Zellic

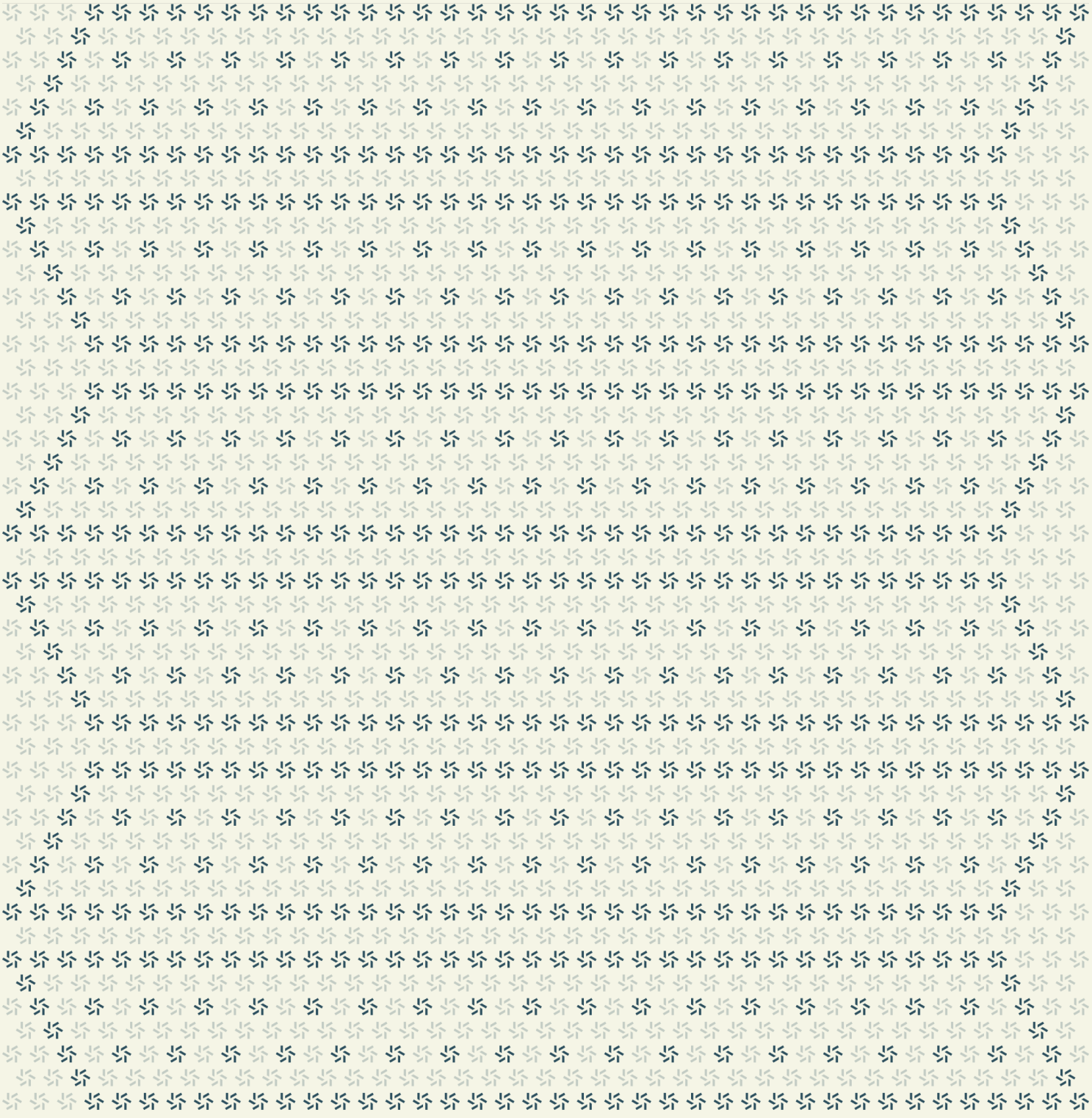
Prepared for
Ray J
Marcus
Roneil Rumburg
Audius

