Security Audit Report

Token Wrap Extension Solana

Delivered: October 30, 2025

runtime verification

Table of Contents

Disclaimer

Executive Summary

Anza engaged Runtime Verification Inc. to perform a security audit of its smart contract. The audit was conducted between September 25 and October 6, 2025. The objective was to assess the implementation's security and correctness, identify exploitable vulnerabilities, and provide recommendations to enhance the client system's reliability.

The Token Wrap program, previously audited by Runtime Verification (report reference), was developed to address the interoperability issues between different token types in Solana. It achieves this by enabling users to wrap tokens into either SPL Token or SPL Token 2022-adherent tokens. Although the Token Wrap program has already been audited previously, the Anza team has implemented new features, which are the target of this audit engagement. These features are:

- The use of confidential transfers by default in wrapped token mints following the Token 2022 standard.
- The use of token metadata, which is now implemented for tokens following any of the available token standards.
- The synchronization of metadata between unwrapped and wrapped tokens.
- A standard interface for customization of token mints, created to facilitate third-party modification of mints handled by the Token Wrap program. This includes standard examples of customized mints created by the client.

The audit process involved a comprehensive review of the smart contract codebase, focusing on manual inspection and formal analysis techniques. Runtime Verification utilized several techniques related to formal specification generation, invariant analysis, and testing to rigorously analyze the system's behavior under various conditions. This included developing and analyzing key invariants to ensure the system's integrity across all state transitions.

The audit led to the identification of issues of potential severity for the protocol’s health, summarized as follows:

All findings have been acknowledged by the client, along with suggested mitigations to improve the security and maintainability of the Token Wrap system. If not fully addressed, the pending topics related to that finding have been considered not to be threatening to the health of the project or Solana's ecosystem. Any responses or fixes submitted by the client have been reviewed and noted as part of the report's finalization.

Goal

The goal of the audit is threefold:

- Review the high-level business logic (protocol design) of the Token Wrap program based on the provided documentation and code;
- Review the low-level implementation of the system for the Solana smart contract;
- Analyze the integration between abstractions of the modules that interact with the contract within the scope of the engagement, and reason about possible exploitative corner cases.

The audit focuses on identifying issues in the system's logic and implementation that could render the system vulnerable to attacks or cause it to malfunction. Furthermore, the audit highlights informative findings that could be used to improve the safety and efficiency of the implementation.

Scope

The scope of this audit is limited to the code contained in a single public GitHub repository provided by the Anza team. The Token Wrap contract was identified within this repository as the primary artifact under review for this engagement.

- Token Wrap Repository (public)
  - https://github.com/solana-program/token-wrap
  - Commit: `228dc976d454b766e649ea7759304e1fb457c76d`
  - `program/src`: Core files specifying the business logic behind the minting, wrapping, and unwrapping within the Token Wrap contracts.

Although the entirety of the `program/src` files is in scope for this audit, the focus of this engagement was on the implementation of confidential transfers (when wrapping to a Token 2022), metadata usage and management on wrapped tokens, customization of wrapped mints, and the sample customized mints developed by Anza.

The codebase under review consists of approximately 1,200 lines of Rust code written for the Solana ecosystem. While preparing for the audit, Runtime Verification reviewed code comments, publicly available documentation, and supplemental materials shared by the Anza team.

The audit is strictly limited to the artifacts listed above. Off-chain components, frontend logic, deployment infrastructure, and third-party integrations are outside the scope of this engagement.

Commits addressing any findings presented in this report have also been reviewed to verify that identified issues were appropriately addressed prior to report finalization.

Methodology and Engagement Plan

The audit lasted three calendar weeks, with each phase was designed to identify and validate both high-level and low-level security concerns. We followed a structured and thorough approach to maximize the effectiveness of this audit engagement within the agreed-upon timeframe.

To facilitate our understanding of the platform's behavior, higher-level representations of the Rust codebase were created, including:

- Modeled sequences of logical operations, considering the limitations enforced by the identified invariants and checking whether all desired properties hold for any possible input value;
- Manually built high-level function call maps, aiding the comprehension of the code and organization of the protocol's verification process;
- Created abstractions of the elements outside the scope of this audit to build a complete picture of the protocol's business logic in action;
- Explored static analyzers and testing tools in a time-boxed effort to identify potential issues detectable through different knowledge bases.

