Zellic

Audius Solana Programs

Smart Contract Security Assessment

November 21, 2022

Prepared for:

Ray Jacobson

Audius, Inc

Prepared by:

Filippo Cremonese and Jasraj Bedi

Zellic Inc.

Contents

About Zellic

Zellic was founded in 2020 by a team of blockchain specialists with more than a decade of combined industry experience. We are leading experts in smart contracts and Web3 development, cryptography, web security, and reverse engineering. Before Zellic, we founded perfect blue, the top competitive hacking team in the world. Since then, our team has won countless cybersecurity contests and blockchain security events.

Zellic aims to treat clients on a case-by-case basis and to consider their individual, unique concerns and business needs. Our goal is to see the long-term success of our partners rather than simply provide a list of present security issues. Similarly, we strive to adapt to our partners' timelines and to be as available as possible. To keep up with our latest endeavors and research, check out our website zellic.io or follow @zellic_io on Twitter. If you are interested in partnering with Zellic, please email us at hello@zellic.io or contact us on Telegram at https://t.me/zellic_io.

1 Executive Summary

Zellic conducted an audit for Audius, Inc from October 10th to October 14th, 2022.

Our general overview of the code is that it was very well-organized and structured. Tests are included for the majority of the functions. The documentation was adequate, although it could be improved. The code was easy to comprehend, and in most cases, intuitive.

We applaud Audius, Inc for their diligence in maintaining high code quality standards in the development of Audius Solana Programs as well as their responsiveness demonstrated while the audit was ongoing.

Zellic thoroughly reviewed the Audius Solana Programs codebase to find protocol-breaking bugs as defined by the documentation and to find any technical issues outlined in the Methodology section (2.2) of this document.

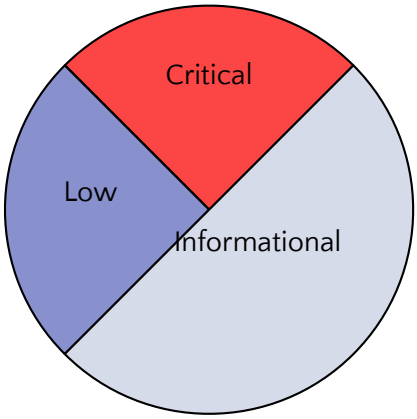
Specifically, taking into account Audius Solana Programs's threat model, we focused heavily on issues that would break core invariants such as requiring signatures from the appropriate signers to redeem and transfer reward tokens as well as to manage groups of trusted signers.

During our assessment on the scoped Audius Solana Programs contracts, we discovered four findings. One finding was of critical severity. Of the remaining findings, one was of low severity, and the remaining findings were informational in nature.

Additionally, Zellic recorded its notes and observations from the audit for Audius, Inc's benefit in the Discussion section (4) at the end of the document.

Breakdown of Finding Impacts

Impact Level	Count
Critical	1
High	0
Medium	0
Low	1
Informational	2



2 Introduction

2.1 About Audius Solana Programs

Audius Solana Programs is a decentralized, community-owned and artist-controlled music-sharing protocol. Audius provides a blockchain-based alternative to existing streaming platforms to help artists publish and monetize their work and distribute it directly to fans. The mission of the project is to give everyone the freedom to share, monetize, and listen to any audio.

2.2 Methodology

During a security assessment, Zellic works through standard phases of security auditing including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of open-source tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. We analyze the scoped smart contract code using automated tools to quickly sieve out and catch these shallow bugs. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so forth as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We manually review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents. We also thoroughly examine the specifications and designs themselves for inconsistencies, flaws, and vulnerabilities. This involves use cases that open the opportunity for abuse, such as flawed tokenomics or share pricing, arbitrage opportunities, and so forth.

Complex integration risks. Several high-profile exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. We perform a meticulous review of all of the contract's possible external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so forth.

