SPEARBIT

Aztec teegeee Security Review

Auditors

Noah Marconi, Lead Security Researcher

Xmxanuel, Lead Security Researcher

Cryptara, Security Researcher

Chinmay Farkya, Associate Security Researcher

Report prepared by: Lucas Goiriz

March 17, 2025

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Aztec Labs was founded in 2017, and has a team of +50 leading zero-knowledge cryptographers, engineers, and business experts. Aztec Labs is developing its namesake products: AZTEC (a privacy-first L2 on Ethereum) and NOIR (the universal ZK language).

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Aztec teegeee according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- 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.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 3 days in total, Aztec Labs engaged with Spearbit to review the aztec-teegeee protocol. In this period of time a total of **22** issues were found.

Summary

Project Name	Aztec Labs
Repository	aztec-teegeee
Commit	8062e3a6
Type of Project	Governance, Staking
Audit Timeline	Feb 25th to Feb 28th

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	0	0	0
Low Risk	4	3	1
Gas Optimizations	1	0	1
Informational	16	5	11
Total	**22**	**9**	**13**

5 Findings

5.1 High Risk

5.1.1 `beneficiary` can withdraw all their `tokens` in staking by exploiting the `ATP.upgradeStaker` function

Severity: High Risk

Context: *(No context files were provided by the reviewer)*

Description: The `beneficiary` has the power to upgrade the implementation logic of the Staker contract by calling `upgradeStaker`. Only new versions of the `Staker` implementation that have been approved in the `registry` can be used. This mechanism should guarantee that the `beneficiary` can't get access to their tokens in staking. However, the `_initdata` is provided by the `beneficiary` as parameter to call for example a new `initialize` method on the new implementation.

```
function upgradeStaker(StakerVersion _version, bytes memory _initdata)
    external
    override(IATPCore)
    onlyBeneficiary
{
    address impl = REGISTRY.getLinearStakerImplementation(_version);
    UUPSUpgradeable(address(staker)).upgradeToAndCall(impl, _initdata);

    emit StakerUpgraded(_version);
}
```

The `beneficiary` could instead of calling `initialize` call `upgradeToAndCall` again with a new `implementation` address, effectively taking over the `staker` contract with a malicious implementation to withdraw their tokens. Technically, this would result in nested-delegate-calls but the `msg.sender` would remain the `atp` address.

Call Flow:

- `beneficiary` → `atp.upgradeStaker` → `stakerProxy` ⇒ `currStakerImpl.upgradeToAndCall`
- ⇒ `stakerProxy` ⇒ `newCorrectVersionStakerImpl.upgradeToAndCall`
- ⇒ `stakerProxy` ⇒ `attackerStakerVersion.getAllTokens`

Legend:

- → represents `CALL`.
- ⇒ represents `DELEGATE_CALL`.

Proof of Concept: Modified Test File: `test/btt/atps/linear/upgradeStaker/upgradeStaker.t.sol`:

```
contract AttackStaker is FakeLinearStaker {
    constructor(Staking _staking) FakeLinearStaker(_staking) {}
    uint public testSuccess = 0;
    function getAllTokens(address target, uint amount) external onlyATP {
        STAKING.unstake(target, amount);
    }
}
// ---
function test_upgradeStakerAttack() external {
    address expectedStakerBefore = registry.getLinearStakerImplementation(
        StakerVersion.wrap(0)
    );
    address expectedStakerAfter = registry.getLinearStakerImplementation(
        fakeStakerVersion
    );

    assertEq(atp.getStaker().getImplementation(), expectedStakerBefore);
    // attack
    AttackStaker attackStakerImpl = new AttackStaker(staking);
    address attackerDeposit = address(0xeee);

    // move tokens into staking
    aztec.mint(address(this), 1000e18);
    aztec.approve(address(staking), 1000e18);
    staking.stake(address(atp.getStaker()), 1000e18);

    uint amount = token.balanceOf(address(staking));
    uint amountToTake = 1000e18;

    bytes memory data = abi.encodeWithSelector(
        UUPSUpgradeable.upgradeToAndCall.selector,
        address(attackStakerImpl),
        abi.encodeWithSelector(
            attackStakerImpl.getAllTokens.selector,
            attackerDeposit,
            amountToTake
        )
    );
    assertEq(token.balanceOf(attackerDeposit), 0);
    atp.upgradeStaker(fakeStakerVersion, data);
    assertEq(token.balanceOf(attackerDeposit), amountToTake);
}
```

Recommendation: The `beneficiary` can't be allowed to provide the full `_initdata`. Any required `initialize` call should be tracked in the `registry` contract along with the version and implementation.. The `getLinearStakerImplementation` should return the `function signature` required for this upgrade. The full `_initdata` needs to be constructed in the `upgradeStaker` function.

