

Sablier Token Distribution

Security Review

Cantina Managed review by:

Phaze, Lead Security Researcher
Rikard Hjort, Lead Security Researcher

August 31, 2025

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**
- 2.1 Overall Assessment . . . 3

3 Findings **4**
- 3.1 Medium Risk . . . 4
  - 3.1.1 Incorrect cliff time handling in streaming start calculation . . . 4
- 3.2 Low Risk . . . 4
  - 3.2.1 Cancelled streams continue accruing withdrawable amounts over time . . . 4
  - 3.2.2 Potential overflow in fee calculation for non-standard oracle decimals . . . 6
- 3.3 Gas Optimization . . . 7
  - 3.3.1 Redundant seed checks for Metaplex accounts . . . 7
- 3.4 Informational . . . 8
  - 3.4.1 The program does not verify campaign argument correctness . . . 8
  - 3.4.2 Timestamps are not validated . . . 8
  - 3.4.3 Program initialization can be front-run by malicious actors . . . 9
  - 3.4.4 Account creation can be blocked through predictable address pre-creation . . . 9
  - 3.4.5 Use of deprecated Metaplex token metadata functions . . . 10
  - 3.4.6 NFT names can be mined to match existing streams for potential phishing . . . 11
  - 3.4.7 IPFS CID omitted from campaign account seeds . . . 12
  - 3.4.8 Code quality issues . . . 13

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

The Sablier Protocol is a collection of persistent, non-upgradeable smart contracts to facilitate streaming of ERC-20 tokens on Ethereum and other EVM blockchains.

From Aug 4th to Aug 13th the Cantina team conducted a review of solsab on commit hash 2883177b. The team identified a total of **12** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	2	2	0
Gas Optimizations	1	0	1
Informational	8	2	6
Total	**12**	**5**	**7**

The Cantina Managed team reviewed Sablier Finance's `solsab` on commit `9ededd` and concluded that all findings were addressed and no new issues were identified.

2.1 Overall Assessment

The Cantina Managed team has reviewed Sablier Finance's new Solana deployment and found it largely correct and in line with their current EVM deployments.

One accounting error (Incorrect cliff time handling in streaming start calculation) was discovered and mitigated. One further calculation error (Cancelled streams continue accruing withdrawable amounts over time) would have lead to a user experience issue if not uncovered, and made it difficult for a non-technical user to withdraw their vesting after a vesting schedule is cancelled. One potential overflow in fee calculations (Potential overflow in fee calculation for non-standard oracle decimals) was discovered, which would have blocked withdrawals – however it is very unlikely that the bug would have presented itself. Further findings were informational.

All of the above described issues have been fixed.

Users are encouraged to read finding "NFT names can be mined to match existing streams for potential phishing". It describes a potential issue where a malicious user may try to trick end users of NFT wallets. All non-technical users are encouraged to use Sablier's own interface to interact with the protocol.

3 Findings

3.1 Medium Risk

3.1.1 Incorrect cliff time handling in streaming start calculation

Severity: Medium Risk

Context: lockup_math.rs#L33-L38

Summary: The streaming start time calculation in the lockup contract incorrectly uses the cliff unlock amount instead of the cliff timestamp to determine when streaming should begin. This causes tokens to be unlocked too early when a stream has a cliff time configured but no cliff unlock amount, violating the intended vesting schedule.

Description: The `get_streamed_amount()` function in `lockup_math.rs` incorrectly determines the streaming start time by checking the `cliff_unlock` amount instead of the `cliff` timestamp. This causes issues when a stream has a cliff time configured but no cliff unlock amount. The current logic at line 34-38:

```
let start_time: i64 = if amounts.cliff_unlock == 0 {
    timestamps.start
} else {
    timestamps.cliff
};
```

This approach uses the cliff unlock amount as a proxy for determining whether streaming should start at the start time or cliff time. However, it's possible to have a valid stream configuration where:

- A cliff time is set (`timestamps.cliff > 0`).
- No tokens are unlocked at the cliff (`amounts.cliff_unlock == 0`).
- Some tokens may be unlocked at start (`amounts.start_unlock_amount > 0`).

In this scenario, the streaming should wait until the cliff time before beginning, but the current logic would incorrectly start streaming immediately because `cliff_unlock == 0`.

Impact Explanation: The impact is medium to high as tokens are unlocked earlier than intended, potentially violating vesting agreements and allowing recipients to access funds before the configured cliff period. While this doesn't result in total loss of funds, it breaks the core timing guarantees of the streaming protocol.

