certora

Security Assessment
Final Report

Light Protocol
December 2025

Prepared for Light Protocol

Table of content

Project Summary

Project Scope

Project Name	Repository (link)	Diff Hashes	Fix Commit Hash	Platform
Light Protocol	https://github.com/Lightprotocol/light-protocol	93c5828..e3eef25	13c64a1	Solana

Project Overview

This document describes the specification and verification of **Light Protocol** using manual code review findings. The work was undertaken from **October 20** to **December 5, 2025**, with fixes reviewed from **December 8, 2025** to **December 12, 2025**.

The following contract list is included in our scope:

```
{
      programs/compressed-token/program/*
              (excluding src/mint_action/actions/create_spl_mint)
      programs/registry/*
      program-libs/account-checks/*
      program-libs/array-map/*
      program-libs/compressible/*
      program-libs/ctoken-types/*
      program-libs/hasher/*
      program-libs/zero-copy/*
}
```

The team performed a manual audit of all the **Rust** programs. During the verification process and the manual audit, the Certora team discovered bugs in the Rust program code, as listed on the following page.

Protocol Overview

The compressed token program is a program that provides similar functionality to the SPL token program, while utilizing the underlying Light System Program in order to save on rent for token accounts.
The code being audited contains an upgrade to the existing compressed token program.
The upgrade includes:

- `Transfer2` ix - a more efficient way to transfer funds, allowing multiple transfer from multiple mints in one ix
- `cToken` - a token that's held in a Solana account - the rent-exempt for the account is being paid by the rent sponsor, in exchange for a rent that's being paid to the rent sponsor.
- Associated token account for the `cToken` accounts
- Ability to create a compressed mint (an SPL equivalent mint that's stored in a compressed account) and execute mint actions with the compressed mint

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	**Discovered**	**Confirmed**	**Fixed**
Critical	2	2	2
High	4	4	4
Medium	7	7	7
Low	4	4	2
Informational	5		
Total	22		

Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
			Likelihood	

Detailed Findings

ID	Title	Severity	Status
C-01	Attacker can create a duplicate mint of an existing mint, allowing them to freely mint an existing token	Critical	Fixed
C-02	Amount isn't checked when owner compresses and closes, allowing an attacker to mint free tokens	Critical	Fixed
H-01	Rent Authority Can Steal Funds during Compress And Close	High	Fixed
H-02	Out of Bounds write in derive_address() leads to undefined behavior	High	Fixed
H-03	Mint isn't checked during compress and close, allowing compression authority to steal funds	High	Fixed
H-04	ATA creation can get DoS-ed by donating a few lamports to the account	Critical	Fixed
M-01	transfer2 allows executing transfers where the user loses funds	Medium	Fixed
M-02	mint_signer isn't checked to actually be a signer, allowing anybody to create their mint	Medium	Fixed
M-03	eq() function fails to check some fields	Medium	Fixed
M-04	Top-up lamports aren't enforced/transferred during minting	Medium	Fixed
M-05	Rent deficit would be charged for each account mention, overcharging the caller	Medium	Fixed

M-06	Top up lamports calculation may wrongly exempt the user from a top up	Medium	Fixed
M-07	write_top_up can be set to an unreasonably high value, to DoS the account or steal funds	Critical	Fixed
L-01	Multiple ATA accounts can be created per user, using different bumps	Low	Acknowledg…
L-02	It's possible to create multiple mints per mint signer using different bumps	Low	Fixed
L-03	Supply reduction due to burning SPL token wouldn't register in the compressed mint	Low	Fixed
L-04	in CToken equality check, we return false on unknown extensions instead of throwing an error	Low	Fixed

Critical Severity Issues

C-01 Attacker can create a duplicate mint of an existing mint, freely minting an existing token		
Severity: **Critical**	Impact: **High**	Likelihood: **High**
Files: programs/compressed-token/program/src/mint_action/actions/create_mint.rs	Status: Fixed	

Description: The Compressed-token program allows anybody to create a new mint, as long as the mint signer signs the tx.
We rely on inserting the address to the right address merkle tree to prevent creating duplicate mints (with the same mint pubkey).