Prepared by
Ethan Lee
Guo Xuehao
Zellic

November 11, 2025

Claim and Rewards Programs

Smart Contract Patch Review



Contents

About Zellic

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

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

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

1. Introduction

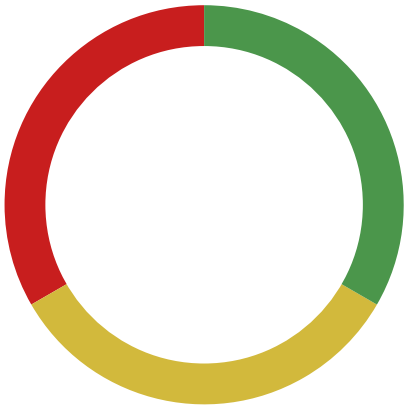
We were asked to review a minor patch to the claim and rewards programs on November 9th, 2025, which changed two components of the project. The changes were made to the reward-manager and claimable-tokens programs. The patch was made to add the ability to burn tokens when tokens are sent to the zero address and to allow for ownership transfer and close account authority on claimable token accounts.

1.1. Results

During our assessment on the scoped claim and rewards programs, we discovered three findings. One critical issue was found. One was of medium impact and one was of low impact.

Breakdown of Finding Impacts

Impact Level	Count
Critical	1
High	0
Medium	1
Low	1
Informational	0



1.2. Scope

The engagement involved a review of the following targets:

Claim and Rewards Programs

Type	Solana
Platform	Solana VM

Target	apps
Repository	https://github.com/AudiusProject/apps ↗
Version	`f92bbdf1d6fe5953974f42b42e653fc7542c46ff`
Programs	`reward-manager`

Target	apps
Repository	https://github.com/AudiusProject/apps ↗
Version	`d8faad177e8fdaa3cf8b79bee452bf00050c7195`
Programs	`claimable-tokens`

Contact Information

The following project manager was associated with the engagement:

Pedro Moura
Engagement Manager
pedro@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Ethan Lee
Engineer
ethl@zellic.io ↗

Guo Xuehao
Engineer
guo@zellic.io ↗

1.3. Disclaimer

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

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

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

2. Detailed Findings

2.1. Arbitrary authority modification in `SetAuthority`

Target	claimable-tokens		
Category	Coding Mistakes	**Severity**	Critical
Likelihood	High	**Impact**	Critical

Description

The `SetAuthority` path does not strictly validate the offsets and instruction indexes used by the secp256k1 native program. As a result, the data covered by secp signature verification and the data subsequently read by the contract may potentially come from different instructions.

An attacker can make secp verify a legitimate but business-irrelevant signed data segment, while having the contract read, and use another segment of data that is not protected by the signature — thereby bypassing signature binding and changing the target account's authority to an attacker-specified value.

```
fn process_set_authority_instruction<'a>(
    program_id: &Pubkey,
    token_account_info: AccountInfo<'a>,
    authority_account_info: AccountInfo<'a>,
    sysvars_instruction_info: AccountInfo<'a>,
    recent_blockhashes_account_info: AccountInfo<'a>,
) -> ProgramResult {
        ...

    // is that instruction is `new_secp256k1_instruction`
    if instruction.program_id != secp256k1_program::id() {
        msg!("Incorrect program id for secp256k1 instruction");
        return Err(ClaimableProgramError::Secp256InstructionLosing.into());
    }

    // Parse the secp256k1 instruction
    let offsets = SecpSignatureOffsets::try_from_slice(
        &instruction.data[1..(1 + SIGNATURE_OFFSETS_SERIALIZED_SIZE)],
    )?;
    let eth_address_signer = &instruction.data
        [offsets.eth_address_offset as usize..(offsets.eth_address_offset
    as usize + size_of::<EthereumAddress>())];
    let message = &instruction.data
        [offsets.message_data_offset as usize
            ..(offsets.message_data_offset + offsets.message_data_size)
```

```
    as usize
        ];
    ...
```

Impact

An attacker can construct a signature unrelated to the target account, execute `SetAuthority` on any claimable token account, and specify an arbitrary `new_authority`. This allows them to take over subsequent asset-transfer rights of the account, ultimately resulting in asset loss.

Recommendations

Validate the secp offsets and indexes in the `set_authority` instruction. Also validate that there is only one secp instruction.