Code maturity. We review for possible improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradeability weaknesses, centralization risks, and so forth.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact; we assign it on a case-by-case basis based on our professional judgment and experience. As one would expect, both the severity and likelihood of an issue affect its impact; for instance, a highly severe issue's impact may be attenuated by a very low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Similarly, Zellic organizes its reports such that the most important findings come first in the document rather than being ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their importance may differ. This varies based on numerous soft factors, such as our clients' threat models, their business needs, their project timelines, and so forth. We aim to provide useful and actionable advice to our partners that consider their long-term goals rather than simply provide a list of security issues at present.

2.3 Scope

The engagement involved a review of the following targets:

Audius Solana Programs Contracts

Repository	https://github.com/AudiusProject/audius-protocol
Versions	`9b82a8c962aa0524c2db2e73c55854b4c7200c9d`
Programs	• claimable-tokens • reward-manager
Type	Rust
Platform	Solana

2.4 Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of two person-weeks. The assessment was conducted over the course of one calendar week.

Contact Information

The following project managers were associated with the engagement:

Jasraj Bedi, Co-founder
jazzy@zellic.io

Stephen Tong, Co-founder
stephen@zellic.io

The following consultants were engaged to conduct the assessment:

Filippo Cremonese, Engineer
fcremo@zellic.io

Jasraj Bedi, Co-founder, Engineer
jazzy@zellic.io

2.5 Project Timeline

The key dates of the engagement are detailed below.

October 10, 2022	Start of primary review period
October 14, 2022	End of primary review period
November 21, 2022	Final report delivery

3 Detailed Findings

3.1 Missing PDA validation leading to multiple transfers

- **Target**: Rewards Manager
- **Category**: Coding Mistakes
- **Likelihood**: High
- **Severity**: Critical
- **Impact**: Critical

Description

Rewards are redeemed using a two-step process. First, signed messages are submitted and stored on-chain in an account of type `VerifiedMessages`. When the required amount of signed messages has been submitted, the `EvaluateAttestations` instruction is invoked to process the transfer. The instruction performs a number of checks on the provided accounts and then performs the token transfer to the destination account. In order to avoid a single transfer being repeated multiple times, a PDA is created (`transfer_account_info`), marking the transfer as completed. The PDA is unique for the transfer since the address is derived from the details of the transfer, including a unique ID. In addition, the account containing the `VerifiedMessages` is deleted by zeroing its lamports. Both these measures are flawed and can be bypassed.

The `transfer_account_info` account is not checked to be the intended PDA. An attacker can supply any signer account as an input to the transaction, and the account will be created successfully. This is because any signer account can be passed to the `create_account` system instruction, even if the `invoke_signed` function is used to perform an invocation with signer seeds for the intended PDA. The signer seeds will just be ignored as they do not correspond to any account in the subtransaction.

It is also possible to reuse the `VerifiedMessages` account, despite it having zero lamports, by referencing it in multiple instructions within the same transaction. This specific issue is discussed more in detail in finding 3.3.

Impact

It is possible to redeem rewards multiple times. We confirmed this issue by modifying an existing test.

Recommendations

Ensure that the `transfer_account_info` account matches the expected PDA. Properly invalidate the data stored in the `VerifiedMessages` accounts so that it cannot be reused

even within the same transaction.

Remediation

The Audius team was alerted of this issue while the audit was ongoing. The issue was acknowledged within 10 minutes, and a remediation patch was suggested within 40 minutes. The patch was quickly deployed after review from both Zellic and Audius engineers to ensure a complete fix to the issue. The complete timeline of events follows (times in UTC, October 15th):

- 17:52 Audius is informed of the issue
- 18:02 Audius acknowledges the issue
- 18:31 Audius proposes a remediation
- 18:35 Zellic confirms that proposed remediation patches the issue, suggesting additional changes to invalidate `VerifiedMessages` accounts
- ~21:45 Audius finalizes remediation commits, including suggested additional changes
- ~22:00 Zellic confirms that remediation patches the issue
- ~22:00 Audius deploys and tests patch on testnet
- 23:31 Audius deploys patch on mainnet

