Security Audit
Audius Payment Router

Lead Auditor:	Thomas Lambertz
Second Auditor:	Nico Gründel
Administrative Lead:	Thomas Lambertz

November 27th 2023

Table of Contents

Executive Summary

Neodyme audited two of **Audius'** on-chain programs, namely the **Staking Bridge** and **Payment Router** during late August and September 2023. This report is only for the **Payment Router**, with an accompanying report detailing the bridge.

The auditors found that Audius' programs comprised a clean design and straightforward functionality. According to Neodymes Rating Classification, **no security relevant** and only **two informational issues** were found. The number of findings identified throughout the audit, grouped by severity, can be seen in Figure 1.

All findings were reported to the Audius developers and **addressed promptly**. The security fixes were verified for completeness by Neodyme. In addition to these findings, Neodyme delivered the Audius team a list of nit-picks and additional notes that are not part of this report.

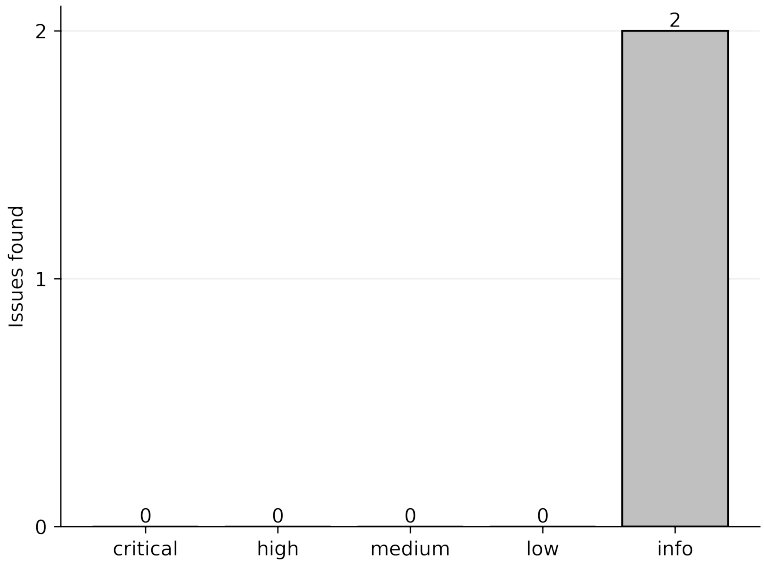

Figure 1: Overview of Findings

1 Introduction

During the summer of 2023, Audius commissioned Neodyme to conduct a detailed security analysis of two of their new contracts: The Staking Bridge and Payment Router. Two senior auditors performed the audit between August 28th and September 1st, 2023. To account for fixes in the contract, additional auditing was carried out later in September. This report details all findings and their fixes for the **Payment Router**.

The audit mainly focused on the contract's technical security but also considered its design and potential social engineering attack vectors. After the introduction, this report details the audit's Scope, gives a brief Overview of the Contract's Design, then goes on to document Findings.

The contract is straightforward, with no unnecessary authorities or complications. They are open-source and rely on the established Anchor framework, integration tests are available. The Audius team always responded quickly and **competent** to findings and questions of any kind.

Findings Summary

During the audit, **no security-relevant** and **two informational** findings were identified. Audius remediated all of those findings before the contract's launch.

In total, the audit revealed:

0 critical • **0** high-severity • **0** medium-severity • **0** low-severity • **2** informational

issues. All findings are detailed in the Findings section.

2 | Scope

The contract audit's scope comprised of a smart-contracts developed by Audius: The **Payment Router**. Another contract, the **Staking Bridge** was audited as well but is not part of this report.

During the audit, we focused on the **implementation** security of the source code and the security of the **overall design**.

All of the source code is located at https://github.com/AudiusProject/audius-protocol/tree/main/solana-programs. Only the router contract mentioned is in scope; third-party dependencies are not. As Audius only relies on the well-established Anchor library, the security-txt standard, and a dependency on the hex crate, this does not seem problematic.

Relevant source code revisions are:

- `1659604268cf5f8ac562f05d829d982f2c41c3d1` • Start of the audit
- `e10316718141bd96e683dac57422b5716c70b850` • Commit including all reviewed security fixes

3 | Project Overview

This section briefly outlines the contract's functionality, design, and architecture followed by a discussion on its authorities and security features.