It might introduce additional complexity to manage correct function signatures for different upgrades from different versions to different versions.

For example, Version 2 requires calling a new `initialize` method with additional parameters. Version 3 does not require an `initialize` call if upgraded from Version 2. However, an upgrade from Version 1 to Version 3 would require the `initialize` function to be called again. Storing only a single `function signature` along with Version 3 in the `registry` would not be sufficient.

Consider allowing only upgrades to the direct next version to reduce complexity.

Another way to prevent calling `upgradeToAndCall` again during an upgrade would be a `nonReentrant` modifier in the `LinearBaseStaker` and `MilestoneBaseStaker`.

```
// override upgradeToAndCall with a nonReentrant modifier in LinearBaseStaker and MilestoneBaseStaker
function upgradeToAndCall(address newImplementation, bytes memory data) public override payable
↪  nonReentrant onlyATP {
    super.upgradeToAndCall(newImplementation, data);
}
```

Aztec: Fixed in commit 2aac8775.

Cantina Managed: Fixed.

5.2 Low Risk

5.2.1 Ownership Transfer Lacks Two-Step Process

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: The contracts `ATPFactory`, `Registry`, and `Aztec` implement ownership transfer in a single step rather than using OpenZeppelin's two-step transfer mechanism. This introduces two key risks:

1. Accidental Loss of Administrative Control — If ownership is transferred to an unintended or incorrect address, administrative functionality (such as `addMilestone`, `setMilestoneStatus`, etc...) could become permanently inaccessible.

2. Renouncing Ownership Without Recovery — The current implementation allows direct renouncement of ownership, which could lead to irreversible loss of control over critical functions.

Recommendation: It is recommended to implement OpenZeppelin's two-step ownership transfer mechanism. This approach ensures that the new owner must explicitly accept ownership before the transfer is finalized, preventing situations where ownership is accidentally set to an incorrect or unusable address. Additionally, the `renounceOwnership` method should be overridden to always revert, preventing unintentional renouncement of ownership and preserving administrative access to essential contract functions.

Aztec: The issue has been fixed in PR 69.

Cantina Managed: The code has been reviewed and now utilizes `Ownable2Step` from OpenZeppelin. However, the issue concerning `renounceOwnership()` remains unresolved. The `renounceOwnership()` function bypasses the two-step process and directly assigns `address(0)` as the owner, which can result in the loss of administrative rights. It is assumed that this risk is intentional on the admin's part, as it does not impact any "transfer" logic directly, and the admin would need to invoke the function explicitly.

5.2.2 `MATPCore.updateStakerOperator` is callable by `revoker` after `revoke` with no impact

Severity: Low Risk

Context: MATPCore.sol#L193

Description: After a `revoke` call the new `beneficiary` becomes the `revoker` and `getOperator` will always call `REGISTRY.getRevokerOperator()` instead of returning the `operator` from storage. However, the `revoker` can still call `updateStakerOperator` and update the `operator` in storage, but this has no impact on the `getOperator` function. The `MATPCore.getOperator` function is crucial because it is used by the `MilestoneBaseStaker` to get the operator for the `staker` contract.

```
// MilestoneBaseStaker
function getOperator() public view virtual override(IMilestoneStaker) returns (address) {
    return IMATP(atp).getOperator();
}
```

The behavior of `updateStakerOperator` in this state might lead to the false assumption that the `operator` has changed for the staker contract, which is not the case.

Recommendation: The `MATPCore.updateStakerOperator` should revert after the `revoke` call.

Aztec: Fixed in commit a8d58db3.

Cantina Managed: Fixed.

5.2.3 `beneficiary` might not care about avoiding slashing in `MATP` if they know already the never can reach the `milestone`

Severity: Low Risk

Context: MATPCore.sol#L164

Description: In the `MATP` all tokens can be used for staking before the milestone is reached. In the `README` the following sentences describe the LATP staking:

> This is to ensure that the ATP can be revoked and that economic incentives are simpler to reason about. If Alice could use revokable assets, she might not care about avoiding slashing, if she knows she will be revoked anyway.

