SPEARBIT

Redacted Dinero Pirex ETH Security Review

Auditors

0xRajeev, Lead Security Researcher

0x52, Lead Security Researcher

Slowfi, Security Researcher

Ayeslick, Junior Security Researcher

Report prepared by: Lucas Goiriz

April 24, 2024

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

The Redacted ecosystem is a product suite of smart contracts empowering on-chain liquidity, governance, and cash flow for DeFi protocols.

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

The following directories/files are part of the scope:

Directory
`src/libraries`
`src/AutoPxEthGatway.sol`
`src/InstitutionalPirexEth.sol` (and its dependencies)
`src/InstitutionalPirexEthValidators.sol` (and its dependencies)

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 10 days in total, Redacted engaged with Spearbit to review the dinero-pirex-eth protocol. In this period of time a total of **18** issues were found.

Summary

Project Name	Redacted
Repository	dinero-pirex-eth
Commit	306def...a789c7
Type of Project	Staking, DeFi
Audit Timeline	Mar 11 to Mar 22
Two week fix period	Mar 28 - Mar 29

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	1	1	0
High Risk	1	1	0
Medium Risk	5	5	0
Low Risk	6	4	2
Gas Optimizations	3	2	1
Informational	2	2	0
Total	**18**	**15**	**3**

5 Findings

5.1 Critical Risk

5.1.1 Sharing `pxEth` rewards between retail-institution sides in the shared `AutoPxEth` vault breaks protocol requirements and leads to insolvency

Severity: Critical Risk

Context: PxEth.sol, AutoPxEth.sol

Description: The motivating rationale for this upgrade to the Pirex ETH liquid staking protocol is to augment the existing retail flows with a parallel set of flows for institutional clients. The key requirement is to separate the retail and institutional interactions with the protocol by deploying a new staking validator set and an institution-specific set of contracts (`iupxEth`, `institutionalPxEth`, `oracleAdapter`, `rewardRecipient`, `autoPxEthGateway` and `institutionalPirexEth` along with their associated libraries). The retail contracts `pxEth`, `pirexFees` and `autoPxEth` will be reused even for institutional flows.

The communicated retail-institution separation requirements are:

1. Validators spun up by ETH depositors to institution contracts should be different from retail counterparts.
2. Rewards earned by validators spun up via retail and institution should be separately managed via different `rewardRecipient`.
3. `pxEth` and `apxEth` tokens should never be held by any institutional depositors or protocol entities interacting with the institutional flows.

However, sharing the same `AutoPxEth` vault and `pxEth` (as the core LST) between retail and institution sides leads to sharing of retail/institution validator `pxEth` rewards with everyone but is backed by rewarded validator ETH (used for the newly spun up validators) only on the side (either retail or institution) receiving the actual validator rewards. This effectively breaks the 1:1 pxEth-ETH peg on both sides: over-collaterized (more ETH than `pxEth`) on the side receiving the ETH/pxEth rewards and under-collaterized (less ETH than `pxEth`) on the other side, i.e. the side whose validator set was not responsible for rewards becomes insolvent because its `pxEth` balance is not backed up by an equivalent ETH staked in its validator set.

This is illustrated in the below simplified scenario:

1. Retail user *R* deposits 32 ETH with compounding, yielding them 32 `apxEth` with 32 `pxEth` deposited in the vault (this spins up one retail validator).
2. Institutional user *I* deposits 32 ETH yielding them 32 wrapped-apxEth with another 32 `pxEth` deposited in the vault (this spins up one institutional validator).
3. Let's assume, for the sake of illustration purposes, that the retail validator gets lucky to be chosen for a block proposal with a huge MEV reward of 64 ETH, which gets harvested resulting in 64 `pxEth` being deposited into the vault and another two retail validators being spun up. The vault now has 32 + 32 = 64 `apxEth` share supply and 32 + 32 + 64 = 128 `pxEth` assets, i.e. 1:2 share-to-asset ratio.
4. At this point, given that the 64 `pxEth` rewards are effectively averaged across all the `apxEth` shares in the shared vault, the 32 `apxEth` shares on the retail side have 32 + 32 = 64 `pxEth` assets which are backed by $1+2=3$ validators, i.e. $3\times32=96$ ETH - the retail side is over-collateralized. However, the 32 `apxEth` shares on the institutional side have 32 + 32 = 64 `pxEth` assets which are backed by only the 1 initial validator, i.e. 32 ETH - the institutional side is under-collateralized.
5. If the institutional user *I* wants to then redeem their 32 wrapped `apxEth` shares for the now underlying 64 pxEth (32 original + 32 rewards), it is not possible because the institutional side only has one validator backed by 32 ETH balance and cannot meet the 64 ETH effective obligation by exiting/withdrawing that validator, i.e. the institutional side of the protocol is insolvent.