3.2 Ambiguous format for signed messages

- **Target**: Rewards Manager
- **Category**: Code Maturity
- **Likelihood**: N/A
- **Severity**: Informational
- **Impact**: Informational

Description

Verified messages are serialized as the concatenation of multiple fields separated by an underscore:

```
// Valid senders message
let valid_message = [
    transfer_data.eth_recipient.as_ref(),
    b"_",
    transfer_data.amount.to_le_bytes().as_ref(),
    b"_",
    transfer_data.id.as_ref(),
    b"_",
    bot_oracle.eth_address.as_ref(),
]
.concat();
```

This format is inherently prone to ambiguities. Consider the example of the following `amount` and `id` variations (other fields left out for simplicity):

```
amount:   123
id:           _myid
message:  123__myid

amount:   123_
id:            myid
message:  123__myid
```

The same message can be obtained by composing different `amounts` and `ids`.

Impact

This issue can potentially be exploited to submit manipulated values to invocations of `process_evaluate_attestations`. The Audius team claimed `amounts` and `ids` containing underscores (`0x5f` bytes) cannot be generated by the relevant off-chain programs;

therefore, the issue is not exploitable in practice. For this reason this potentially critical issue is reported as informational.

Recommendations

Even though the issue might not be exploitable at the time of this security audit, we strongly advise to review the message format to make ambiguities impossible in order to to harden the code and avoid being exposed to a risk of a critical issue. One remediation option would be to adopt a serialization format where the various fields have a fixed length. Another more flexible (but more complex and bug-prone) option would be to adopt a tag-length-value encoding (or just length-value).

Remediation

The Audius team acknowledged this finding. No change to the codebase was deemed to be immediately required.

3.3 Unsafe account deletion method

- **Target**: Rewards Manager
- **Category**: Coding Mistakes
- **Likelihood**: N/A
- **Severity**: Low
- **Impact**: Low

Description

The `EvaluateAttestations` instruction processes an account of type `VerifiedMessages` containing signed assertions authorizing the transfer of a given amount of tokens to a specific account. Towards the end of the instruction, the `VerifiedMessages` account is deleted by zeroing its lamports.

This account deletion method is unsafe and prone to abuse. The reason is that account deletion does not happen immediately after an instruction is finished processing, and a zero-lamports account is usable by other instructions within the same transaction.

Impact

It is possible to reuse a `VerifiedMessages` account after an `EvaluateAttestations` instruction has been processed, despite it having zero lamports, by referencing the same account in multiple instructions within the one transaction. This issue was part of the exploit for issue 3.1.

Recommendations

Invalidate or immediately delete `VerifiedMessages`.

Invalidating the account can be done by zeroing the `version` field, thus making `unpacking` the account fail.

Truly and fully deleting the account is not possible; however, it is possible to achieve an equivalent effect by zeroing the account lamports, resizing the account to zero, and transferring the account ownership to the system program.

Remediation

The Audius team was alerted of this issue while the audit was ongoing, together with issue 3.1. The Audius team quickly applied a remediation that invalidates the account, making `unpack` fail.

3.4 Lack of parameters validation in `InitRewardManager`

- **Target**: Rewards Manager
- **Category**: Code Maturity
- **Likelihood**: N/A
- **Severity**: Informational
- **Impact**: Informational

Description

The processor for the `InitRewardManager` is not performing some checks that would prevent misuse of the program outside of the intended functionality.

Specifically, the `min_votes` parameter is not required to be greater than zero. It would be possible to initialize a `RewardManager` that requires zero signers. In addition, the `mint_info` account is not constrained to be the mint of one specific token.

Impact

This is an informational finding, and there is no direct security impact. The off-chain programs invoking `InitRewardManager` are responsible for calling it with appropriate parameters and could potentially invoke it with invalid parameters by mistake, creating a `RewardManager` that requires no signers.