Likelihood Explanation: The likelihood is medium as this affects any stream configured with a cliff time but zero cliff unlock amount. While this may be a less common configuration, it is explicitly supported by the validation logic and represents a legitimate use case for vesting schedules.

Recommendation: Update the streaming start time calculation to check the cliff timestamp instead of the cliff unlock amount:

```
-   let start_time: i64 = if amounts.cliff_unlock == 0 {
+   let start_time: i64 = if timestamps.cliff == 0 {
      timestamps.start
  } else {
      timestamps.cliff
  };
```

Sablier: Fixed in PR 240.

Cantina Managed: Fix verified.

3.2 Low Risk

3.2.1 Cancelled streams continue accruing withdrawable amounts over time

Severity: Low Risk

Context: cancel.rs#L106-L119

Summary: The stream cancellation mechanism in the Sablier lockup contract contains a vulnerability where canceled streams continue to accrue withdrawable amounts over time instead of being frozen at the cancellation state. This occurs because the `get_streamed_amount()` function does not account for

canceled or depleted stream states, allowing recipients to attempt withdrawals of amounts that were already refunded to the sender.

Description: When a stream is canceled through the `cancel()` function, the contract correctly:

1. Calculates the streamed amount at the time of cancellation.
2. Refunds the unstreamed portion to the sender.
3. Marks the stream as canceled (`was_canceled = true`).
4. Sets `amounts.refunded` to track the refunded amount.

However, the `get_streamed_amount()` function in `lockup_math.rs` continues to calculate increasing streamed amounts over time without checking if the stream was canceled or depleted. This leads to several issues:

1. The `withdrawable_amount` continues to grow after cancellation based on time progression.
2. Recipients can attempt to withdraw more than the amount that should be available.
3. While `transfer_tokens()` will ultimately fail due to insufficient balance in the stream account, this creates poor user experience and allows transaction fees to be wasted.

Consider this scenario:

- Creator deposits 100 tokens.
- Stream is 20% complete (streamed_amount = 20, recipient hasn't withdrawn).
- Stream is canceled: sender gets 80 tokens refunded, 20 tokens remain for recipient.
- Time passes and streamed amount calculation reaches 40 tokens.
- Recipient attempts `withdraw_max()` expecting 40 tokens but only 20 are available.

Impact Explanation: The impact is low as normal withdrawals can still be used with the correct capped amount. While the `withdraw_max()` function may attempt to withdraw more than available, individual withdrawals with specific amounts will work properly up to the actual available balance. The main issues are poor user experience and perhaps wasted transaction fees.

Likelihood Explanation: The likelihood is medium as this will occur when streams are canceled and recipients subsequently use `withdraw_max()` rather than individual withdrawals, or when `withdrawable_amount_of()` is checked. Users can work around the issue by using normal withdraw functions with appropriate amounts, though they may not be aware of this limitation.

Recommendation: Modify the `get_streamed_amount()` function to properly handle canceled and depleted stream states:

```
-   pub fn get_streamed_amount(timestamps: &Timestamps, amounts: &Amounts) -> u64 {
+   pub fn get_streamed_amount(timestamps: &Timestamps, amounts: &Amounts, is_depleted: bool, was_canceled:
↪  bool) -> u64 {
+     // If the stream is depleted, return the withdrawn amount
+     if is_depleted {
+         return amounts.withdrawn;
+     }
+
+     // If the stream was canceled, return deposited minus refunded amount
+     if was_canceled {
+         return amounts.deposited - amounts.refunded;
+     }

      let now = Clock::get().unwrap().unix_timestamp;

      // If the start time is in the future, return zero.
      if timestamps.start > now {
          return 0;
      }

      // ... rest of existing logic
  }
```

Sablier: Fixed in PR 240.

Cantina Managed: Fix verified.

3.2.2 Potential overflow in fee calculation for non-standard oracle decimals

Severity: Low Risk

Context: fee_calculation.rs#L63

Summary: The fee calculation function in the Sablier lockup contract contains a potential integer overflow when processing Chainlink oracle feeds with non-standard decimal formats. While current Chainlink feeds use 8 decimals, the code attempts to handle arbitrary decimal counts using u64 arithmetic, which can overflow for larger decimal values and high USD fees.