However, when writing to CPI context the caller controls to which merkle address tree we write the address.
This allows them to write to another address tree, which would effectively bypass the check (since addresses are unique only inside the tree and not across trees).

By creating a duplicate mint, the attacker gets access to the existing token and can freely mint and freeze.

As said, the mint signer has to sign the tx, but the mint signer has no authority after the mint creation and shouldn't have this power (also note that combining with issue M-02, mints can be created without mint signer's signature).

Exploit Scenario:

- USDX stablecion is created by Bob and 1M USDX is minted
- Eve gets access to the mint signer and deploys a duplicate mint, setting herself as the authority

- Eve now mints additional 1M USDX and trades it for USDC. Leaving 2M USDX which are backed only by 1M USD

Recommendations: Ensure the address is inserted to the right address tree.

Customer's response: Fixed in b27463

Fix Review: Fix confirmed

C-02 Amount not checked when owner compress-and-closes, allowing an attacker to mint free tokens

Severity: **Critical**	Impact: **High**	Likelihood: **High**
Files: programs/compressed-token/program/src/transfer2/compression/ctoken/compress_and_close.rs	Status: Fixed	

Description: When processing a compress-and-close compression we check that the specified amount is equal to the amount in the token account inside `validate_compressed_token_account()`.
However, this function is called only if `compression_authority_is_signer` is true, meaning not if the owner is the authority for this compression.
This would allow an attacker to specify an amount higher than the actual amount in the account, allowing them to mint free tokens (since the sum of the out tokens would be higher than the actual sum of the in tokens).

```rust
    if compression_authority_is_signer {
        // Compress the complete balance to this compressed token account.
        validate_compressed_token_account(
            packed_accounts,
            amount,
            close_inputs.compressed_token_account,
            ctoken,
            compress_to_pubkey,
            token_account_info.key(),
        )?;
    }
```

Exploit Scenario:

- Eve creates a USDC token account
- Eve calls transfer2 with:
    - A compress and close compression with that account, specifying 1M in the amount
    - A decompression of 1M USDC
- The sum check passes, and Eve gets 1M USDC out of nowhere

Recommendations: Check that the amount in the compression matches the amount in the account before zeroing it out.

Customer's response: Fixed in [21ddf7]

Fix Review: Fix confirmed

High Severity Issues

H-01 Rent Authority Can Steal Funds during Compress And Close		
Severity: **High**	Impact: **High**	Likelihood: **Medium**
Files: programs/compressed-token/program/src/transfer2/compression/ctoken/compress_and_close.rs	Status: Fixed	

Description: If a user fails to pay rent for cToken then the rent authority has the right to compress and close the account.
This would only move the tokens from a cToken account to a compressed account.

In order to ensure that the tokens indeed moved to a compressed account the caller provides an index of the compressed account, and we check that the values indeed match (owner, mint, amount etc.).
However, if the owner has 2 accounts with the same values that have rent deficit the caller can close both of them while specifying only one compressed account. Effectively stealing half of the funds from the owner.

Exploit Scenario:

- Bob has 10K USDC on one account, and 7K USDC on another account
- Bob fails to pay rent and they both default
- Eve is the rent authority of the compressed token, she drops 3K USDC into the second account and then compresses both of them into a single compressed account with 10K USDC
- Eve has earned 7K USDC at the expense of Bob

Recommendations: Check that each cToken account has a separate compressed account and there are no duplicate references.

Customer's response: Fixed in e1fa0b

Fix Review: Fix confirmed

H-02 Out of Bounds write in derive_address() leads to undefined behavior

Severity: **High**	Impact: **High**	Likelihood: **Medium**
Files: program-libs/ctoken-types/src/instructions/extensions/compressible.rs	Status: Fixed	

Description: In `derive_address()` we check that the seeds length isn't greater than `MAX_SEEDS`, but we allow it when they're equal.
The `data` array is of size of `MAX_SEEDS+2` .
This would cause an overflow when seeds length is `MAX_SEEDS`, because on top of the seeds we have 3 elements - bump, program_id and `PDA_MARKER` constant.

