program-2022/src/extension/mod.rs	

Description

The comments on the `MINT_WITH_EXTENSION` variable are incorrect. See figure 9.1 for the incorrect comments, highlighted in red, and the corrected comments, highlighted in yellow.

```
const MINT_WITH_EXTENSION: &[u8] = &[  
    // base mint  
    1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
  
    1, 1, 1, 1, 1, 42, 0, 0, 0, 0, 0, 0, 7, 1, 1, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,  
    2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,  
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, // account type  
    1, // extension type == really account type  
    3, 0, // length == really extension type  
    32, 0, // data == really extension length  
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
    1, 1, //== really extension data  
];
```

Figure 9.1:

<https://github.com/solana-labs/solana-program-library/blob/50abadd819df2e406567d6eca31c213264c1c7cd/token/program-2022/src/extension/mod.rs#L828-L841>

Recommendations

Short term, update the comments to align with the data. This will ensure that developers working on the tests will not be confused by the data structure.

10. The Transfer and TransferWithFee instructions are identical

Severity: Informational

Difficulty: Low

Type: Undefined Behavior

Finding ID: TOB-STK-10

Target:

token/program-2022/src/extension/confidential_transfer/processor.rs

Description

The implementations of the Transfer and TransferWithFee instructions are identical. Whether fees are used is determined by whether the mint has a TransferFeeConfig extension, regardless of the instruction used.

```
ConfidentialTransferInstruction::Transfer => {
    msg!("ConfidentialTransferInstruction::Transfer");
    #[cfg(feature = "zk-ops")]
    {
        let data = decode_instruction_data::<TransferInstructionData>(input)?;
        process_transfer(
            program_id,
            accounts,
            data.new_source_decryptable_available_balance,
            data.proof_instruction_offset as i64,
        )
    }
    #[cfg(not(feature = "zk-ops"))]
    Err(ProgramError::InvalidInstructionData)
}
ConfidentialTransferInstruction::TransferWithFee => {
    msg!("ConfidentialTransferInstruction::TransferWithFee");
    #[cfg(feature = "zk-ops")]
    {
        let data = decode_instruction_data::<TransferInstructionData>(input)?;
        process_transfer(
            program_id,
            accounts,
            data.new_source_decryptable_available_balance,
            data.proof_instruction_offset as i64,
        )
    }
    #[cfg(not(feature = "zk-ops"))]
    {
        Err(ProgramError::InvalidInstructionData)
    }
}
```

Figure 10.1:

https://github.com/solana-labs/solana-program-library/blob/50abadd819df2e406567d6eca31c213264c1c7cd/token/program-2022/src/extension/confidential_transfer/processor.rs#L1192-L1223

Recommendations

Short term, deprecate the `TransferWithFee` instruction, and update the documentation for the `Transfer` instruction to clarify the use of fees. This will ensure that contract users will not be misled in how the instructions are performed.

11. Some instructions operate only on the lo bits of balances

Severity: **Undetermined**

Difficulty: **High**

Type: Undefined Behavior

Finding ID: TOB-STK-11

Target:

token/program-2022/src/extension/confidential_transfer/processor.rs

Description

For confidential transfers, the pending balance is split into lo and hi values: of the total 64 bits representing the value, lo contains the low 16 bits, and hi contains the high 48 bits. Some instructions seem to update only the lo bits. For example, the `process_destination_for_transfer` function updates only the `pending_balance_lo` field of the `destination_confidential_transfer_account` account.

Changing the `ct_withdraw_withheld_tokens_from_accounts` integration test so that the resulting fee is greater than `u16::Max` (and updating the test to account for other changes to account balances) breaks the test, which indicates that the pattern of updating only the lo bits is problematic.