Description: The `convert_usd_fee_to_lamports()` function calculates withdrawal fees by converting USD amounts to lamports using Chainlink price feeds. For non-8-decimal feeds, the calculation uses:

```
fee_usd * 10_u64.pow(1 + decimals as u32) / price
```

This calculation can overflow when using u64 arithmetic. For example:

- `fee_usd = 2e8` (representing $2 with 8 decimals).
- `decimals = 12`.
- `fee_usd * 10^(12+1) = 2e8 * 10^13 = 2e21`.

Since the maximum value for u64 is approximately 1.8e19, this calculation would overflow and cause the transaction to panic.

While the protocol expects to use 8-decimal Chainlink feeds, the code includes logic to handle other decimal formats, creating a potential vulnerability if non-standard feeds are ever used or if Chainlink changes their decimal format in the future.

Impact Explanation: The impact is medium as an overflow would cause withdrawal transactions to fail completely, preventing users from accessing their funds. However, the issue only affects scenarios with non-standard oracle decimal formats, which are not currently used by Chainlink.

Likelihood Explanation: The likelihood is low as current Chainlink SOL/USD feeds use 8 decimals, and the protocol team intends to only use 8-decimal feeds. The overflow would only occur if feeds with higher decimal counts were used, which is unlikely given current Chainlink standards.

Recommendation: Consider using u128 arithmetic for the fee calculation to handle arbitrary decimal formats safely:

```
-   let fee_in_lamports: u64 = match oracle_decimals {
+   let fee_in_lamports: u128 = match oracle_decimals {
      8 => {
          // If the oracle decimals are 8, calculate the fee.
-         fee_usd * LAMPORTS_PER_SOL / price
+         fee_usd as u128 * LAMPORTS_PER_SOL as u128 / price as u128
      }
      decimals => {
          // Otherwise, adjust the calculation to account for the oracle decimals.
-         fee_usd * 10_u64.pow(1 + decimals as u32) / price
+         fee_usd as u128 * 10_u128.pow(1 + decimals as u32) / price as u128
      }
  };

+   // Safely convert back to u64, capping at u64::MAX if needed
+   fee_in_lamports.min(u64::MAX as u128) as u64
```

This approach provides safety against overflow while maintaining the current behavior for 8-decimal feeds and handling edge cases where the calculated fee exceeds u64 limits.

Sablier: Fixed in PR 240.

Cantina Managed: Fix verified.

Note: only added `u128` conversion in the arbitrary decimals case. But $\frac{\text{u64::MAX}}{\text{LAMPORTS_PER_SOL}} \approx \frac{2^{64}}{10^9} \approx 1.84 \times 10^{10} = 184 \times 10^8$ so for the hardcoded fees of 1×10^8 and 2×10^8, it won't overflow.

3.3 Gas Optimization

3.3.1 Redundant seed checks for Metaplex accounts

Severity: Gas Optimization

Context: create_with_timestamps.rs#L58-L83

Description: The stream creation instruction includes explicit seed validation constraints for Metaplex token metadata accounts (`nft_collection_master_edition` and `nft_collection_metadata`) that may be redundant since the Metaplex program itself already performs these validations during execution. The seed constraints at lines 58-83 in `create_with_timestamps.rs`:

```
#[account(
    seeds = [
        METADATA,
        token_metadata_program.key().as_ref(),
        nft_collection_mint.key().as_ref(),
        EDITION,
    ],
    seeds::program = token_metadata_program.key(),
    bump,
)]
pub nft_collection_master_edition: UncheckedAccount<'info>,

#[account(
    mut,
    seeds = [
        METADATA,
        token_metadata_program.key().as_ref(),
        nft_collection_mint.key().as_ref(),
    ],
    seeds::program = token_metadata_program.key(),
    bump,
)]
pub nft_collection_metadata: UncheckedAccount<'info>,
```

While these constraints don't cause functional issues, they duplicate validation logic that the Metaplex program already enforces. The redundant checks provide some benefits including earlier failure detection and clearer error messages, but they also add computational overhead during instruction context validation.