The economic incentives for the milestones (`MATP`) are the same. If Alice knows she will never reach the required milestone she might not care about avoiding slashing. Because, she will never get access to the funds.

Recommendation: Consider this fact in the design of the `staking` and `slashing`. If the potential misuse is too high, the last resort would be disallow `MATP` tokens for staking. Another way would be to ensure the `MATP` at least receives the `staking rewards` even if the Milestone is not reached so that they still have some skin in the game.

Aztec: This is acknowledged, but will not be fixed.

Cantina Managed: Acknowledged.

5.2.4 Differing treatment of rescued tokens for revoked ATP beneficiaries

Severity: Low Risk

Context: MATPCore.sol#L142

Description: The `rescueFunds` function intends to `Rescue funds that have been sent to the contract by mistake`. After having their ATP revoked the LATP beneficiaries may rescue any funds provided they are rescuing a token asset other than`TOKEN`. For the MATP beneficiary, the beneficiary is replaced with the revoker leaving rescue no longer possible for them.

Recommendation: Consider unifying the treatment between the two. If it's desirable to have MATP rescue post revocation, consider validating against their address directly, instead of using the `onlyBeneficiary` modifier.

Aztec: Fixed in commit 50f0c1d4.

Cantina Managed: Fixed.

5.3 Gas Optimization

5.3.1 Duplicate checks for `store.isRevokable`

Severity: Gas Optimization

Context: LATPCore.sol#L140

Description: The `revoke` function checks to confirm the ATP is revokable. After performing the check `getAccumulationLock` is called, where the check is performed again. Next `getRevokableAmount` is called where `getAccumulationLock` is called a second time making 3 total times the check for `isRevokable` is made.

Recommendation: The duplicate checks only cost gas. Consider if a refactor for gas saving is worth the risk of introducing an error.

Aztec: Acknowledged.

Cantina Managed: Acknowledged.

5.4 Informational

5.4.1 Comment indicates any token when only `TOKEN` may be recovered

Severity: Informational

Context: ATPFactory.sol#L73

Description: The `owner` may recover `TOKEN`s sent to the factory contract. The comments indicate `any token`, however, the implementation limits to recovering `TOKEN` only.

Recommendation: Update the comment for clarity if this is the intended behavior.

Aztec: Fixed in commit a0fded65.

Cantina Managed: Fixed.

5.4.2 `StakerVersion.wrap(0)` is the only version to be confirmed to be initialized

Severity: Informational

Context: LATPCore.sol#L117

Description: `upgradeStaker` allows upgrading to any version registered on the `REGISTRY`. If there are multiple additional staking versions registered, beneficiaries are permitted to upgrade to them out of order, then fallback to an older version.

Recommendation: Care needs to be taken to ensure one of:

- Order of upgrades are enforced.
- Or out of sequence initialization is safe.

Aztec: Acknowledged.

Cantina Managed: Acknowledged.

5.4.3 Add event when `operator` is updated

Severity: Informational

Context: LinearBaseStaker.sol#L49-L51

Description: Consider adding an event whenever `operator` is updated.

Aztec: Fixed in commit 890ff671.

Cantina Managed: Fixed.

5.4.4 Consider restricting initialization on the implementation contracts

Severity: Informational

Context: LATPCore.sol#L54-L60

Description: While no harm was noted when reviewing the current implementation, there could be irrelevant events and transactions appearing on index sites like etherscan if the implementation is left uninitialized.

Recommendation:

- Use OpenZeppelins `initializer` pattern and modifiers along with `_disableInitializers()` in the constructor.
- Disable initialize on the implementation by setting a value for staker in the constructor.

Aztec: Fixed in commit 85802245.

Cantina Managed: Fixed.

5.4.5 Hardcoded Timing Variables

Severity: Informational

Context: Registry.sol#L68-L69

Description: In the `Registry` contract, the variables `unlockStartTime` and `executeAllowedAt` are statically defined rather than being configurable at deployment. Hardcoding these values reduces flexibility, making it difficult to adjust the contract's behavior based on different deployment environments or upgrade needs. Moreover, because these values are not set via the constructor, any changes require modifying and redeploying the contract, which can be inefficient and costly.