This approach enabled us to systematically check consistency between the intended design and the provided Solana Rust implementation of the system.

On the tooling side, we used `cargo audit`, `cargo outdated`, and `Almanax` to look for potential issues. `cargo audit` and `cargo outdated` reported vulnerabilities related to libraries used by the Solana infrastructure needed for the Token Wrap program. Almanax reported a series of issues, all of which were either accepted by design, identified as false positives, or related to out-of-scope content (i.e., client-side code).

Finally, we conducted rounds of internal discussions with security experts over the code and platform design to verify possible exploitation vectors and identify improvements for the analyzed contracts.

Additionally, given the nascent Solana development and auditing community, we reviewed this list of known Ethereum security vulnerabilities and attack vectors and checked whether they apply to the smart contracts and scripts; if they apply, we checked whether the code is vulnerable to them.

Audit Plan

Week 1

- Design review and business logic comprehension of the newly implemented features;
- Investigation of static analysis tools;
- High-level code review;
- Initiate the investigation of the usage of the Confidential Transfers extension in wrapped tokens following the Token 2022 standard.

Week 2

- Continue investigating the implementation of the Confidential Transfers extension in wrapped tokens following the Token 2022 standard.
- Creation of the formal specification of the newly implemented features;
- Analyze the Mint Customizer implementation and the sample implementation of customized tokens;
- Evaluate the consequences of enabling different extensions in wrapped tokens.

Week 3

- Evaluate the endpoints for synchronization of metadata between unwrapped and wrapped tokens;
- Finalization of pending analysis;
- Curation of the report.

It was possible to follow the audit plan, with minor deviations, to analyze potentially dangerous topics pertinent to the Token Wrap’s usage and the management of wrapped and unwrapped tokens.

Platform Features and Logic Description

The Token Wrap program is a Solana smart contract designed to enhance interoperability between the original SPL Token standard and the feature-rich SPL Token 2022 standard. The original standard established a robust foundation, but subsequent evolution of the blockchain space necessitated more sophisticated capabilities. Token 2022 addressed this need by introducing powerful extensions such as transfer fees, confidential transfers, and interest-bearing tokens. This innovation created a divergence in the ecosystem, requiring a strategic way to bridge the two token systems without sacrificing new functionality or breaking older applications.

Token wrapping provides the crucial solution by allowing tokens to conform to one standard while being represented as the other, maintaining a one-to-one backing. This mechanism ensures that assets with enhanced features can operate seamlessly within legacy applications, while older tokens gain necessary access to newer applications designed for Token 2022 capabilities. Without this essential process, the Solana ecosystem would face significant fragmentation, hindering liquidity and limiting digital assets' overall utility and composability.

The previous audit report documents the full description of wrapping, unwrapping, and mint creation, as well as the general logic of the original features of the Token Wrap contract.

In this updated version, the Anza team aimed to make the wrapped tokens more flexible and expand the number of scenarios in which they can be used. Introducing customizable mints also provides a seamless way for developers to modify wrapped token mints, creating wrapped tokens suitable for their needs or the needs of their protocols.

Updated Token Wrap Interface

Among the previously discussed instructions (`createMint` , `wrap` , `unwrap` , and `CloseStuckEscrow`), the Token Wrap program itself supports two new distinct instructions:

- `SyncMetadataToToken2022` : copies metadata from an unwrapped mint to its corresponding wrapped Token 2022 mint's Token Metadata extension. It supports synchronization from both SPL Token (via Metaplex metadata) and Token 2022 sources to Token 2022 wrapped mints.
- `SyncMetadataToSplToken` : copies metadata from an unwrapped mint to its corresponding wrapped SPL Token mint's Metaplex metadata account. It supports synchronization from both Token 2022 and SPL Token sources to SPL Token wrapped mints.

In Figure 1, we display the instruction interface of the Token Wrap program, with a highlight on the functions listed above, together with the forwarded instruction information (accounts and parameters) and account state changes consequent to the successful execution of each instruction.

