Security Audit Report

Push Protocol – Comm Rust

v1.0

December 6, 2024

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by Push Comm Ltd to perform a security audit of Push Protocol – Comm Rust.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/push-protocol/push-comm-rust
Commit	`54aafbe5560be9d62e264321e70e713ec3f1ea16`
Scope	All contracts were in scope.
Fixes verified at commit	`781afaaf98e09331ede5d5449d713a981a58bd98` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Push Protocol is a Web3 communication protocol that enables any dApps, smart contracts, backends, or protocols to communicate both on-chain and off-chain via user wallet addresses in an open, gasless, multichain, and platform-agnostic fashion.

The Push Comm Rust contract includes features that allow users to subscribe to a channel and unsubscribe from a channel as well as sending notifications as a channel's delegate.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending, Acknowledged**, **Partially Resolved**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	**Low-Medium**	-
Code readability and clarity	**Medium**	The code is mostly self-explanatory but lacks code comments providing function documentation and explaining business logic.
Level of documentation	**Medium**	The documentation describes the fundamental flow and functionality of the program with some technical details, but lacks description of the pausing mechanism.
Test coverage	**Medium-High**	Comprehensive unit test coverage, but lack of integration testing with off-chain components.

Summary of Findings

No	Description	Severity	Status
1	Inadequate access control for delegate subscriptions may enable DoS or griefing attacks	Major	Resolved
2	Possible protocol dysfunction due to hard limits on log size	Major	Acknowledged
3	Inability to modify existing notification settings	Major	Resolved
4	Notifications settings are not cleared when a user unsubscribes from channel	Minor	Acknowledged
5	Potential compatibility issue regarding future upgrades	Minor	Partially Resolved
6	Admin functionality can be permanently disabled	Minor	Resolved
7	Inconsistency in authorization logic between EVM and Solana contracts	Minor	Acknowledged
8	Inaccurate maximum length of notification settings	Informational	Acknowledged
9	Pausing functionality lacks documentation and clarity	Informational	Acknowledged
10	Missing access control of `initialize` function	Informational	Resolved
11	The `chain_cluster` value cannot be modified post-initialization	Informational	Resolved
12	Unvalidated `chain_cluster` string size	Informational	Acknowledged
13	The used Anchor version has vulnerable dependencies	Informational	Resolved
14	Miscellaneous comments	Informational	Resolved

Detailed Findings

1. Inadequate access control for delegate subscriptions may enable DoS or griefing attacks

Severity: Major

In the current design, the EVM implementation restricts subscription management to the user. However, on Solana, the `add_delegate` function allows anyone to add subscriptions on behalf of any user by designating the user as a delegate to an arbitrary channel.

Standard subscriptions use the seeds `SUBSCRIPTION + signer + channel`, while delegated subscriptions use `SUBSCRIPTION + delegate + signer`. As the delegate parameter is passed without validation, an attacker can impersonate any user by setting them as a delegate, bypassing user control and initiating unwanted subscriptions.

By exploiting this, an attacker could create accounts for other users, which allows them to set user-specific fields, such as `user_start_block`. This could lead to inaccurate data, affecting system integrity.

An attacker may use this vulnerability to subscribe a target user to multiple fake channels, causing either annoyance (griefing) or a form of denial-of-service by overloading the target's subscription list.

Recommendation

We recommend implementing a two-step delegation process where the delegate must confirm or co-sign the delegation request before it becomes active. This approach will prevent unauthorized subscriptions and ensures that only verified delegations are recorded, enhancing security and data integrity.

Status: Resolved

2. Possible protocol dysfunction due to hard limits on log size

Severity: Major

The `send_notification` instruction defined in `programs/push_comm/src/lib.rs` emits the `SendNotification` log that contains the message of an arbitrarily chosen size. However, the Solana ecosystem imposes a 10 kB restriction on log size for the whole execution. This means that all of the instructions processed within the transaction have a total of 10 kB of logs to emit.

Should this limit be exhausted, the transaction can still finish execution successfully, but the logs will be truncated. Consequently, the message, or parts of it, may be irrecoverably lost.

Recommendation

We recommend using Anchor's `emit_cpi` macro instead of the `emit` one so that events are converted to instructions and, as such, are prevented from being truncated.

Status: Acknowledged

3. Inability to modify existing notification settings

Severity: Major

The `set_user_notification_settings` instruction creates the `UserNotificationSettings` account to store data related to the settings. However, in that instruction, the `UserNotificationSettings` account is marked with the `init` macro only. No instruction contains this account with `mut` or `init_if_needed` macros.