Functionality

The **Payment Router** contract's task is simple: Distribute a set of payments from one paying party to a list of recipients. This is implemented by having the paying party transfer tokens onto an arbitrary token account owned by a fixed PDA. This gives the contract access to the funds. By invoking the route instruction, the funds are then distributed according to the passed parameters. It supports any token and isn't intended to have user accounts or hold any funds long-term.

On-Chain Data and Accounts

The on-chain contract does not need to maintain much state, it just controls PDA token accounts users create. It uses a single PDA for their token authority, with `payment_router` as seeds.

Authority Structure and Off-Chain Components

There are no explicit contract authorities except for the upgrade authority. Audius plans reqlinquish the upgrade authority before the contracts have substantial adoption. As the contract does not store any data, they can be redeployed to different addresses if changes are required. This will make the contracts truly permissionless.

There is no configuration or an admin account, as all functions are permissionless. The functionality users gain by distributing payments is handled off-chain and out-of-scope in this audit.

Instructions

The contract is straightforward and only has two instructions, which we briefly summarize here for completeness.

Table 1: Instructions with Descriptions for Payment Router

Instruction	Category	Summary
CreatePaymentRouterBalancePDA	Permissionless, One-Time	Initialize an anchor account at the PDA, which will temporarily hold funds. Not strictly necessary to create, but makes it clear which account is the correct one.
Route	Permissionless	Transfers tokens from an ATA owned by a PDA to arbitrary recipient accounts, which are passed as parameters together with the respective amounts.

4 | Findings

This section outlines all of our findings. They are classified into one of five severity levels, detailed in Appendix C. In addition to these findings, Neodyme delivered the Audius team a list of nit-picks and additional notes that are not part of this report.

All findings are listed in Table 3 and further described in the following sections.

Table 2: Findings

Identifier	Name	Severity	State
ND-AUD1-I1	Potential footgun in route instruction	Info	Resolved
ND-AUD1-I2	Anchor readability improvements	Info	Resolved

ND-AUD1-I1 – Potential footgun in route instruction

Severity	Impact	Affected Component	Status
Informational	-	Payment Router	Remediated

The design of the route instruction creates a potential footgun that all frontends must be careful about: All routing *has* to happen in one transaction.

The current flow is like follows:

1. A user transfers funds into a PDA owned by the routing program
2. A user calls the routing program to transfer the funds into a set of targets

If an attacker can interject transactions between 1. and 2., he can send the user's funds to arbitrary other locations. Even if a user sends two transactions together, there is no guarantee that they won't be split up, if accidentally or on purpose by an attacker.

The current design, with the transfer to the contract, also has the property that funds might be left in the contract if not all are routed out.

Potential Alternative

There is a potential alternative, though it isn't necessarily better. The current implementation has the really nice property that the user does *not* give a signature to the payment-router program. So even if the payment-router program is maliciously upgraded sometime in the future, it can not access more funds than the user transferred.

If we are willing to trade that rather nice property for fewer foot guns, we could make the contract do all the transfers directly from the user to all recipients. That would remove the need for a PDA entirely, and no temporary holding in a PDA means no footguns.

In this case, the footgun is a worthwhile tradeoff for the added security, but it should be well documented, and frontends must be careful.

For the second issue with left-over funds, the route instruction could take the user's token account as a fallback account. That fallback account would always get any left-over funds back at the end of the routing loop. If the token-account owner, which isn't needed anyways, is not passed to the instruction, the security properties mentioned above are kept intact.

Remediation

The Audius team was already aware of the raised point, and will always do all routing in a single transaction. In addition, the frontend will always send the same token amount as specified in the `total_amount` argument to the Route instruction, which will ensure that the amount to distribute in this call matches the total.