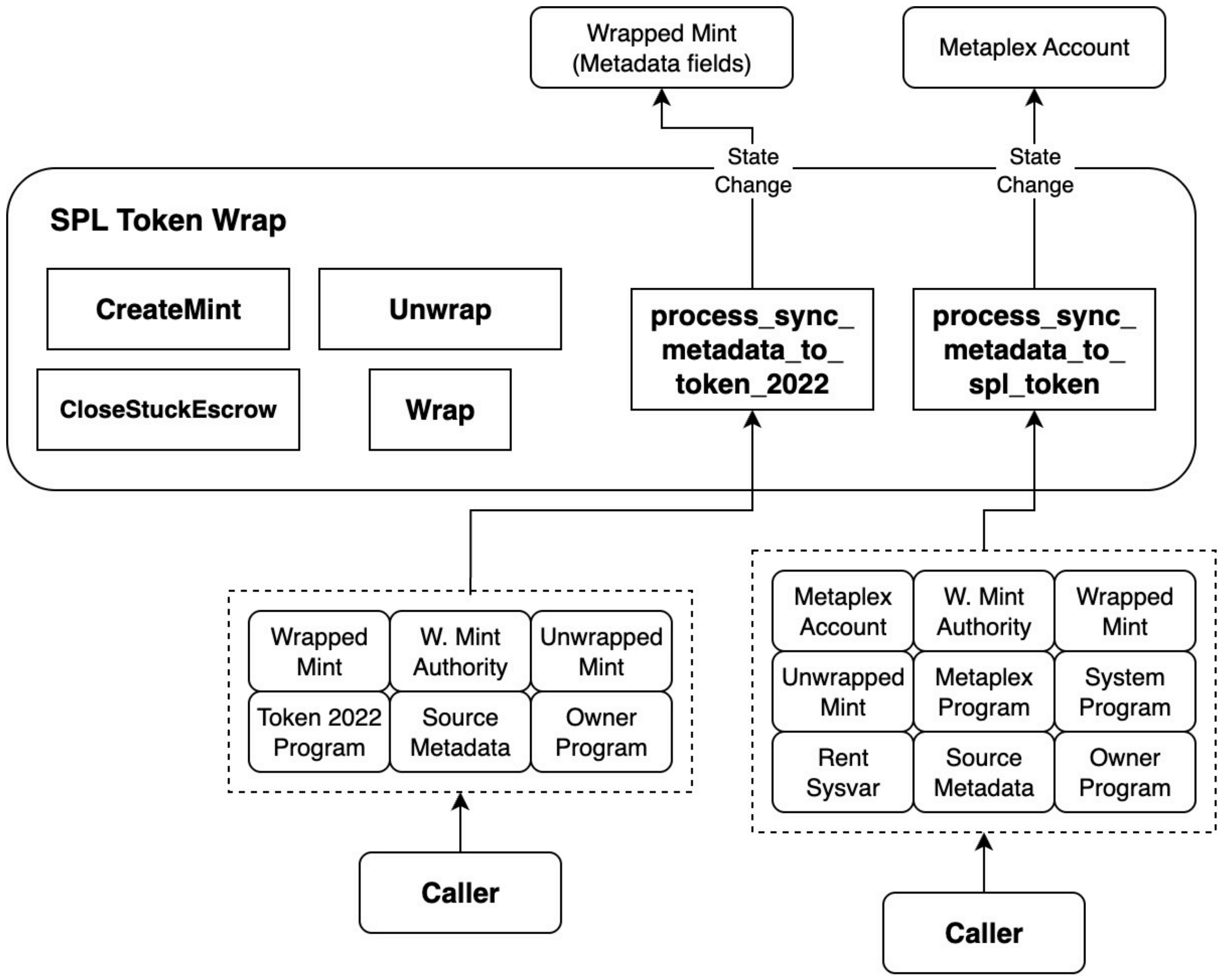

Figure 1: Updated Token Wrap program interface.