Consequently, once a particular `UserNotificationSettings` account is created, there is no way to change its content. This results in very limited functionality for the protocol's users.

Recommendation

We recommend introducing another instruction that allows users to change the data present in their `UserNotificationSettings` account.

Status: Resolved

4. Notifications settings are not cleared when a user unsubscribes from channel

Severity: Minor

In `programs/push_comm/src/lib.rs:126`, the channel-specific `notif_settings` are written within the `set_user_notification_settings` function according to the user's specified settings.

However, when a user unsubscribes from a communication channel, the corresponding notification settings are never cleared or reset.

Consequently, when a user unsubscribes from a channel and re-subscribes at a later point in time, the prior notification settings still persist which might be unexpected.

Recommendation

We recommend clearing the corresponding notification settings when a user unsubscribes, i.e. restore the defaults.

Status: Acknowledged

5. Potential compatibility issue regarding future upgrades

Severity: Minor

The `PushCommStorageV3` account is the core data account holding the general configuration for the protocol. However, the Anchor framework uses the struct name to calculate a discriminator - the first 8 bytes of account data used during deserialization to determine the kind of account used in instructions.

Should the struct name change, for example, to `PushCommStorageV4`, those accounts would not be compatible, and separate instructions would need to be implemented to handle the V4 account.

Similarly, the `PUSH_COMM_STORAGE` seed is set to be `b"push_comm_storage_v3"`, which introduces additional complexity in future changes.

Recommendation

We recommend changing the `PushCommStorageV3` structure name to a generic one, e.g. `PushCommStorage`, and introducing a field inside the struct corresponding to the version. Deserializing and potential account resizing should be implemented based on this additional field. Similarly, the seeds used for the account's address derivation should not depend on the version.

Status: Partially Resolved

6. Admin functionality can be permanently disabled

Severity: Minor

The Solana implementation of `transfer_admin_ownership` allows setting `push_channel_admin` to zero, effectively disabling the "admin" functionality. However, the EVM version does not permit this. This represents a significant difference in functionality between the two implementations.

Recommendation

We recommend aligning the behavior of `transfer_admin_ownership` across both Solana and EVM versions. Either allow setting `push_channel_admin` to zero in both implementations or prevent it in both. This will ensure consistency and avoid unexpected behavior when migrating or interacting with the contract on different chains.

Status: Resolved

7. Inconsistency in authorization logic between EVM and Solana contracts

Severity: Minor

The Solana version of the `send_notification`, which is similar to the `_checkNotifReq` function in the EVM version, has a check which is missing the `push_channel_admin` authorization logic. This discrepancy could lead to inconsistent behavior, as the Solidity version grants the `pushChannelAdmin` special privileges to send notifications, while the Solana version does not.

Recommendation

We recommend implementing one of the following:

- Align the Solana version's functionality by adding the `push_channel_admin` authorization logic.
- Document the intentional difference in authorization logic between the EVM and Solana versions, clearly outlining the specific privileges of the `push_channel_admin/pushChannelAdmin` in each.

Status: Acknowledged

8. Inaccurate maximum length of notification settings

Severity: Informational

The program currently expects `notif_settings` to support up to 100 characters, as defined by `MAX_NOTIF_SETTINGS_LENGTH`. Such a condition is expressed in a check in the `set_user_notification_settings` function.

However, due to the formatting requirements, i.e. `notif_id + "+" + notif_settings`, the effective capacity for `notif_settings` is approximately 90 characters, depending on the length of `notif_id`. This discrepancy prevents users from utilizing the full expected 100-character length.

Recommendation

We recommend adjusting `MAX_NOTIF_SETTINGS_LENGTH` to align with the actual maximum length available for `notif_settings`.

Status: Acknowledged

9. Pausing functionality lacks documentation and clarity

Severity: Informational

The `pause` functionality in the contract is currently undocumented, making it unclear which actions should be restricted during a paused state. Presently, the `pause_contract` function prevents the execution of certain functions, such as `transfer_admin_ownership` and `verify_channel_alias`, but it does not restrict user actions like `subscribe` and `unsubscribe`.

This can lead to ambiguity in contract behavior and may not achieve the intended level of security control during a paused state.

Recommendation

We recommend expanding the `pause` functionality to restrict all user interactions or clearly document which functions remain accessible during a paused state.

Status: Acknowledged

10. Missing access control of `initialize` function

Severity: Informational

The `initialize` function lacks access control allowing any user to set the `governance` and `push_channel_admin`. This poses a frontrunning opportunity as the deployment script does not initialize the contract in the same transaction. An attacker could frontrun the legitimate initialization and set a malicious `push_admin`.