Out of bounds writing leads to unexpected behavior in Rust, so theoretically this can cause arbitrary write to any part of the memory.

```rust
const PDA_MARKER: &[u8; 21] = b"ProgramDerivedAddress";
    if seeds.len() > MAX_SEEDS {
        return Err(CTokenError::TooManySeeds(MAX_SEEDS));
    }
    const UNINIT: MaybeUninit<&[u8]> = MaybeUninit::<&[u8]>::uninit();
    let mut data = [UNINIT; MAX_SEEDS + 2];
    let mut i = 0;

    while i < seeds.len() {
        // SAFETY: `data` is guaranteed to have enough space for `N` seeds,
        // so `i` will always be within bounds.
        unsafe {
            data.get_unchecked_mut(i).write(seeds.get_unchecked(i));
        }
        i += 1;
    }

    // TODO: replace this with `as_slice` when the MSRV is upgraded
    // to `1.84.0+`.
    let bump_seed = [bump];

    // SAFETY: `data` is guaranteed to have enough space for `MAX_SEEDS + 2`
    // elements, and `MAX_SEEDS` is as large as `N`.
    unsafe {
        data.get_unchecked_mut(i).write(&bump_seed);
        i += 1;

        data.get_unchecked_mut(i).write(program_id.as_ref());
        data.get_unchecked_mut(i + 1).write(PDA_MARKER.as_ref());
    }
```

Recommendations: Change the inequality check to allow only up to `MAX_SEEDS-1`

Customer's response: Fixed in 917c43

Fix Review: Fix confirmed

H-03 Mint isn't checked during compress and close, allowing compression authority to steal funds

Severity: **High**	Impact: **High**	Likelihood: **Medium**
Files: programs/compressed-token/program/src/transfer2/compression/ctoken/compress_and_close.rs	Status: Fixed	

Description: Compress and close allows the compression authority to move funds into a compressed account when the owner fails to pay rent for the cToken account.
When compression authority calls compress and close, we ensure that the funds are transferred into a compressed account, by checking that there's a compressed account with the same values as the account to be closed.

However, we don't check the mint field. This would allow the compression authority to give the user funds from a cheaper token in exchange.

Exploit Scenario:

- Bob has 10K USDC in a cToken account
- Bob fails to pay the rent
- Eve, the compression authority crafts a `transfer2` ix with the following compressions:
    - Compress and close for Bob's account
    - Compress 10K of a fake token X created by Eve
    - Decompress 10K USDC to Eve's account
    - Output 10K of X to Bob
- The checks would pass, and Eve has effectively stolen 10K USDC from Bob

Recommendations: Check also the mint field

Customer's response: Fixed in 21ddf7

Fix Review: Fix confirmed

H-04 ATA creation can get DoS-ed by donating a few lamports to the account		
Severity: **High**	Impact: **Medium**	Likelihood: **High**
Files: programs/compressed-token/program/src/shared/create_pda_account.rs	Status: Fixed	

Description: ATA accounts are created by invoking the `CreateAccount` instruction. However, this ix requires that the account would have no lamports, and reverts if it has any. This would allow an attacker to DoS the creation of specific ATA accounts by donating as little as 1 lamports to the account.

Recommendations:

- If the account has any lamports - use a combination of assign, allocate and transfer (if needed) instead of create account
- Validate the ATA account address to ensure that it's not the fee payer account (since once we allow creating an account with lamports, this would allow passing the rent sponsor and relying on its signer seeds)

Customer's response: Fixed in fbbd1f

Fix Review: Fix Confirmed

Medium Severity Issues

M-01 `transfer2` allows executing transfers where the user loses funds		
Severity: **Medium**	Impact: **High**	Likelihood: **Low**
Files: programs/compressed-token/program/src/transfer2/processor.rs	Status: Fixed	

Description: `transfer2` allows executing multiple transfers at once, while specifying multiple inputs and outputs (or subtraction and additions) in one tx. As long as the sum of addition and subtractions per mint are zero everything is fine.

However, when executing without the system program we fail to do that check.
We call `sum_compressions()` which implicitly verifies that the sum of additions isn't greater than the subtractions (because the balance variable is an unsigned integer, so it can't be negative), but we don't verify that the subtractions aren't more than the additions.