Recommendations

Even though this issue does not pose a direct security vulnerability, we recommend to be as restrictive as possible in the inputs accepted by on-chain programs as a hardening measure.

Remediation

The Audius team acknowledged this finding. No change to the codebase was deemed to be immediately required.

4 Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment.

4.1 Ownership change process

The reward manager `ChangeManagerAccount` instruction allows to change the owner associated with a reward manager. The ownership change process does not require to prove that the new owner account is valid and that the associated private key is known (or that a program that can sign for it exists, in the case of a PDA). A two-step process is quite common to ensure protection against mistaken ownership changes: First, an ownership change is requested, and the to-be admin is stored alongside the old one. A second instruction is then invoked with the second admin as a signer, confirming the ownership transfer and replacing the former admin with the new one. Such a design change could be a minor improvement to Audius design.

4.2 Usage of constant strings instead of enums

The reward manager `validate_secp_add_delete_sender` function takes a `message_prefix` string reference, which in practice is always one of the two constant values `DELETE_SENDER_MESSAGE_PREFIX` or `ADD_SENDER_MESSAGE_PREFIX`. A very minor improvement to be considered would be to switch from accepting a string reference to accepting an enum implementing the `From` and `Into` traits to aid in converting to and from a string form.

4.3 Lack of discriminators

Solana accounts are, by default, untyped byte buffers with some associated metadata. It is up to the individual programs to discern between the various types of data that can be stored in an account. Audius determines the type of an account by looking at the size of the account's data. Since every type in Audius programs differs in size, this is possible and not ambiguous. However, this prevents Audius from introducing account types with the same size in a future update.

The most common and general approach, also adopted by Anchor, the most common Solana framework, is to include a discriminator field in the account data, containing a value that is unique for each type of account. This approach allows to introduce new

types (even with identical sizes) without the risk of type confusion.

Audius is not vulnerable to account confusion in the version under review. We note that extreme care by the development team must be taken not to introduce new account types with size conflicting with any other existing type in the future.

4.4 The checks `is_signer` and `owner` should be consolidated

In multiple places in the eth registry are seperate `is_signer` and `owner` checks performed on the Signer Group. The Signer Group does have the `SignerGroup::check_owner( ... )` method, which does both already with the correct error return variants. These scattered `is_signer` and `owner` checks should be consolidated into `check_owner` calls.

4.5 No account ownership enforced in rewards manager change manager account instruction

The Solana VM will enforce the property that a program must be the owner account it has performed a write to. This property is commonly leveraged to avoid having to perform an explicit ownership check inside the contract code itself as the Solana VM should enforce this.

This is actually not entirely true. This property is only enforced if there is *a change* to the account data field. So, if the Solana program writes *the same* data that already exists in the account on top of the existing data, then the Solana runtime will not require the program to have ownership of the account, since it's data before and after the program executes hasn't changed.

This is typically an issue if the account being written to, or one of the arguments passed in via the instruction, is also being used to perform a write to another important account. In this case, that is not true; therefore, this did not rise to the level of a security issue. We wanted to note, though, that the following comment found in the code does not always hold for the reason explained above:

```
// Note: We do not have to assert that we own the `reward_manager` account
// as we would normally, because in writing to it the runtime
// enforces ownership
```

5 Audit Results

During our audit, we discovered four findings. Of these, one was of critical severity, one was low risk, and two were suggestions (informational). Audius, Inc acknowledged all findings and implemented fixes for the critical and low severity findings.

At the time of our audit, the code was deployed to mainnet Solana. The Audius team was informed of the critical finding as soon as Zellic confirmed exploitability by perfecting a proof of concept. The Audius team promptly acknowledged, triaged, and remediated the issue, minimizing the risk of the issue being exploited.

5.1 Disclaimers

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 additional code added to the assessed project after the audit version of 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 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.

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.