Recommendation

We recommend implementing access control within the `initialize` function to ensure only authorized parties can set the initial `governance` and `push_channel_admin`. Also, consider combining deployment and initialization into a single transaction to prevent frontrunning.

Status: Resolved

11. The `chain_cluster` value cannot be modified post-initialization

Severity: Informational

The `chain_cluster` value, currently set during the initialize function, is immutable within the current contract design. If a blockchain network undergoes a name change—such as the anticipated transition from `mainnet-beta` to `mainnet`, there would be no way to update the `chain_cluster` value accordingly. This lack of flexibility could limit the contract's adaptability in the case of network renaming or future chain forks.

Recommendation

We recommend implementing a function that allows the contract admin to update the `chain_cluster` value post-initialization.

Status: Resolved

12. Unvalidated `chain_cluster` string size

Severity: Informational

The `chain_cluster` parameter in the `initialize` function is currently defined as a `String`. Although the storage account allocates only `20` bytes for chain_cluster, there is no validation to ensure that the input stays within this limit. Without this validation, users may unknowingly input a longer `chain_cluster` value, leading to transaction reverts when the storage allocation is exceeded.

Currently, the program has no capability to accept or store a `chain_cluster` string with more than 20 characters, so any input beyond this length will fail.

Recommendation

Consider using an enum type to define `chain_cluster`, with specific options such as `Devnet`, `Testnet`, or `Mainnet-Beta`. This will enforce fixed, bounded values and eliminate length-related reversion risks.

Status: Acknowledged

13. The used Anchor version has vulnerable dependencies

Severity: Informational

The used Anchor version 0.29.0 has dependencies that are affected by vulnerabilities such as:

- `curve25519-dalek` which is affected by [RUSTSEC-2024-0344]
- `proc-macro-error` which is affected by [RUSTSEC-2024-0370]
- `borsh` which is affected by [RUSTSEC-2023-0033]

Recommendation

We recommend updating Anchor to the latest version.

Status: Resolved

14. Miscellaneous comments

Severity: Informational

Miscellaneous recommendations can be found below.

Recommendation

The following are some recommendations to improve the overall code quality and readability:

- We recommend adding in-line documentation/Natspec to all functions, especially complex ones like `set_user_notification_settings`, to improve readability.
- We recommend removing unused `borsh` import in `programs/push_comm/src/lib.rs:2`.
- We recommend fixing the development environment by removing hardcoded development key in `Anchor.toml:15`.
- Insufficient role segregation at initialization, in `programs/push_comm/src/lib.rs:25-26` both storage variables `governance` as well as `push_channel_admin` are assigned to the same `push_admin`. We recommend assigning separate keys for these roles already at initialization.
- When a user subscribes to a channel and does not explicitly set notifications settings, the corresponding `UserNotificationSettings` account does not exist. We recommend initializing a default `UserNotificationSettings` account on subscription.
- We recommend closing the corresponding `DelegatedNotificationSenders` account when removing a delegate and therefore recovering the account's rent exemption balance.
- We recommend closing the corresponding `Subscription` account on unsubscribing and therefore recovering the account's rent exemption balance.
- The `notif_id` field of the `UserNotifcationSettingsAdded` structure defined in `programs/push_comm/src/events.rs:45` is of type `u64`. The corresponding EVM implementation uses a `uint256` type for this field, therefore we recommend reconsidering the expected numerical range and the respective choice of data type.
- The `verify_channel_alias` function implements a verification that the provided `channel_address` is at most 64 characters long. This limit does not correspond with the Ethereum address format of 20 bytes-long addresses. We recommend

changing the limit to one that corresponds to the format used for `channel_address`.

- The `governance` data field present in the `PushCommStorageV3` is not documented and appears not to have any meaningful on-chain functionality within the program. We recommend either deleting it if it is not needed or documenting it.
- The `push_token_ntt` data field present in the `PushCommStorageV3` is not documented and appears not to have any meaningful on-chain functionality within the program. We recommend either deleting it if it is not needed or documenting it.
- The Solana version of `add_delegate` checks if a delegate is already added and throws the `DelegateAlreadyAdded` error if so. However, the EVM/Cairo versions follow a "perform actions as needed but do not revert if conditions are unmet" approach effectively ignoring this check. This inconsistency in error handling is also present in `remove_delegate` concerning the `DelegateNotFound` error. We recommend aligning the Solana version with the EVM/Cairo approach.

Status: Resolved