Recommendation: Consider removing the redundant seed constraints for optimization:

```
-   #[account(
-       seeds = [
-           METADATA,
-           token_metadata_program.key().as_ref(),
-           nft_collection_mint.key().as_ref(),
-           EDITION,
-       ],
-       seeds::program = token_metadata_program.key(),
-       bump,
-   )]
+   #[account()]
  /// CHECK: This account will only be touched by the Metaplex program
  pub nft_collection_master_edition: UncheckedAccount<'info>,

-   #[account(
-       mut,
-       seeds = [
-           METADATA,
-           token_metadata_program.key().as_ref(),
-           nft_collection_mint.key().as_ref(),
-       ],
-       seeds::program = token_metadata_program.key(),
-       bump,
-   )]
+   #[account(mut)]
  /// CHECK: This account will only be touched by the Metaplex program
  pub nft_collection_metadata: UncheckedAccount<'info>,
```

This optimization reduces the computational overhead of instruction context validation while maintaining security, as the Metaplex program will still enforce proper account relationships during execution.

Sablier: Acknowledged. If we remove the seed constraints from these accounts, they would, then, have to be passed explicitly (and not derived automatically) when calling the respective Instruction.

```
 .accounts({
   creator: this.sender.keys.publicKey,
   depositTokenMint: this.usdc,
   depositTokenProgram: token.TOKEN_PROGRAM_ID,
   nftTokenProgram: token.TOKEN_PROGRAM_ID,
+  nftCollectionMasterEdition: this.getNftCollectionMasterEdition(),
+  nftCollectionMetadata: this.getNftCollectionMetadata(),
   recipient: this.recipient.keys.publicKey,
   sender: this.sender.keys.publicKey,
 })
```

Cantina Managed: Acknowledged.

3.4 Informational

3.4.1 The program does not verify campaign argument correctness

Severity: Informational

Context: clawback.rs#L74, create_campaign.rs#L15-L121

Description: The program does not verify that the Merkle root is correct -- which is sensible. But it should be performed on the frontend to avoid fake and scam campaigns.

For example, a malicious user can create a fake campaign with a well-known token and a large `aggregate_amount`. This encourages spam -- as it might be listed high on a leaderboard or similar -- and can crowd out visibility of real airdrops, as well as creating fake legitimacy for scam airdrops.

Currently, the team has described the frontend as presenting airdrops as they are, with the assumption that they are only created through their frontend.

Recommendation: It is easy to check if a campaign is valid and/or trustworthy in three main dimensions:

1. Is the campaign Merkle root a true representation of a real tree with valid leaves with matching `aggregate_amount`, `recipient_count`, etc.?
2. Is the campaign fully funded, based on the total amounts in the Merkle tree?
3. Is the campaign currently in a state where it can be clawed back at any time?

The frontent can calculate the Merkle root from the IPFS data and check the correctness of the on-chain Merkle root, and determine whether or not the campaign is fully funded. This could be used to filter out fake or problematic airdrops and give the users relevant information on which claims may fail due to a lack of funds, and encourage proper campaign creation.

In that vein, it could also be sensible to add a boolean indicating whether or not the campaign has been clawed back.

Sablier: Acknowledged. After discussing with our frontend team, we found out that they perform multiple checks to make sure that a campaign is correct. Therefore, we believe that adding extra data to the campaign on-chain storage wouldn't make the verification process more secure, but would only increase the associated gas costs.

Cantina Managed: Acknowledged.

3.4.2 Timestamps are not validated

Severity: Informational

Context: campaign.rs#L39-L60

Description: Timestamps are not validated on campaign creation. They could be less than the current timestamp or even negative. While this seems to cause no direct bugs -- they are simply "in the past", or, in the case of `expiration_time` being `0`, treatead as infinity (no expiry) -- it is an unnecessary complexity to allow.

It may also cause strange UI behavior, since the UI would have to validate and correctly deal with negative timestamps.

Recommendation: Check on creation that the campaign timestamps are `>= 0`.

Sablier: Fixed in PR 249. After some consideration, we decided to switch to `u64` timestamps in both `MerkleInstant` and `Lockup`.

Cantina Managed: It could make sense to validate that timestamps are not in the past.

Sablier: Yes, this is the intended behaviour - we are ok with allowing timestamps in the past.

Cantina Managed: Fix verified.

3.4.3 Program initialization can be front-run by malicious actors

Severity: Informational

Context: initialize.rs#L127-L132, initialize.rs#L38-L43

Description: The Sablier lockup and merkle_instant program's initialization functions lack access control, allowing any user to initialize the programs with arbitrary parameters, potentially setting malicious fee collectors or oracle configurations.

The `initialize()` function accepts critical configuration parameters including:

- `fee_collector`: The address that receives withdrawal fees.
- `chainlink_program`: The Chainlink program address for price feeds.
- `chainlink_sol_usd_feed`: The SOL/USD price feed address.