We found the same pattern in the `process_withdraw_withheld_tokens_from_mint` function for the `destination_confidential_transfer_account` account and in the `process_withdraw_withheld_tokens_from_accounts` function for the `destination_confidential_transfer_account` account.

```
#[cfg(feature = "zk-ops")]
fn process_destination_for_transfer(
    destination_token_account_info: &AccountInfo,
    mint_info: &AccountInfo,
    destination_encryption_pubkey: &EncryptionPubkey,
    destination_ciphertext_lo: &EncryptedBalance,
    destination_ciphertext_hi: &EncryptedBalance,
    encrypted_fee: Option<EncryptedFee>,
) -> ProgramResult {
    check_program_account(destination_token_account_info.owner)?;
    ...
    // subtract fee from destination pending balance
    let new_destination_pending_balance = ops::subtract(
        &destination_confidential_transfer_account.pending_balance_lo,
        &ciphertext_fee_destination,
    )
    .ok_or(ProgramError::InvalidInstructionData)?;
```



```
// add encrypted fee to current withheld fee
let new_withheld_amount = ops::add(
    &destination_confidential_transfer_account.withheld_amount,
    &ciphertext_fee_withheld_authority,
)
.ok_or(ProgramError::InvalidInstructionData)?;

destination_confidential_transfer_account.pending_balance_lo =
    new_destination_pending_balance;
destination_confidential_transfer_account.withheld_amount = new_withheld_amount;
...
```

Figure 11.1:

https://github.com/solana-labs/solana-program-library/token/program-2022/src/extension/confidential_transfer/processor.rs#L761-L777

Recommendations

Short term, investigate the security implications of operating only on the 10 bits in operations and determine whether this pattern should be changed.

12. Instruction susceptible to front-running

Severity: Informational

Difficulty: Low

Type: Undefined Behavior

Finding ID: TOB-STK-12

Target:

token/program-2022/src/extension/confidential_transfer/instruction.rs

Description

The code comments for the `WithdrawWithheldTokensFromAccounts` instruction state that the instruction is susceptible to front-running. The comments list two alternatives to the function—`HarvestWithheldTokensToMint` and `WithdrawWithheldTokensFromMint`—indicating that this vulnerable function could be deprecated.

```
/// Transfer all withheld tokens to an account. Signed by the mint's withdraw withheld tokens
/// authority. This instruction is susceptible to front-running. Use
/// `HarvestWithheldTokensToMint` and `WithdrawWithheldTokensFromMint` as an alternative.
///
/// Note on front-running: This instruction requires a zero-knowledge proof verification
/// instruction that is checked with respect to the account state (the currently withheld
/// fees). Suppose that a withdraw withheld authority generates the
/// `WithdrawWithheldTokensFromAccounts` instruction along with a corresponding zero-knowledge
/// proof for a specified set of accounts, and submits it on chain. If the withheld fees at any
/// of the specified accounts change before the `WithdrawWithheldTokensFromAccounts` is
/// executed on chain, the zero-knowledge proof will not verify with respect to the new state,
/// forcing the transaction to fail.
///
/// If front-running occurs, then users can look up the updated states of the accounts,
/// generate a new zero-knowledge proof and try again. Alternatively, withdraw withheld
/// authority can first move the withheld amount to the mint using
/// `HarvestWithheldTokensToMint` and then move the withheld fees from mint to a specified
/// destination account using `WithdrawWithheldTokensFromMint`.
```

Figure 12.1:

https://github.com/solana-labs/solana-program-library/blob/50abadd819df2e406567d6eca31c213264c1c7cd/token/program-2022/src/extension/confidential_transfer/instruction.rs#L313-L330

Recommendations

Short term, consider deprecating this instruction in favor of the alternatives suggested in the `WithdrawWithheldTokensFromAccounts` code comments, `HarvestWithheldTokensToMint` and `WithdrawWithheldTokensFromMint`. This will ensure that contract users who may have missed the comment describing the front-running vulnerability will not be exposed to the issue.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	Description
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	Description
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	Description
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Non-Security-Related Findings

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

- Several `msg!` lines are missing an ending semicolon. Insert all of the missing semicolons:

```
> grep -rn 'msg!' --include '*.rs' | grep '$'
./src/error.rs:188:      msg!("Error: Invalid number of provided signers")
./src/error.rs:191:      msg!("Error: Invalid number of required signers")
./src/error.rs:195:      msg!("Error: Instruction does not support native tokens")
./src/error.rs:198:      msg!("Error: Non-native account can only be closed if its balance
is zero")
./src/error.rs:204:      msg!("Error: Account does not support specified authority type")
./src/error.rs:209:      msg!("Error: decimals different from the Mint decimals")
./src/error.rs:212:      msg!("Error: Instruction does not support non-native tokens")
./src/error.rs:215:      msg!("Error: New extension type does not match already existing
extensions")
./src/error.rs:218:      msg!("Error: Extension does not match the base type provided")
./src/error.rs:221:      msg!("Error: Extension already initialized on this account")
./src/error.rs:224:      msg!("Error: An account can only be closed if its confidential
balance is zero")
./src/error.rs:227:      msg!("Error: Account not approved for confidential transfers")
./src/error.rs:230:      msg!("Error: Account not accepting deposits or transfers")
./src/error.rs:233:      msg!("Error: ElGamal public key mismatch")
./src/error.rs:236:      msg!("Error: Balance mismatch")
./src/error.rs:239:      msg!("Error: Mint has non-zero supply. Burn all tokens before
closing the mint")
./src/error.rs:254:      msg!("Fee parameters associated with zero-knowledge proofs do not
match fee parameters in mint")
./src/error.rs:275:      msg!("Deposit amount exceeds maximum limit")
```