In the `SyncMetadataToToken2022` instruction, the Token Wrap program first validates the accounts and extracts metadata from the source, then either initializes the `TokenMetadata` extension if it doesn't exist or updates existing fields that have changed, including base fields (name, symbol, URI) and additional metadata key-value pairs. The `SyncMetadataToSplToken` instruction works similarly but targets Metaplex metadata accounts for SPL Token wrapped mints. It can create new Metaplex metadata accounts or update existing ones, with the wrapped mint authority PDA acting as the payer for the Metaplex program CPI calls. Both functions enable seamless metadata synchronization across token standards, allowing wrapped tokens to maintain consistent metadata with their unwrapped counterparts while supporting bidirectional conversions between the SPL Token and Token 2022 ecosystems.

To synchronize data available for the wrapped token, we highlight that the Token Wrap is capable of extracting data from:

- Metaplex PDAs in case of an unwrapped token following the SPL Token standard, or optionally, if the token following the Token 2022 uses the MetadataPointer extension for referencing a Metaplex PDA;
- The unwrapped token mint, if it follows the Token 2022 standard and holds metadata in the mint account;
- A third-party program, if the unwrapped token mint follows the Token 2022 standard with the MetadataPointer extension pointing to a program.

If the wrapped token follows the SPL Token standard, the synchronized data will be placed in a Metaplex PDA created for that specific token mint. If the wrapped token follows the Token 2022 standard, the synchronized data will be placed in the wrapped mint itself.

According to the Token Wrap's design, any other possible scenario is prohibited.

Mint Customizer

The MintCustomizer trait allows developers to customize how wrapped Token 2022 mints are created by controlling three key aspects: the account space needed for extensions, which extensions get initialized, and the freeze authority and decimals copied from the unwrapped mint. The client highlights this as a core feature, enabling developers to fork the program and add custom logic to include any Token 2022 extensions or to modify default properties. The trait is used during mint creation when the Token Wrap initializes the wrapped token mint, within the `CreateMint` instruction.

The trait defines three methods that control different stages of mint initialization. First, `get_token_2022_mint_space()` calculates the required account size including extensions. Second, `initialize_extensions()` is called before the base mint is initialized to set up extensions that must be added to an uninitialized account, like `ConfidentialTransferMint`, added for the default wrapped tokens minted by this version of the Token Wrap program. Finally, `get_freeze_auth_and_decimals()` determines these properties for the wrapped mint, typically by copying them from the unwrapped mint. The program currently uses `DefaultToken2022Customizer` as the implementation for its wrapped token mints, which can be swapped out at the processor level.

By implementing this trait, you can create custom mint configurations with different Token 2022 extensions or behaviors. The codebase includes three example implementations: `DefaultToken2022Customizer`, which adds `ConfidentialTransferMint` and `MetadataPointer` extensions, `NoExtensionCustomizer`, which creates mints without any extensions, and `ComplianceMintCustomizer`, which adds compliance-focused extensions, including a permanent delegate, pausable transfers, confidential transfers with a designated auditor, and a freeze authority. To use a custom implementation, you fork the program, create a new struct implementing `MintCustomizer` in `program/src/mint_customizer`, replace it in the processor, update tests, and if using clients, update the CLI's mint customizer type and the client's mint size calculation.

Confidential Transfers By Default

Finally, the Token Wrap program automatically adds the `ConfidentialTransferMint` extension to every wrapped mint created under the Token 2022 standard. This is implemented in the `DefaultToken2022Customizer`, which initializes the extension with specific configuration: no authority (making it immutable), automatic approval for new accounts, and no auditor. The extension is included in the mint space calculation alongside the `MetadataPointer` extension, and is one of the modifications developers should be aware of if attempting to modify the wrapped token mint.

The rationale for this design is to enable privacy-preserving transactions by default without requiring additional setup from users. By setting the authority to `None`, the configuration becomes immutable, ensuring that privacy features cannot be disabled after creation. The automatic approval setting (`auto_approve_new_accounts: true`) allows users to make confidential transactions permissionlessly without needing explicit approval. The absence of an `auditor` means no third party can decrypt transaction amounts, maximizing privacy. This approach provides users with the latest Token 2022 privacy features while maintaining the program's permissionless design philosophy.