Since the function has no access controls and can only be called once, a malicious actor monitoring transactions could front-run the legitimate initialization with their own parameters. This would allow them to:

1. Set themselves as the fee collector to steal all withdrawal fees.
2. Provide a malicious oracle feed to manipulate fee calculations.
3. Configure incorrect Chainlink program addresses.

While the legitimate deployers would notice the failed initialization and could redeploy, this creates operational risks and potential for silent failures if deployment scripts are not robust enough to detect the front-running.

Recommendation: Consider adding access control to the initialization function by checking that the initializer is the upgrade authority.

Additionally, ensure deployment scripts include robust verification that the initialization was performed with the correct parameters and by the intended authority.

Sablier: Acknowledged. We agree that `initialize()` is front-runnable, in theory. In practice, however, we initialize the programs right after deploying them, which happens in a single script. Besides, a failure in any of these 2 steps is clearly reported in the console logs of the script. In case of a failure, we'll just redeploy & re-initialize the programs (potentially, via a fresh deployer address). Given the one-time nature of the initialization, the very small race window and our operational checks, we accept this risk and plan no code changes.

Cantina Managed: Acknowledged.

3.4.4 Account creation can be blocked through predictable address pre-creation

Severity: Informational

Context: create_with_timestamps.rs#L100-L124, create_campaign.rs#L56-L63

Description: The stream creation process in the Sablier lockup contract uses deterministic addresses for creating accounts, which can be exploited by attackers to block legitimate transactions through preemptive account creation. Specifically, the `recipient_stream_nft_ata` account creation in `create_with_timestamps.rs` uses predictable parameters:

```
#[account(
    init,
    payer = creator,
    associated_token::mint = stream_nft_mint,
    associated_token::authority = recipient,
    associated_token::token_program = nft_token_program,
)]
pub recipient_stream_nft_ata: Box<InterfaceAccount<'info, TokenAccount>>,
```

If an attacker can predict the `sender`, `salt`, and `recipient` parameters, they can pre-create the associated token account at the expected address, causing the legitimate stream creation transaction to fail when it attempts to initialize an already-existing account.

A similar issue exists in the merkle instant program where the `campaign_ata` account uses deterministic addressing based on campaign parameters that might be discoverable before the official campaign launch.

While the likelihood is reduced by the ability to use unpredictable salts and retry with different parameters, the potential for denial of service exists when account addresses can be anticipated.

Recommendation: Consider using `init_if_needed` instead of `init` for accounts that could potentially be pre-created by attackers:

```
  /// Create account: the ATA for the stream NFT owned by the recipient.
  #[account(
-     init,
+     init_if_needed,
      payer = creator,
      associated_token::mint = stream_nft_mint,
      associated_token::authority = recipient,
      associated_token::token_program = nft_token_program,
  )]
  pub recipient_stream_nft_ata: Box<InterfaceAccount<'info, TokenAccount>>,
```

This change allows the transaction to proceed even if the account was pre-created, while maintaining the same functionality. The approach is beneficial for associated token accounts where pre-creation by attackers would not compromise the account's security or functionality, only its creation timing.

Sablier: Acknowledged. While pre-creating the `recipient_stream_nft_ata` and `campaign_ata` is, theoretically, possible, considering that:

- Doing so would require a great deal of insight into the planned stream/campaign creation (i.e. collecting all of the different seeds that will be used for the creation of `stream_nft_mint` and `campaign`, respectively).
- Even if this does happen, the resulting situation can be mitigated as simply as re-submitting the failed transactions, tweaking the input data (e.g. the `salt` of the stream - or the `campaign_start_time`) a little (as the probability of an attacker obtaining the tweaked input data, as well, is very low).
- Accepting the fact that the aforementioned ATAs might've already been created would go against the logical/natural flow of our programs (e.g. the ATA holding the balance of the stream NFT isn't supposed to exist before the stream is created) - and would require additional documentation to explain the motivation behind the "unnatural" `init_if_needed`.

We will stick to the current way of creating the `recipient_stream_nft_ata` and `campaign_ata` accounts.

Cantina Managed: Acknowledged.

3.4.5 Use of deprecated Metaplex token metadata functions

Severity: Informational

Context: nft.rs#L54-L81

Description: The Sablier lockup contract uses deprecated Metaplex token metadata functions that have been replaced with newer alternatives. Specifically, the code uses `create_metadata_accounts_v3` and `CollectionDetails::V1` which have been deprecated in favor of the `create_v1` instruction.