This can lead to a case where a user mistakenly specifies a tx where they fail to distribute all of the funds, and lose the undistributed amount.

Exploit Scenario:

- Bob intends to send 50 USDC to Alice and 50 USDC to Eve
- Bob creates a `transfer2` ix but fails to mention the transfer to Eve (so it only subtracts 100 from Bob and adds 50 to Alice)
- Transfer executes, and Bob has lost 50 USDC

Recommendations: Ensure the balance is zero after calling `sum_compressions()`

Customer's response: Fixed in 453182

Fix Review: Fix confirmed

M-02 `mint_signer` isn't checked to actually be a signer, allowing anybody to create their mint

Severity: **Medium**	Impact: **Low**	Likelihood: **High**
Files: programs/compressed-token/program/src/mint_action/accounts.rs	Status: Fixed	

Description: As part of the design of the system, mints are created only with a mint signer - an account that has to sign the tx, and which is used as part of the seeds for the mint address. However, in practice we don't check the `mint_signer` has signed the tx, allowing anybody to create a mint using a `mint_signer` that didn't approve the tx.

```rust
        let mint_signer = iter.next_option("mint_signer", config.with_mint_signer)?;
```

Recommendations: Check that the mint signer is indeed a signer.

Customer's response: Fixed in 871215

Fix Review: Fix confirmed

M-03 eq() function fails to check some fields

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: program-libs/ctoken-types/src/state/ctoken/zero_copy.rs	Status: Fixed	

Description: The Ctoken struct has an `eq()` function to check equality against ZCToken. However, this function fails to check the `compress_to_pubkey` and `account_version` fields for the `Compressible` extension.
This means that two `CToken` with different values in those fields might be considered the same.

```JavaScript
impl PartialEq<CToken> for ZCToken<'_> {
    fn eq(&self, other: &CToken) -> bool {
// ....
}
```

Recommendations: Check those fields as well.

Customer's response: Fixed in [90da89]()

Fix Review: Fix confirmed

M-04 Top-up lamports aren’t enforced/transferred during minting		
Severity: **Medium**	Impact: **Low**	Likelihood: **High**
Files: programs/compressed-token/program/src/mint_action/actions/mint_to_ctoken.rs	Status: Fixed	

Description: When minting to cToken, we don’t charge the top-up lamports from the caller. The `compress_or_decompress_ctokens()` function returns the amount to be topped up, but we don’t do anything with the returned value.

```JavaScript
   // For mint_to_ctoken, we don't need to handle lamport transfers
    // as there's no compressible extension on the target account
    compress_or_decompress_ctokens(inputs)?;
    Ok(())
```

Recommendations: Transfer top up lamports during mint as well.

Customer’s response: Fixed in 7ce220 and 82470c

Fix Review: Fix confirmed

M-05 Rent deficit would be charged for each account mention, overcharging the caller		
Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: programs/compressed-token/program/src/transfer2/compression/mod.rs	Status: Fixed	

Description: During transfer we top up the lamports, which means charging the fee payer the rent deficit (if any) for every cToken account, plus a fixed 'lamports per write' amount.

However, under current implementation we charge this amount per every mention of the account in the transfer. Meaning that if there's a rent deficit we're going to charge it multiple times.

Exploit Scenario:

- Alice's account has a rent deficit of 3K lamports
  - Lamports per write is set to 100 lamports
- Bob is transferring to Alice funds, while mentioning her account in 4 different compressions
- Bob (the `fee_payer`) would be charged 12,400 lamports for the top up, instead of 3,400. This might as well be over the max epochs funded ahead.

Recommendations: Charge the rent deficit only once, and don't charge more than the max epochs funded ahead.

Customer's response: Fixed in 82470c

Fix Review: Fix confirmed.
Note that an account would still receive double top up if it’s mentioned twice with different indexes, but this is less likely to happen.