- Impact: Sharing `pxEth` rewards between retail and institutional sides in the same `AutoPxEth` vault while separating their validator sets (and thereby redemptions) will lead to protocol insolvency on one of the retail or institutional sides whenever rewards are harvested.

- Likelihood: High (Happens throughout the protocol lifetime whenever rewards are harvested) + Impact: High (1. Breaks the critical 1:1 pxEth-ETH peg protocol invariant on one of the retail or institution sides 2. Breaks protocol's retail-institution separation requirement for rewards 3. Protocol is effectively insolvent) = Severity: Critical.

Recommendation: Reconsider the design of sharing `pxEth` rewards in the shared `AutoPxEth` vault to instead evaluate the use of separate vaults and different LSTs in addition to the separated validator sets to achieve the best separation possible between retail and institutional users.

Redacted Cartel: Addressed as part of the consolidated changes in PR 57.

Spearbit: Reviewed that PR 57, as recommended, significantly re-architects the institutional side of the protocol to use a separate `pxEth` token and its associated `institutionalAutoPxEth` vault, both of which are different and isolated from the existing retail token and its vault. This major refactoring is summarized in PR 54, whose key aspects are quoted below:

> Refactor current institutional system into a separate vault system not tied into using pxEth as the underlying token, which would avoid the need for the AutoPxEthGateway and prevent buffer issues among others.
>
> - Use a new (generic) pxEth deployment to be used as the underlying token for the institutional system (instead of the official pxEth used in the retail system).
> - Deploy a new AutoPxEth vault using the same new (generic) pxEth above, which itself will be the underlying token for the wrapper (ipxEth).
> - We will deploy the new RewardRecipient (to be called as RewardRecipientGateway from this point on), which will acts as the intermediary for the legacy RewardRecipient rewards as well as the new RewardRecipient responsible for receiving and forwarding rewards from validators using the new credential. This new recipient system will allow us to properly distribute rewards proportionately between the 2 PirexEth systems (retail and institutional). Aside from that, the new recipient will also be responsible for the slash and dissolve operations between the 2 systems (the new recipient will be able to differentiate which system a validator (pubKey) belongs to).

The re-architected design introduces a `RewardRecipientGateway` to act as an "*intermediary*" between retail and institution flows and distribute rewards proportionately (based on the underlying vault assets) between the 2 PirexEth systems (retail and institutional). We reviewed these changes and identified two serious issues in them:

1. Double counting of `msg.value` if/when legacy `RewardRecipient` needs to slash an older validator.
2. Dissolve validator flow will revert for legacy validators because of mismatched parameter counts between `RewardRecipient.dissolveValidator()` and `RewardRecipientGateway.dissolveValidator()`).

We suggested further mitigations, which have since been addressed, and recommend more validation of newly introduced components and flows.

While the retail-institutional proportional distribution splits the rewarded ETH (leading to newly spun validators) and the equivalent minted `pxEth` between retail and institutional sides to maintain the ETH-`pxEth` 1:1 peg on both sides, we discussed the following aspects with the protocol team:

> 1. Why shouldn't the retail rewards/yield be reserved only for the retail `pxEth` stakers (and vice-versa)? I.e. strict separation instead of proportional distribution?
> 2. Given that retail `pxEth` staking is optional but mandatory for institution, won't this yield distribution favor institution stakers at the expense of retail for retail-validator-rewards? Is this institutional-bias intentional for increasing their adoption at the expense of retail and if so, is this a business strategy?
> 3. If we assume institutional staking amounts will be much greater than retail amounts, (2) will be magnified to favor institution and drastically lower existing retail yields.

After reviewing the above points, the protocol team acknowledged these aspects and communicated that they will stick to the current design.