Invariants

Alongside the invariants raised in the previous audit of the Token Wrap, we generated an additional set of invariants during the design review of this engagement with the objective of narrowing down the security specification of the newly implemented features. These invariants are:

- When applicable, the sum of all transparent and confidential balances of wrapped token accounts must always equal the total balance of their unwrapped counterpart held in the program’s escrow account.
- The Confidential Transfer mint configuration should not be modified.
- The customization trait logic must be restricted from modifying critical properties that violate the wrapping program's invariants.
- The metadata fields on the wrapped token must be set to perfectly match the corresponding metadata of the unwrapped token.

We note that these invariants were derived under the assumption that Solana’s native features used by the protocol are sound in terms of safety and security (e.g., no PDA derivation collisions).

Furthermore, although these invariants guided our analysis, any behavior diverging from the expected protocol business logic was noted and brought to the client’s attention.

Findings

Findings presented in this section are issues that can cause the system to fail, malfunction, and/or be exploited, and should be properly addressed.

All findings have a severity level and an execution difficulty level, ranging from low to high, as well as categories in which it fits. For more information about the classifications of the findings, refer to our Smart Contract Analysis page (adaptations performed where applicable).

[A1] Confidential Transfer Auditor Bypass Through Token Wrapping

Severity: Low Difficulty: Medium Recommended Action: Fix Design
Not addressed by client

Description

The Token Wrap program's default configuration creates wrapped Token-2022 mints with the `ConfidentialTransferMint` extension enabled but without a confidential transfer auditor. While this design choice is intentional for privacy, it creates a potential bypass mechanism for tokens that originally had auditor oversight.

When users wrap tokens from mints configured with a confidential transfer auditor (for compliance or regulatory purposes), the resulting wrapped tokens lose this auditor capability entirely. This allows users to effectively circumvent auditor oversight by wrapping their tokens and conducting confidential transfers through the wrapped version, where the original auditor authority cannot decrypt transaction amounts. While this behavior aligns with the program's permissionless and privacy-focused design, it may have regulatory implications for tokens where auditor oversight was a deliberate compliance feature. Developers deploying tokens with required auditor oversight should know that the Token Wrap program allows users to bypass such controls through the wrapping mechanism.

Recommendation

To prevent the use of wrapped tokens to bypass security features of an unwrapped token, we recommend enforcing that wrapped tokens use the same auditor as their unwrapped counterpart.

Status

This finding has been acknowledged by the client. Copying over the Confidential Transfers extension settings could potentially cause configuration synchronization issues between the wrapped and unwrapped token programs.

[A2] Freeze Authority Bypass Through Wrapped Token Trading

Severity: Medium · Difficulty: Medium · Recommended Action: Document Prominently · Not addressed by client

Description

The Token Wrap program can be used to circumvent freeze authority controls through wrapped token trading mechanisms, even though the freeze authority is carried over from the unwrapped token program to its wrapped counterpart.

Even when the freeze authority manages the freezing rights for the unwrapped and the wrapped token programs, fundamental timing and awareness problems remain. Consider a single user who holds two distinct accounts, one for the original asset and one for the wrapped asset, both representing the same fungible value. For the token freezing mechanism to effectively prevent users from transacting with their funds, the freeze authority must be prepared to target and lock both these token accounts simultaneously.

The primary vulnerability comes from the freeze authority's awareness of the wrapped token counterpart, burdening this entity with the task of finding a possible second target token account that must be frozen as well.

Following this, another vulnerability lies within the evasion window when freezing occurs sequentially. Imagine the freeze authority successfully submits a transaction to freeze the user's unwrapped token account. At this point, the user's balance in the original asset is locked, preventing its use. However, if the user's wrapped token account has not yet been frozen, the user retains complete control over the wrapped equivalent. During the critical time delay between the first freeze transaction finalizing and the second freeze transaction being executed, the user can quickly transfer or exchange their remaining wrapped tokens for another asset of equal value. This action successfully drains the valuable asset from the ecosystem before the freeze authority can complete its intended action, rendering the feature useless.

Recommendation

To mitigate this risk, it is paramount that the freezing authority is not only aware of the wrapped token counterpart but is also architecturally prepared to freeze both accounts in one coordinated, atomic operation.