M-06 Top up lamports calculation may wrongly exempt the user from a top up

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: program-libs/compressible/src/compression_info.rs	Status: Fixed	

Description: When calculating the top up lamports, we return 0 (exempt the caller from topping up) if epochs funded ahead have reached the max funded epoch value.
However, the calculation of epochs funded ahead is wrong, since it's based on the available balance, which includes unclaimed rent for previous epochs.
So we might exempt the caller from paying the top up when they actually should pay.

```rust
Rust
    // Calculate epochs funded ahead using available balance
     let available_balance = state.get_available_rent_balance(
         rent_exemption_lamports,
         self.rent_config.compression_cost(),
     );
     let rent_per_epoch = self.rent_config.rent_curve_per_epoch(num_bytes);
     let epochs_funded_ahead = available_balance / rent_per_epoch;
     // Skip top-up if already funded for max_funded_epochs or more
     if epochs_funded_ahead >= self.rent_config.max_funded_epochs as u64 {
         Ok(0)
     } else {
```

Recommendations: Correct the calculations, subtract the unclaimed rent from the available balance.

Customer's response: Fixed in 801b4f

Fix Review: Fix confirmed

M-07 write_top_up can be set to an unreasonably high value, to DoS the account or steal funds

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: program-libs/compressible/src /compression_info.rs	Status: Fixed	

Description: When writing to an account the caller has to pay a top-up amount that would go to the account rent, in order to maintain the account.

The issue is that this amount is set by the creator of the account, and can be set to an unreasonably high amount.

- For ATA accounts - a user might set it to a high value to DoS the interaction with the account
- For regular accounts - the owner can front-run a transfer to/from the account by closing and re-opening the account with a higher `write_top_up` value. Causing the caller to pay a top up amount they didn't intend.
    - The owner can then take the top up lamports by closing the account

Note also that because the top up isn't limited to funding 'max funded epochs', this allows the attacker to extract unlimited amount from the payer.

Recommendations: Cap the `write_top_up` lamports so that the creator can't set it to an unreasonable value

Customer's response: Fixed in 0a08db

Fix Review: Fix confirmed

Low Severity Issues

L-01 Multiple ATA accounts can be created per user, using different bumps		
Severity: **Low**	Impact: **Low**	Likelihood: **Medium**
Files: program-libs/ctoken-types/src/instructions/create_associated_token_account.rs	Status: Acknowledged	

Description: The associated token account (ATA) instruction allows creating for each user a token account associated with that user - where the user would be the owner, and the user is part of the seeds of the account address.
It should be possible to open only one ATA per user, but since we allow the caller to specify the bump then we can create multiple accounts per user by providing different bumps.

Recommendations: Calculate the bump instead of getting it from the user

Customer's response: Acknowledged

L-02 It’s possible to create multiple mints per mint signer using different bumps		
Severity: **Low**	Impact: **Low**	Likelihood: **High**
Files: programs/compressed-token/program/src/mint_action/actions/create_mint.rs	Status: Fixed	

Description: Each mint is associated with a mint signer, the mint signer is used as part of the seeds of the mint pubkey.
Each mint signer should be able to create only one mint associated with them.
However, since we allow the caller to specify the bump this allows creating multiple mints per mint signer, by providing different bumps.

Recommendations: Calculate the bump rather than getting it from the user

Customer’s response: Fixed in f30d22

Fix Review: Fix has removed a critical check of the mint address, this was identified during the review and fixed in commit 9c3b34e

L-03 Supply reduction due to burning SPL token wouldn't register in the compressed mint

Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: -	Status: Acknowledged	

Description: Compressed mint created in the compressed-token program can then be extended to have also an SPL mint (this isn't implemented yet, but it's planned to be implemented), which would allow decompressing the tokens into SPL tokens.
However, the SPL tokens can be burned by the owner without the compressed-token being aware of that. So this supply change wouldn't register in the compressed mint, and the supply there would be overestimated.