ND-AUD1-I2 – Anchor readability improvements

Severity	Impact	Affected Component	Status
Informational	-	Anchor Checks	Resolved

The contracts use Anchor to do a lot of the necessary account checks. All required checks are present, but some could be improved. While it does not cause direct security or usability issues, it does make it harder for users to review the contract quickly. For the payment router in particular, we have one improvement suggestion:

The `payment_router_pda` account is initialized as `Account<'info, Empty>`, but later used as `UncheckedAccount<'info>`. While this doesn't cause user-facing issues, it would be more consistent to use the `Empty` tag everywhere. That way, it is also verified that the bump seed provided by the user is the correct one, which isn't guaranteed otherwise. Anchor does *not* verify that the bump seed on an account is the canonical account seed if it is explicitly provided and the account is *not* being initialized. Usually, the account is either being initialized or type-tag-checked. However, since `UncheckedAccount` is used, the type tag check is missing. Due to the way the programs are structured, this doesn't cause visible issues, though. An attacker could swap between a different token-account pair but doesn't gain anything.

Remediation

The Audius team improved the checks in a number of commits: `cc4cfab`, `e741112`, `fdf5581`, `308ba2e`, `82f03ac`, `55b8323`.

A | About Neodyme

Security is difficult.

To understand and break complex things, you need a certain type of people. People who thrive in complexity, who love to play around with code, and who don't stop exploring until they fully understand every aspect of it. That's us.

Our team never outsources audits. Having found over 80 High or Critical bugs in Solana's core code itself, we believe that Neodyme hosts the most qualified auditors for Solana programs. We've also found and disclosed critical vulnerabilities in many of Solana's top projects and have responsibly disclosed issues that could have resulted in the theft of over $10B in TVL on the Solana blockchain.

All of our team members have a background in competitive hacking. During such hacking competitions, called CTFs, we competed and collaborated while finding vulnerabilities, breaking encryption, reverse engineering complicated algorithms, and much more. Through the years, many of our team members have won national and international hacking competitions, and keep ranking highly among some of the hardest CTF events world-wide. In 2020, some of our members started experimenting with validators and became active members in the early Solana community. With the prospect of an interesting technical challenge and bug bounties, they quickly encouraged others from our CTF team to look for security issues in Solana. The result was so successful that after reporting several bugs, in 2021, the Solana Foundation contracted us for source code auditing. As a result, Neodyme was born.

B | Methodology

Neodyme prides itself on not being a checklist auditor. We adapt our approach to each audit, investing considerable time into understanding the program upfront and exploring its expected behavior, edge cases, invariants, and ways in which the latter could be violated. We use our uniquely deep knowledge of Solana internals, and our years-long experience in auditing Solana programs to even find bugs that others miss. We often extend our audit to cover off-chain components, in order to see how users could be tricked or the contract affected by bugs in those components.

Nonetheless, we also have a list of common vulnerability classes, which we always exhaustively look for. We provide a sample of this list below.

- Rule out common classes of Solana contract vulnerabilities, such as:
  - Missing ownership checks
  - Missing signer checks
  - Signed invocation of unverified programs
  - Solana account confusions
  - Redeployment with cross-instance confusion
  - Missing freeze authority checks
  - Insufficient SPL account verification
  - Missing rent exemption assertion
  - Casting truncation
  - Arithmetic over- or underflows
  - Numerical precision errors
- Check for unsafe design which might lead to common vulnerabilities being introduced in the future
- Check for any other, as-of-yet unknown classes of vulnerabilities arising from the structure of the Solana blockchain
- Ensure that the contract logic correctly implements the project specifications
- Examine the code in detail for contract-specific low-level vulnerabilities
- Rule out denial of service attacks
- Rule out economic attacks
- Check for instructions that allow front-running or sandwiching attacks
- Check for rug pull mechanisms or hidden backdoors

C | Vulnerability Severity Rating

Critical Vulnerabilities that will likely cause loss of funds. An attacker can trigger them with little or no preparation, or they are expected to happen accidentally. Effects are difficult to undo after they are detected.

High Bugs that can be used to set up loss of funds in a more limited capacity, or to render the contract unusable.

Medium Bugs that do not cause direct loss of funds but that may lead to other exploitable mechanisms, or that could be exploited to render the contract partially unusable.

Low Bugs that do not have a significant immediate impact and could be fixed easily after detection.

Info Bugs or inconsistencies that have little to no security impact.

Neodyme AG

Dirnismaning 55
Halle 13
85748 Garching
E-Mail: contact@neodyme.io

https://neodyme.io