Cyberscope

A TAC Security Company

Audit Report

Unclaimed SOL

October 2025

Repository :

https://github.com/nedim1511/unclaimed-sol-close-token-program

Commit :

ddfc570b1a21e2a2ae0a59eca11d0aca0f455241

Audited by © cyberscope

Table of Contents

Risk Classification

The criticality of findings in Cyberscope’s smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Repository	https://github.com/nedim1511/unclaimed-sol-close-token-program/tree/main
Commit	ddfc570b1a21e2a2ae0a59eca11d0aca0f455241

Audit Updates

Initial Audit	16 Oct 2025

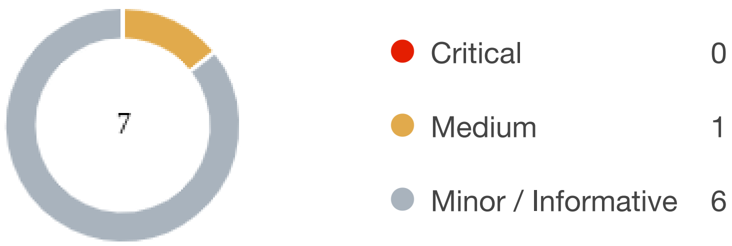
Source Files

Filename	SHA256
lib.rs	f252e9de56d9037e50918bf335af52f9c911ae882c5aed61a950f40391a7005f

Overview

The audited Solana program provides a secure, automated solution for managing SPL token accounts, enabling users to burn tokens and close accounts while reclaiming lamports with a limited fee. It enforces strict account validation, including verification of the fee recipient and signer authorization. Users submit instructions specifying mint and token account pairs, which the program processes by burning any remaining token balance and closing the account to recover lamports. A fee, capped at 5% of the recovered lamports, is transferred to the designated fee address. This design prevents unauthorized actions and streamlines lamport recovery across multiple token accounts in a single transaction, delivering secure, transparent, and efficient token management on Solana.

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	1	0	0	0
Minor / Informative	6	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	MFV	Missing Fee Validation	Unresolved
Minor / Informative	ALM	Array Length Mismatch	Unresolved
Minor / Informative	MCM	Misleading Comment Messages	Unresolved
Minor / Informative	MC	Missing Check	Unresolved
Minor / Informative	MEM	Missing Error Messages	Unresolved
Minor / Informative	MEE	Missing Events Emission	Unresolved
Minor / Informative	MWV	Missing Writability Validation	Unresolved

MFV - Missing Fee Validation

Criticality	Medium
Location	lib.rs#L84
Status	Unresolved

Description

The contract implements a fee mechanism on the redeemed amount. However, the fee percentage is provided by the user when invoking the method. As a result, users can set the fee to zero, effectively bypassing the intended fee mechanism.

```rust
let fee_percentage = instruction_data[1].min(5);
```

Recommendation

It is advisable to consider implementing a fixed fee percentage or a minimum fee. This will ensure the fee mechanism remains consistent and functions as intended.

ALM - Array Length Mismatch

Criticality	Minor / Informative
Location	lib.rs#L90
Status	Unresolved

Description

The contract is designed to handle the process of elements from an array through a function that accepts an array as input. This function is intended to iterate over the array, processing elements in a coordinated manner. However, there are no explicit checks to verify that the length of the input array is consistent to the operations. This lack of validation could lead to scenarios where the arrays have differing lengths than intented, potentially causing out-of-bounds access. Such situations could result in unexpected behavior or errors during the contract's execution, compromising its reliability and security.

```rust
while let (Ok(mint_account_info),
Ok(token_account_info)) = (
next_account_info(accounts_iter),
next_account_info(accounts_iter),)
```

Recommendation

To mitigate this, it is recommended to incorporate a validation check at the beginning of the function that accepts an array to ensure that the lengths align to the intended design. In this case this can be achieved by ensuring the array length is an even number. Such validations will prevent out-of-bounds errors and ensure that the elements of the array are processed in a paired and coordinated manner.

MCM - Misleading Comment Messages

Criticality	Minor / Informative
Location	lib.rs#L148
Status	Unresolved

Description

The contract is using misleading comment messages. These comment messages do not accurately reflect the actual implementation, making it difficult to understand the source code. As a result, the users will not comprehend the source code's actual implementation.

Shell
```
solana_program::instruction::AccountMeta::

new(*token_account_info.key, false),

// Destination for rent SOL
```

Recommendation

The team is advised to carefully review the comment in order to reflect the actual implementation. To improve code readability, the team should use more specific and descriptive comment messages.

MC - Missing Check

Criticality	Minor / Informative
Location	lib.rs#L52
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. Specifically the contract does not validate that the provided accounts are the default address. These variables may produce vulnerability issues.

```rust
fn burn_and_close_token_accounts(
    _program_id: &Pubkey,
    accounts: &[AccountInfo],
    instruction_data: &[u8],
) -> ProgramResult {
...

}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	lib.rs#L90
Status	Unresolved

Description

The contract is missing error messages. Specifically, there are no error messages to accurately reflect the case of incomplete or invalid elements in the array, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

Rust

```
while let (Ok(mint_account_info),
Ok(token_account_info))
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	lib.rs#L37
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

Rust
```rust
pub fn process_instruction(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
    instruction_data: &[u8],
) -> ProgramResult {
    if instruction_data.is_empty() {
        return Err(ProgramError::InvalidInstructionData);
    };
    let instruction_selector = instruction_data[0];
    match instruction_selector {
        0 => burn_and_close_token_accounts(program_id,
accounts, instruction_data),
        _ => Err(ProgramError::InvalidInstructionData),
    }

}
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

MWV - Missing Writability Validation

Criticality	Minor / Informative
Location	lib.rs#L52
Status	Unresolved

Description

The contract accepts user-provided accounts and modifies their state. These accounts may be non-writable. If non-writable accounts are provided, the contract execution will fail.

Rust
```rust
fn burn_and_close_token_accounts(
    _program_id: &Pubkey,
    accounts: &[AccountInfo],
    instruction_data: &[u8],
) -> ProgramResult {
...

}
```

Recommendation

The team is advised to implement the necessary checks. Ensuring that all state changing accounts are writable will maintain consistency of operations.

Summary

Unclaimed SOL contract implements a utility mechanism enabling users to burn tokens from their accounts and claim the associated lamports as refunds. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a TAC blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

A *TAC Security* Company

The Cyberscope team

cyberscope.io