Remediation

This issue has been acknowledged by Audius, and a fix was implemented in commit `76d5ee21` ↗.

2.2. Signature replay in the `SetAuthority` instruction

Target	claimable-tokens		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	High	**Impact**	Medium

Description

The `process_set_authority_instruction` function in the claimable-tokens program contains a signature-replay vulnerability. The `SignedSetAuthorityData` struct only includes `blockhash` and `instruction` data but does not include the specific `token_account` address that should be modified. This allows an attacker to reuse a valid signature intended for one token account to modify the authority of a different token account, as long as both accounts are derived from the same Ethereum signer address.

```
//
    https://github.com/AudiusProject/apps/blob/d8faad177e8fdaa3cf8b79bee452bf00050c7195
    programs/claimable-tokens/program/src/state.rs#L24-L29
pub struct SignedSetAuthorityData {
    /// A recent blockhash to prevent replays
    pub blockhash: Hash,
    /// The instruction data as a serialized byte array
    pub instruction: Vec<u8>,
}
```

The validation logic only verifies that the provided accounts match the derivation from the Ethereum signer; it does not check whether the signature was specifically created for the provided token account.

```
//
    https://github.com/AudiusProject/apps/blob/d8faad177e8fdaa3cf8b79bee452bf00050c7195
    programs/claimable-tokens/program/src/processor.rs#L419-L432
let signer_address = EthereumAddress::try_from_slice(eth_address_signer)
    .map_err(|_| ClaimableProgramError::SignatureVerificationFailed)?;
let mint =
    &spl_token::state::Account::unpack(&token_account_info.data.borrow())?.mint;
let pair = find_address_pair(program_id, mint, signer_address)?;
let derived_authority = pair.base.address;
let derived_token_account = pair.derive.address;
if *authority_account_info.key != derived_authority {
    msg!("Authority account mismatch");
```

```
    return Err(ClaimableProgramError::SignatureVerificationFailed.into());
}
if *token_account_info.key != derived_token_account {
    msg!("Token account mismatch");
    return Err(ClaimableProgramError::SignatureVerificationFailed.into());
}
// No check that the signature was created for this specific token_account
```

An attacker who intercepts a legitimate `SetAuthority` signature can replay it against any other token account derived from the same Ethereum address.

Impact

When a victim transfers authority to an address they do not control (e.g., burning authority by setting to null), an attacker can intercept and replay this signature to unintentionally change the authority of other token accounts derived from the same Ethereum address. While the attacker cannot choose the new authority arbitrarily, they can cause unintended authority changes on the victim's other token accounts.

Recommendations

Modify `SignedSetAuthorityData` to include the `token_account` address as part of the signed message. Add validation to verify that the signed token account matches the provided account before executing the authority change.

Remediation

This issue has been acknowledged by Audius, and a fix was implemented in commit `76d5ee21` ↗.

2.3. Claimable token close can be subjected to denial of service via dust attack

Target	claimable-tokens		
Category	Business Logic	**Severity**	Low
Likelihood	Medium	**Impact**	Low

Description

When a claimable token account is about to be closed with a balance of zero, an attacker can perform a dust attack by transferring a very small amount of tokens with the same mint to that account before the close operation. This dust attack makes the balance nonzero, triggering the SPL token's zero-balance validation for `close_account` and causing the close operation to fail.

```
fn process_close_instruction<'a>(
    program_id: &Pubkey,
    token_account_info: AccountInfo<'a>,
    authority_account_info: AccountInfo<'a>,
    destination_account_info: AccountInfo<'a>,
    eth_address: EthereumAddress,
) -> Result<(), ProgramError> {
    ...

    invoke_signed(
        &spl_token::instruction::close_account( // <-----
            &spl_token::id(),
            token_account_info.key,
            destination_account_info.key,
            authority_account_info.key,
            &[authority_account_info.key],
        )?,
        &[token_account_info, destination_account_info,
    authority_account_info],
        seeds,
    )

}
```

Impact

The `Close` instruction can be subjected to a denial-of-service attack, preventing rent reclamation.

Recommendations

Extract the balance before performing the close operation.

Remediation

This issue has been acknowledged by Audius.