Status

This finding has been acknowledged by the client.

[A3] Privacy Loss When Wrapping Tokens With Confidential Balances

Severity: Low Difficulty: Medium Recommended Action: Document Prominently

Not addressed by client

Description

Combining the Token Wrap program with confidential tokens creates a tricky situation for privacy. The core idea behind confidential tokens is that the amounts you hold and transfer are hidden. However, the Token Wrap program is designed to work with public (transparent) balances and does not support direct wrapping of tokens through confidential transfers. This means if you want to wrap your confidential tokens, you can't just do it privately.

You must first move the amount you want to wrap from your confidential balance to your account’s public balance for the unwrapped token. This necessary step temporarily exposes the exact number of tokens involved for a moment. Then, the program wraps this public balance and gives you an equivalent amount of wrapped tokens in a public balance again. You then have to do another step to shift those wrapped tokens back into the wrapped token’s confidential balance.

This sequence of events creates a temporary privacy leak involving information that was originally intended to be confidential. Even though the transfers within the confidential system are hidden, the specific amount of the wrap or unwrap transaction is visible on the public ledger. An observer can easily see the precise value you are moving through the wrapping service, which defeats the purpose of having a confidential balance in the first place for that specific transaction amount.

Recommendation

The initial solution for this issue is to directly wrap tokens using confidential, instead of standard, transfers. Still, we are aware of the technical complexity of introducing such a feature. Having the wrapping program handle the zero-knowledge proofs required for confidential transfers directly could present a challenge and a significant overhead for using the Token Wrap program.

Still, we recommend raising awareness of this potential loss of privacy for users aiming to use confidential transfers with wrapped tokens.

Status

The client has acknowledged this finding.

[A4] Partial Synchronization of Token Metadata

Severity: Low | Difficulty: Low | Recommended Action: Fix Code | Not addressed by client

Description

There will be a loss of Metadata when wrapping to SPL tokens, regardless of the standard followed by the unwrapped token.

The core of the problem lies in the structural difference between the complete Metaplex PDA data and the `DataV2` object used in update operations. The Metaplex PDA account holds the entire metadata state, which includes essential information like the `name`, `symbol`, and `URI`, as well as other system-relevant fields such as `updateAuthority`, `primarySaleHappened`, `isMutable`, and crucially, `collection` or `uses`. When a synchronization instruction is executed, it relies on constructing a new `DataV2` object, which is essentially a subset of the full PDA data, containing only the publicly visible fields like name, symbol, and off-chain URI.

Because the `DataV2` object is incomplete and doesn't account for all fields in the full PDA structure, the update instruction only overwrites the fields defined within `DataV2`. Any fields in the PDA that are not included in `DataV2` are either left untouched, must be managed by separate instructions, or, in the case of the remaining fields that can be modified in the `UpdateMetadataAccountV2` instruction (namely `primary_sale_happened`, `new_update_authority`, and `is_mutable`), are assigned the `None` value. This creates a synchronization challenge for wrapped tokens: if the unwrapped token's PDA has certain flags or auxiliary data sets outside the `DataV2` structure, simply constructing and using it to update the wrapped token’s Metaplex PDA will result in a discrepancy. The wrapped token will look correct superficially but will fail to replicate the full, complex state of the original metadata, leading to potential issues with marketplace listings, collection verification, or future operations that rely on these overlooked fields.

Furthermore, any additional fields of an unwrapped token following the Token 2022 standard will be lost in the synchronization process, resulting in an incomplete mirroring of metadata between the wrapped and unwrapped counterparts.

Recommendation

Synchronize all available fields of the Metaplex PDA when possible.

Given the limitations to perform the on-chain synchronization of unwrapped to wrapped token metadata, bring awareness to users and developers of potential challenges and issues associated with using wrapped tokens with metadata.

Status

The client has acknowledged this finding. This decision simplifies handling wrapped tokens with metadata concerning the trust in the metadata update authority and potentially in external metadata sources (when applicable).

Informative Findings

The findings presented in this section do not necessarily represent any flaw in the code itself. However, they indicate areas where the code may need external support or deviate from best practices. We have also included information on potential code size reductions and remarks on the operational perspective of the contract.