The deprecated functions are found in:

1. `create_metadata_accounts_v3` at line 55 in `nft.rs`.

2. `set_and_verify_sized_collection_item` at line 104 in `nft.rs`.
3. `CollectionDetails::V1` at line 184 in `nft.rs`.

According to Metaplex documentation, `CreateMetadataAccountV3` has been replaced with `CreateV1`, which provides a more streamlined approach by directly creating the mint and master edition in a single instruction rather than requiring separate calls.

The `set_and_verify_sized_collection_item` function is also deprecated, as it both verifies the collection item and increments the collection size. The newer approach uses the `verify()` instruction, which calls `verify_collection_v1()` internally, providing a cleaner separation of concerns.

Additionally, the entire collection size tracking mechanism appears to be deprecated in the newer Metaplex implementations. The `CollectionDetails::V1` approach for tracking collection size has been superseded, though the current implementation correctly uses an independent `total_supply` tracking mechanism in the `nft_collection_data` account.

While the deprecated functions continue to work and currently don't introduce security vulnerabilities, using outdated APIs may lead to future compatibility issues and indicates the codebase is not following current best practices.

Recommendation: Consider upgrading to the newer Metaplex token metadata API when feasible:

1. Replace `create_metadata_accounts_v3` with the `create_v1` instruction.
2. Update the collection creation logic to use the current recommended patterns.
3. Ensure thorough testing of the new implementation across both backend and frontend integrations.

This upgrade would improve long-term maintainability and ensure compatibility with future Metaplex updates. Given the complexity of the changes required, this could be considered for a future release.

Sablier: Acknowledged. Because of the size of the required changes and the fact that we now have multiple questions/uncertainties regarding the Metaplex framework (we hope to clarify them with the Metaplex team directly), we have decided to leave the upgrade to `CreateV1` to our next release on Solana.

Cantina Managed: Acknowledged. We agree it's fine for now to use the deprecated code and changing it isn't worth the complexity risk at this point.

3.4.6 NFT names can be mined to match existing streams for potential phishing

Severity: Informational

Context: nft.rs#L35

Description: The NFT naming scheme in the Sablier lockup contract creates stream NFT names using a truncated format that displays only the first and last 5 characters of the mint key:

```
let nft_name = format!("{NFT_NAME_PREFIX}{}...{}", &mint_key[..5], &mint_key[mint_key.len() - 5..]);
```

This format results in names like ”Sablier LL Stream #2qidf...dm8jF” where the middle portion of the mint key is hidden. An attacker could potentially mine mint keys (by trying different salt values) to create streams with NFT names that visually match existing legitimate streams.

Since the salt parameter is user-controlled in the stream creation process, an attacker could iterate through different salt values until they generate a mint key that produces the same truncated display format as a target stream. This could enable phishing attacks where malicious streams appear to have the same name as legitimate ones in wallet interfaces or NFT marketplaces.

While this requires significant computational effort and the full mint keys would still be different, users who only look at the displayed NFT name might be deceived into interacting with the wrong stream.

Recommendation: Consider using a more collision-resistant naming scheme, such as:

1. Use the full mint key: Display the complete mint address to eliminate any possibility of visual collision.
2. Use a longer truncation: Increase the number of characters displayed from each end of the mint key to make collision mining computationally infeasible.

For example:

```
// Option 1: Full mint key
let nft_name = format!("{NFT_NAME_PREFIX}{}", mint_key);

// Option 2: Longer truncation (first 8, last 8)
let nft_name = format!("{NFT_NAME_PREFIX}{}...{}", &mint_key[..8], &mint_key[mint_key.len() - 8..]);
```

Sablier: Acknowledged. Unfortunately, we can't follow any of the above recommendations without seriously trimming the `NFT_NAME_PREFIX` (which would worsen the UX of anyone looking at the stream name, as well). The reason for this is that Metaplex metadata has a maximum name length of 32 bytes.

See solsab issue 199 for more context.

Cantina Managed: Acknowledged. Then perhaps the best is to thoroughly document and to raise awareness.

Sablier: While we agree that not using the full address of the `stream_nft_mint` in the NFT name of a stream makes it possible (albeit not trivial) for someone to create another stream with the same NFT name (i.e. with the same first and last 5 characters), we consider that the NFT name must, also, be human-readable. At the same time, there's, also, the 32-byte maximum length for the NFT name imposed by Metaplex.