Recommendation: It is advisable to define `unlockStartTime` and `executeAllowedAt` as immutable variables that are set within the constructor. This approach ensures that these values remain constant while allowing them to be customized per deployment. Using `immutable` also optimizes gas costs by storing these values directly in the contract's bytecode instead of storage.

Aztec: The reason that they are statically defined is that we do not know what the value should be at the time of the deployment (e.g., immutables don't help us), but we want to give some guarantees to the beneficiaries. The values are therefore used as an upper limit and can be reduced when more information arrives and we actually know what the values should be.

Cantina Managed: Acknowledged.

5.4.6 Missing Allocation Integrity Check in `LATPCore`

Severity: Informational

Context: LATPCore.sol#L82

Description: In the `LATPCore` contract, during the `initialize` function, there is no verification that the contract's token balance (`TOKEN.balanceOf(address(this))`) is at least equal to `_allocation`. While the factory contract is responsible for ensuring that the correct amount of tokens is transferred, this assumption is external to the `LATPCore` contract itself.

If an unexpected failure occurs in the factory's token transfer process, `LATPCore` might be initialized with fewer tokens than expected, potentially leading to miscalculations or failures when attempting to distribute funds later on. This could result in an inconsistent state where the contract operates under the assumption that more funds are available than actually exist.

Recommendation: It would be beneficial to include a verification step in `initialize` to check that `TOKEN.balanceOf(address(this))` is greater than or equal to `_allocation`. If the condition is not met, the function should revert to prevent initializing the contract in an inconsistent state. This integrity check serves as an additional safeguard, ensuring that the contract is only deployed with the correct amount of tokens.

Aztec: Currently it is the job of the ATPFactory to push funds into the contract so it is mainly a check that adds when we:

- Deploy outside of the ATPFactory (not intended).
- Make changes to ATP factory implementations.

Since the factory is non-upgradable, and I don't expect it to change more after this, I don't think that we gain much from the extra gas the check would make.

Cantina Managed: Acknowledged.

5.4.7 Unrestricted Downgrading

Severity: Informational

Context: LATPCore.sol#L112-L116

Description: In the `LATPCore` contract, the `upgradeStaker` function currently allows downgrading to an older version of the staker contract. This could introduce security risks if a previously deployed staker contract contains vulnerabilities or unintended behavior. The general best practice for upgradeable systems is to only allow version increments to prevent rollbacks to flawed implementations.

However, based on the client's response, downgrading is an intentional feature to allow users to revert to a previous staker contract if needed. The reasoning is that the staking mechanism requires a staker contract to be the registered withdrawer, and upgrades may cause withdrawal issues unless a downgrade path is available.

Recommendation: Since downgrading is necessary for the system's functionality, a controlled downgrade mechanism should be implemented to prevent abuse:

1. Track the last set version — Store the currently assigned version to ensure that downgrades are only allowed under specific conditions.

2. Whitelist trusted versions — Instead of unrestricted downgrading, allow rollbacks only to specific versions that have been audited and approved.
3. Restrict downgrade permissions — If feasible, consider limiting downgrade functionality to trusted roles, such as governance or an admin-controlled contract, rather than allowing arbitrary rollbacks.

This ensures that intentional downgrades remain possible while mitigating risks associated with reverting to vulnerable implementations.

Aztec: As discussed, this is intentional to provide the beneficiaries with the best properties. If a malicious or vulnerable staker is added, it cannot be removed and beneficiaries could "change" into the version (seeing it is a change instead of strictly upgrade likely makes it easier to follow). Since we only allow versions from the registry, the versions are all whitelisted, but yes, bad staker being listed can cause bad things.

Cantina Managed: Acknowledged.

5.4.8 Presence of Typos in Contract and Docs

Severity: Informational

Context: LATPCore.sol#L211

Description: The contract contains typographical errors that may affect readability, maintainability, or even functionality in some cases. While these do not typically introduce security risks, they can lead to confusion for developers and auditors reviewing the contract. Identified typos include:.

- `LATPCore`:
  - `alowance` should be `allowance`.
- Readme:
  - The github project is called `teegeeee` however the readme says `teegeee` in the title.

Recommendation: It is recommended to review and correct these typographical errors to maintain clarity and professionalism in the codebase. Ensuring proper spelling helps improve code readability and reduces potential misunderstandings when interacting with the contract.

Aztec: Issue was fixed in commit 91b97447.

Cantina Managed: Reviewed.