[B1] The Usage of the ConfidentialMintBurn Extension Breaks Fundamental Protocol Invariance

Severity: Informative Recommended Action: Document Prominently
Not addressed by client

Description

Consider the first Invariants: "When applicable, the sum of all transparent and confidential balances of wrapped token accounts must always equal the total balances of its unwrapped counterpart held in the program's unwrapped escrow account."

The core issue related to the above arises specifically with the potential usa of the `ConfidentialMintBurn` extension by customized mints. This extension is designed to give a designated authority the power to adjust the token supply arbitrarily, without needing a standard deposit or burn operation. The extension's purpose is typically to facilitate advanced privacy mechanisms, allowing supply changes to occur confidentially. However, if a developer mistakenly sets this authority to an address outside of the main wrapping program's control when creating the customized wrapped mint, or if they purposefully do so with malicious intent, that external party gains the ability to mint or burn an arbitrary number of wrapped tokens at any time.

By acting maliciously and minting new wrapped tokens, for instance, they could completely bypass the requirement for a corresponding unwrapped token deposit. The supply of the wrapped token would immediately become greater than the total collateral held in the wrapping program's escrow account. This event is devastating because it instantly breaks the one-to-one financial invariant, leading to an undercollateralized token and a complete loss of confidence in the wrapper.

Recommendation

Projects considering the usage of `ConfidentialMintBurn` extension on their mints should carefully evaluate these risks and implement additional safeguards, while users should verify they are interacting with trusted, audited implementations of the Token Wrap program. Users should only decide to use such tokens once trust in the Token Wrap authority is established.

Status

This finding has been acknowledged by the client. The client has no control over malicious program forks.

[B2] Permanent Delegate Exploitation Risk in Forked Programs

Severity: Informative | Recommended Action: Document Prominently | Not addressed by client

Description

The implementation of the `compliance.rs` example introduces a significant attack vector if malicious actors fork the Token Wrap program.

By modifying the `PERMANENT_DELEGATE` constant to their own address, an attacker can deploy a malicious version of the program that appears functionally identical to the legitimate one. The Permanent Delegate extension grants unrestricted transfer and burn privileges over all token accounts for that mint, enabling the attacker to drain all user funds and break the 1:1 backing invariant.

Additionally, users may be convinced to wrap their legitimate tokens (e.g., USDC) through the malicious program, which claims to use a version of the Token Wrap developed by the Anza team that grants the attacker complete control over their assets. This attack is particularly dangerous because:

1. The malicious program may appear legitimate and function normally initially,
2. Users may not understand the implications of Permanent Delegate permissions, and
3. The permissionless nature of token wrapping means anyone can create wrapped versions of existing tokens.

Recommendation

Projects considering Permanent Delegate functionality should carefully evaluate these risks and implement additional safeguards, while users should verify that they are interacting with trusted, audited implementations of the Token Wrap program. Users should only decide to use such tokens once trust in the Token Wrap authority is established.

Status

This finding has been acknowledged by the client. The client has no control over malicious program forks.

Final Considerations

It has been a privilege to conduct the code security review for Anza on the Token Wrap project. Our engagement involved a comprehensive analysis of the project's codebase with a focus on the core smart contract, and we are confident that this review has significantly strengthened the project's foundation. A critical component of our work included rigorously verifying the Token Wrap program's security against a defined set of invariants, with the support of the developed formal and semi-formal specifications of the actors and programs involved, and we are pleased to confirm that a minimal number of issues have been identified with the protocol's logic and design.

We highlight that the majority of the identified issues are related to intrinsic attributes of the Solana blockchain, as well as the token standards and their extensions implemented so far, combined with the customization capabilities of the Token Wrap program. While great benefit will come from being capable of modifying the wrapped mints to better suit the developers' needs, this capability might be used as an exploitation vector by malicious users.

We once again would like to bring attention to users' awareness regarding their trust in the authorities of deployed Token Wrap programs with customized mints. While we consider the Token Wrap program and its wrapped tokens safe to use, malicious individuals can customize it for malicious purposes.