- In the `get_first_extension_type` function, change the comparison operator in the line `if tlv_data.len() <= tlv_indices.length_start` from `<=` to `<`. If `tlv_data.len()` is equal to `tlv_indices.length_start`, that means there is enough data to parse the extension type.
- Update the `comment` in the `is_initialized_account` function to provide more information about the hard-coded index and point to the correct line in `state.rs`.
- The `expected_decimals` parameter in the `process_withdraw` function is used only to check against a known “correct” value. Because the correct value is already known, the parameter can be removed.

C. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

On December 27, 2022, Trail of Bits reviewed the fixes and mitigations implemented by the Solana team to resolve the issues identified in this report.

In summary, Solana has resolved nine issues, has partially resolved one issue, and has not resolved the remaining two issues. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Severity	Status
1	Ok returned for malformed extension data	Informational	Resolved
2	Missing account ownership checks	Undetermined	Resolved
3	Use of a vulnerable dependency	Undetermined	Partially resolved
4	Large extension sizes can cause panics	Informational	Resolved
5	Unexpected function behavior	Informational	Resolved
6	Out of bounds access in the get_extension instruction	Low	Resolved
7	Iteration over empty data	Informational	Resolved
8	Missing check in UpdateMint instruction could result in inoperable mints	Low	Unresolved
9	Incorrect test data description	Informational	Resolved

10	The Transfer and TransferWithFee instructions are identical	Informational	Resolved
11	Some instructions operate only on the lo bits of balances	Undetermined	Resolved
12	Instruction susceptible to front-running	Informational	Unresolved

Detailed Fix Review Results

TOB-STK-1: Ok returned for malformed extension data

Resolved (in [PR #3762](#)). The `get_extension_types` function now checks that there is enough data to unpack and returns an error if there is not.

TOB-STK-2: Missing account ownership checks

Resolved (in [PR #3759](#)). Ownership checks have been added to the relevant instructions.

TOB-STK-3: Use of a vulnerable dependency

Partially resolved. An issue outlining the problem ([issue #3697](#)) has been created in the `solana-labs/Solana-program-library` repository, but it still remains to be triaged.

TOB-STK-4: Large extension sizes can cause panics

Resolved (in [PR #3754](#)). The `init_extension` function now propagates the error back to the caller if the length of the given extension is outside the usable range.

TOB-STK-5: Unexpected function behavior

Resolved (in [PR #3760](#)). A comment that clarifies the input requirement and the rationale for the behavior has been added to the `decode_instruction_data` function.

TOB-STK-6: Out of bounds access in the get_extension instruction

Resolved (in [PR #3751](#)). The `get_extension` function now checks that there is enough data to unpack.

TOB-STK-7: Iteration over empty data

Resolved (in [PR #3761](#)). When it encounters uninitialized data, the `get_extension_indices` function now breaks the loop and returns an error indicating that the requested extension was not found.

TOB-STK-8 Missing check in UpdateMint instruction could result in inoperable mints

Unresolved. The Solana team has not resolved this issue and accepts the associated risks.

TOB-STK-9: Incorrect test data description

Resolved (in [PR #3752](#)). The comments on the test data now align with the respective fields.

TOB-STK-10: The Transfer and TransferWithFee instructions are identical

Resolved (in [PR #3766](#)). The Solana team has simplified the contract API by removing the `TransferWithFee` instruction.

TOB-STK-11: Some instructions operate only on the lo bits of balances

Resolved (in [PR #3878](#)). The encrypted fee now uses both the `lo` and `hi` components, which are then added to the balance.

TOB-STK-12: Instruction susceptible to front-running

Unresolved. The Solana team has not resolved this issue and accepts the associated risks.