5.4.9 Improper Claim Handling and Dynamic Beneficiary Issues

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: In the `MATPCore` contract, once a milestone fails or the ATP is revoked, the registry revoker is allowed to use the `claim` function to recover funds. This creates several conceptual and functional issues:

1. Misleading Event Emission and State Updates — The revoker calling `claim` will trigger a `Claimed` event and update the `claimed` storage variable, even though the revoker is not actually claiming but rather recovering funds. This results in misleading logs and state changes.
2. Overloaded Functionality — The same `claim` function is used for both beneficiaries and the revoker, making it difficult to differentiate between legitimate claims and fund recovery.
3. Dynamic `onlyBeneficiary` Modifier — The `getBeneficiary` function dynamically returns the revoker as the beneficiary if the milestone is failed or ATP is revoked. This allows the revoker to call functions like `approveStaker`, potentially affecting staking mechanics in unintended ways.

Additionally, allowing staking-related actions (`approveStaker`, staking flows) while the ATP is revoked or the milestone has failed raises concerns about integrity and whether these actions should still be valid in such scenarios.

Recommendation: To improve clarity and security:.

1. Introduce a Separate Recovery Function -– Instead of overloading `claim`, create a `revokeWithdraw` function that is exclusively callable by the revoker when an ATP is revoked or a milestone has failed. This function should allow multiple calls, as funds may still be locked in the staker over time.
2. Make `onlyBeneficiary` Non-Dynamic — The current approach of dynamically assigning the revoker as the beneficiary can lead to integrity issues. Instead, ensure that `onlyBeneficiary` applies strictly to the original beneficiary and does not shift ownership dynamically.
3. Clarify Staking Flow Post-Revocation — If the design allows for ATP takeovers and closing of staked positions, explicitly enforce that only necessary staking-related actions remain available after revocation, rather than leaving it open-ended. It may be necessary to limit or restrict `approveStaker` under these conditions to prevent unintended interactions.

By introducing a clear separation between claims and revocations, and refining access control mechanisms, the contract will have better state integrity, security, and clarity in handling failed milestones and revoked ATPs.

Aztec: Acknowledged, but won't be fixed. The separation would cause a bunch of near-duplicates and as long as the revoker is not given a bunch of other opportunities than the user on staking etc it seems fine to keep as is.

Cantina Managed: Acknowledged.

5.4.10 `revokeBeneficary` can not be changed by governance in `LATP`

Severity: Informational

Context: LATPCore.sol#L97

Description: The `revokeBeneficiary` is defined in the `initialize` method of the `LATP` and would receive the `tokens` in the event of a `revoke`. This behavior differs from `MATP`, where the `revoker` address can `claim` the tokens after a `revoke`. The `revokeBeneficiary` in `LATP` cannot be changed in any way, and a revoke could occur multiple years in the future.

If the `revokeBeneficiary` address is compromised, a revoke operation cannot be executed on the ATP without losing the tokens. More likely, Aztec simply wants to change its key setup in the future, similar to other multi-sig addresses. It would still be necessary to maintain the legacy addresses for the `revokeBeneficiary` due to old employees/ATPs.

Recommendation: Instead of defining a specific `revokeBeneficiary` for `LATP`, the `revoker` itself could be used, as in `MATP`, to receive the tokens. Alternatively, the `registry` could manage the `revokeBeneficiary` with update functions.

Aztec: This is on purpose, as it was a requirement that it was specified at the time of creation. The main reason behind it being that while there might be just a single revoker entity, the location where funds should go to, could be split between the foundation and labs.

Cantina Managed: Acknowledged.

5.4.11 `LATP` and `MATP` handle staking rewards differently in case of a `revoke`

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The `LATP` and `MATP` handle staking rewards differently in the case of a `revoke`. In the `LATP` case, the revoker only has a claim on the `getRevokableAmount` but not on any additional staking rewards. In the `MATP` case, the `revoker` essentially takes over the `beneficiary` role and can also claim any potential rewards. It is not possible to `claim` rewards by the `beneficiary` before the `revoke` is executed.

Recommendation: Consider whether the `MATP` should also have a claim on the `staking rewards` as a mechanism to ensure there is always some skin in the game, particularly when a successful milestone has become unrealistic.