So, to maintain a balance between the uniqueness and readability of the stream NFT name, we're going to keep the NFT name as `Sablier LL Stream #[first 5 chars of mint key]...[last 5 chars of mint key]`.

3.4.7 IPFS CID omitted from campaign account seeds

Severity: Informational

Context: create_campaign.rs#L37-L53

Description: The campaign account creation in the merkle instant program excludes the `ipfs_cid` parameter from the deterministic address generation seeds, while including other campaign parameters like `merkle_root`, `campaign_start_time`, `expiration_time`, and `name`. The current seed structure at lines 42-50:

```
seeds = [
    CAMPAIGN,
    creator.key().as_ref(),
    merkle_root.as_ref(),
    campaign_start_time.to_le_bytes().as_ref(),
    expiration_time.to_le_bytes().as_ref(),
    name.as_ref(),
    airdrop_token_mint.key().as_ref(),
],
```

The `ipfs_cid` is passed as a parameter and stored in the campaign account but not used for address derivation. This omission was made because IPFS CIDs can exceed the 32-byte limit imposed by Anchor for seed parameters.

However, this creates a scenario where two campaigns with identical parameters (same creator, merkle root, timing, name, and token) but different IPFS CIDs would attempt to use the same account address, potentially causing conflicts during campaign creation. While this is a very unlikely scenario in practice, as it would require a creator to launch multiple campaigns with identical parameters except for the IPFS CID, it represents a potential edge case.

Recommendation: Consider including the IPFS CID in the seed derivation by hashing it to fit within the 32-byte constraint:

```
seeds = [
    CAMPAIGN,
    creator.key().as_ref(),
    merkle_root.as_ref(),
    campaign_start_time.to_le_bytes().as_ref(),
    expiration_time.to_le_bytes().as_ref(),
    name.as_ref(),
    airdrop_token_mint.key().as_ref(),
    hash(&ipfs_cid).as_ref(),  // Add hashed CID to ensure uniqueness
],
```

Using a secure hash function like SHA256 or Keccak256 on the IPFS CID would provide 256-bit security, which should be sufficient for this use case, while fitting within the seed size constraints. While this doesn't provide identical security to the original CID, the 256-bit collision resistance is cryptographically secure for all practical purposes. This approach ensures that campaigns with different IPFS CIDs generate different account addresses, preventing potential conflicts while maintaining strong cryptographic security.

Sablier: Acknowledged. We tried to implement the version where we call the hash function inside the Instruction Context macro directly, but, unfortunately, this is not allowed by the macro.

The alternative solution would involve passing both the `ipfs_cid` and the `ipfs_cid_hash` — and storing them in the program state. However, this would worsen the DX for integrators - and we consider that this downside is more significant than the potential upside of using the `ipfs_cid` as one of the seeds when creating campaigns.

Cantina Managed: Acknowledged.

3.4.8 Code quality issues

Severity: Informational

Context: lib.rs#L133-L144

Description and Recommendations:

1. Incorrect comment in clawback function. The comment at line 81 in `clawback.rs` incorrectly states the transfer destination:

```
- // Interaction: transfer tokens from the Campaign's ATA to the campaign creator's ATA.
+ // Interaction: transfer tokens from the Campaign's ATA to the clawback recipient's ATA.
```

2. Inconsistent function argument order. The `create_campaign` function in `lib.rs` and its handler have different parameter orders. Consider updating the handler function signature to match the order in the main function for consistency:

```
// In lib.rs
pub fn create_campaign(
    ctx: Context<CreateCampaign>,
    merkle_root: [u8; 32],
    campaign_start_time: i64,
    expiration_time: i64,
    name: String,
    ipfs_cid: String,
    aggregate_amount: u64,
    recipient_count: u32,
) -> Result<()>

// Handler should match this order
```

3. Inconsistent event field order. The `FeesCollected` event emission at line 47 in `collect_fees.rs` has fields in the wrong order. Update to match the event definition:

```
  emit!(events::FeesCollected {
+     fee_amount: collectible_amount,
      fee_collector: ctx.accounts.fee_collector.key(),
      fee_recipient: ctx.accounts.fee_recipient.key(),
-     fee_amount: collectible_amount
  });
```

Sablier: Fixed in PR 240.

Cantina Managed: Fix verified.