Exploit Scenario:

- SmartVaults protocol builds on top of the compressed token, using the compressed token as a token to represent the vault's shares
    - During withdrawals they burn the tokens using the SPL token program (user has to decompress the tokens first)
    - The protocol relies on the supply field in the mint to get the total shares in the vault, allowing to pass either the compressed mint or the SPL mint to read this value
- A vault is opened
- Bob and Alice deposit 50K USDC for 50K shares each
- Bob redeems 50K of their shares for 50K USDC
    - Compressed mint shows a supply of 100K, while the SPL mint shows the correct 50K value
- Eve now deposits 50K USDC providing the compressed-mint to read the total supply
    - This gives her 50K*100K/50K=100K shares, rather than 50K
    - SPL mint now shows 100K, while the compressed mint shows 150K
- Eve withdraws providing the SPL mint this time

    ○ This would give her 100K*100K/100K=100K, effectively stealing 50K from

Recommendations: Document this issue, and ensure protocols building on top of this are aware that the supply can be overestimated.

Customer's response: Acknowledged

L-04 in CToken equality check, we return false on unknown extensions instead of throwing an error

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: program-libs/ctoken-types/src/state/ctoken/zero_copy.rs	Status: Fixed	

Description: While checking for equality between CToken and ZCtoken we compare the extensions.
If they're both known extensions from the same type then we compare the internal values. If not - we return false.

However, in case of unknown extensions they might as well be the same unknown extensions and actually match.

```
JavaScript
                for (zc_ext, regular_ext) in
zc_extensions.iter().zip(regular_extensions.iter()) {
                    match (zc_ext, regular_ext) {
      //....
                    }
                       _ => return false, // Different extension types
                    }
```

Recommendations: Revert on unknown extensions rather than returning false.

Customer's response: Fixed in 3f662d

Fix Review: Fix confirmed

Informational Issues

I-01. account_index check is off by one, leading to a less clear error message

Description: In process_token_compression() we revert with an error message if `account_index > 40`, meaning we don't revert in case that the index is 40.
However, when the index is 40 it would be out of bounds of the `transfer_map` array as well, since the size of the array is 40 (and array index starts from 0, so the index of the last element is 39).
Rust runtime checks would revert when we'll use the index on the array, but the error would be less clear.

Recommendation: Revert if the index is equal to 40 as well

Customer's response: Fixed in 82470c
Fix Review: Fix confirmed

I-02. It's possible to change the state of the compressible config from deprecated back to active

Description: The `unpause_compressible_config()` ix sets the state to 1 (active).
This can be executed also for a deprecated config, which is against the design of the system.

Recommendation: Add a check that the current state isn't deprecated (it would still be possible to pause a deprecated config and then unpause, blocking that route would require some more code changes like holding a separate variable for deprecation or using different bits in the same variable).

Customer's response: Fixed in c01cd8
Fix Review: Fix confirmed

I-03. Make the new authorities signers to protect against address typos

Description: When updating the authorities in `UpdateCompressibleConfig` via `update_compressible_config()` consider making the new authorities signers as well.
This would protect against typos in the new authorities addresses.

Customer's response: Fixed in bae974

Fix Review: Fix confirmed

I-04. Unused accounts and variables

Description:

- `UpdateCompressibleConfig` is used in 4 ixs, but in none of those the `fee_payer` or the system program account are used.
- `_params` isn't used in `create_config_counter()`
- In `MintAction` we require the SPL token accounts whenever `spl_mint_initialized` is true
  In practice we need it only if there's a mint-to action (compressed or SPL) or CreateSplMint action. If we're only updating the mint authorities or metadata we don't need those accounts.
- `CompressibleExtensionInstructionData.has_top_up` isn't used

Recommendation: Remove those accounts and variables

Customer's response: Partially fixed in 445952
Fix Review: Looks good

I-05. Wrong error type thrown in registry's `compress_and_close` ix

Description: In `compress_and_close` ix, when the indices are empty we throw the InvalidSigner error.
This error type doesn't seem to match the actual issue.
Also note that this check is duplicate since the indices are checked not to be empty also in compress_and_close_ctoken_accounts_with_indices()

```rust
    // Validate indices
     require!(!indices.is_empty(), RegistryError::InvalidSigner);
```

Recommendation: Replace with a more fitting error, or rely on the check in `compress_and_close_ctoken_accounts_with_indices()`

Customer's response: Fixed in 7deedc
Fix Review: Fix confirmed

Disclaimer

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.