Aztec: This is intentional. In LATP, you earn only on assets you already own, while in MATP, you earn on assets you'll own eventually. We prefer keeping rewards consistent: earned on your assets $\rightarrow$ yours, earned on future assets $\rightarrow$ not yours yet.

Cantina Managed: Acknowledged.

5.4.12 Prevent sending `Aztec` tokens to the `Aztec` contract

Severity: Informational

Context: Aztec.sol#L10

Description: In the `teegeeee` projects multiple contracts have `rescueFunds` functions to withdraw any potentially locked ERC20 token. However, users could still accidentally send Aztec tokens to the Aztec contract, causing them to become locked.

Recommendation: Unintended transfers of Aztec tokens to the Aztec ERC20 contract itself should also be prevented. The following pattern can be used to assign the maximum value to the contract's balance without affecting the total supply:

```
_balances[address(this)] = type(uint256).max;
```

This would cause any `transfer` or `transferFrom` to the `Aztec token address to revert with a math overflow. Alternatively, just adding a simple check` require(to != address(this))‘ to the _update function by overwriting it.

Aztec: Acknowledged, but won't be addressed. As mentioned the `_balances` is a private storage variable so cannot be accessed as recommended. We could override the `_update` but would rather keep the diff minimal.

Cantina Managed: Acknowledged.

5.4.13 Add basic sanity checks to the `ATP.upgradeStaker` functions to verify correct upgrade

Severity: Informational

Context: LATPCore.sol#L118

Description: The new `staker` version added to the `registry` could be any kind of implementation and also theoretically contain errors like storage collisions. The only verified requirement is that the new implementation uses the `ERC1967Utils.IMPLEMENTATION_SLOT` storage slot.

```
// in registry.registerLinearStakerImplementation
UUPSUpgradeable(_implementation).proxiableUUID() == ERC1967Utils.IMPLEMENTATION_SLOT
```

Recommendation: Add a sanity check to `upgradeStaker`. The `ATP` needs to be the correct address after the upgrade call:

```
require(staker.getATP() == address(this), InvalidATPAfterUpgrade);
```

Aztec: Fixed in commit 745b2898.

Cantina Managed: Fixed.

5.4.14 Consider range checks for `registry.setUnlockStartTime`

Severity: Informational

Context: Registry.sol#L199

Description: The `registry.setUnlockStartTime` is very powerful and can release a high amount of new tokens. It is only possible to set the `unlockStartTime` to an earlier date in the past to prevent additional locks and complexity.

Recommendation: Consider additional range checks like for example only a maximum decrease of `x days` per `setUnlockStartTime` or like a lower limit, which would unlock all tokens in the global lock.

Aztec: This is acknowledged, but will not be fixed.

Cantina Managed: Acknowledged.

5.4.15 `revoker` can get access to tokens before the global lock cliffs ends

Severity: Informational

Context: LATPCore.sol#L139

Description: The `revoker` role is powerful and can `revoke` any `ATP`. It can get `tokens` out of the `ATP` before the global unlock ends.

Recommendation: This might be not a problem since the `revoker` is governance controlled role but we created the issue for informational purposes.

Aztec: This is an acceptable risk as the revoker entity is going to be the same as admin, and they are trusted.

Cantina Managed: Acknowledged.

5.4.16 `REGISTRY.getRevokerOperator` can be `address(0)` and `revoker` can't change the `revokeOperator`

Severity: Informational

Context: MATPCore.sol#L252

Description: If an `MATP` get revoked the new `operator` is the `REGISTRY.getRevokerOperator`. However, the `getRevokerOperator` might be not set in the `registry`. The `MATP.getOperator` function would return `address(0)` and the `staker` related operator functions would not be callable. The permission for the `revoker` to add `revokerOperator` does not exist. Both roles `revokeOperator` and `revoker` are controlled by the governance:

```
function setRevoker(address _revoker) external override(IRegistry) onlyOwner {
    revoker = _revoker;
    emit UpdatedRevoker(_revoker);
}

function setRevokerOperator(address _revokerOperator) external override(IRegistry) onlyOwner {
    revokerOperator = _revokerOperator;
    emit UpdatedRevokerOperator(_revokerOperator);
}
```

Recommendation: Since the `revokerOperator` role is a safety mechanism for the revoker actor to not directly interact with the `staker` contract. Consider, allowing the `revoker` to change the `revokeOperator` in the registry.

Aztec: Acknowledged, but won't be addressed.

Cantina Managed: Acknowledged.