5.2 High Risk

5.2.1 Using `pxEth.totalSupply()` shared between retail-institution for determining `maxBufferSize` leads to lower yield and breaks protocol separation requirement

Severity: High Risk

Context: InstitutionalPirexEthConfigurationLogic.sol#L358-L361, InstitutionalPirexEthConfigurationLogic.sol#L380-L383, InstitutionalPirexEthDepositLogic.sol#L324-L340, PirexEthValidators.sol#L863-L86, PirexEthValidators.sol#L876-L878, PirexEthValidators.sol#L959-L962

Description: `maxBufferSize` is updated for both retail and institution flows based on the common/shared `pxEth.totalSupply()` which represents the combined ETH deposited into both sides. This leads to an inflated `maxBufferSize` than expected for both sides, which is technically supposed to be based only on deposits on either the retail or institutional side of the protocol that is relevant for each `maxBufferSize`.

- Impact: More ETH than expected is retained in their respective buffers for instant withdrawals and emergency top-ups, which leads to lesser staked ETH used for spinning up validators and therefore lower yield from rewards.

  This also breaks protocol's retail-institution separation requirement because deposits on one side will increase the `maxBufferSize` and lower the yield on both sides. For e.g., a large deposit on the retail side will reduce the effective yield on the institutional side. This issue also persists with redemptions, slashing and harvesting on both sides leading to unexpected accounting and behavior.

- Likelihood: High (Happens throughout the lifetime of protocol) + Impact: Medium (1. Leads to lower yield from lesser staked ETH in validators 2. Breaks protocol retail-institution separation requirement) = Severity: High.

Recommendation: Consider logic to separately track staked ETH on retail and institutional sides without using the shared `pxEth.totalSupply()`.

Redacted Cartel: Addressed as part of the consolidated changes in PR 57.

Spearbit: Reviewed that PR 57, as recommended, does not use the common total supply of `pxEth` to calculate buffer space on both retail and institutional sides. The fix introduces a separate `pxEth` token (and associated vault) for the institutional side that is different and isolated from the existing retail side, which effectively resolves this accounting issue arising from the earlier shared token.

5.3 Medium Risk

5.3.1 Using `AutoPxEth`'s `transfer()` and `transferFrom()` for initiating redemptions may result in loss of funds after institutional setup

Severity: Medium Risk

Context: AutoPxEth.sol#L448-L488, PirexEth.sol#L365-L372, AutoPxEthGateway.sol#L129-L135

Description: Currently, `AutoPxEth`'s `transfer()` and `transferFrom()` call `pirexEth.initiateRedemption` when `to == address(pirexEth)`, which means that one can initiate redemption by directly calling them with the transfer recipient of `pirexEth` instead of withdrawing pxEth from the vault first and then calling `pirexEth.initiateRedemption()`.

However, with the institutional setup, `AutoPxEth.pirexEth` is set to point to `AutoPxEthGateway` whose `initiateRedemption()` always reverts. So any previously working retail-side calls to `AutoPxEth`'s `transfer()` and `transferFrom()` with the recipient being the original `pirexEth` address will only transfer the pxEth to that contract but not trigger a redemption after the institutional setup migration.

- Impact: Using `AutoPxEth`'s `transfer()` and `transferFrom()` for initiating retail-side PirexEth redemptions will result in loss of funds after institutional setup migration.

- Likelihood: Low (possible that users could be using this single-step redemption flow) + Impact: High (Loss of redeemed funds) = Severity: Medium.

Recommendation: Reconsider the shared vault and intermediate `AutoPxEthGateway` design. At a minimum, ensure that documentation and UI flag this critical change in expected behavior after institutional setup migration.

Redacted Cartel: Addressed as part of the consolidated changes in PR 57.

Spearbit: Reviewed that PR 57 has addressed this with the removal of `AutoPxEthGateway.sol` entirely in its redesign and by the introduction of `redirectEnabled` setting which is used to conditionally call `pirexEth.initiateRedemption()` from any new deployment of `AutoPxEth`'s `transfer()` and `transferFrom()`.

5.3.2 Using `pxEth` for `pirexFees` breaks protocol's institution-retail separation requirement

Severity: Medium Risk

Context: InstitutionalPirexEthDepositLogic.sol#L131, InstitutionalPirexEthDepositLogic.sol#L506-L518, InstitutionalPirexEthWithdrawLogic.sol#L201. InstitutionalPirexEthWithdrawLogic.sol#L350-L354, InstitutionalPirexEthWithdrawLogic.sol#L501-L513

Description: The protocol approves the transfer of `feeAmount` worth of `pxEth` to the `pirexFees` contract during deposits and withdrawals. While this works for the existing retail `PirexEth` flows, it breaks an important separation requirement of the protocol's institutional setup where institutional participants/components should only be able to redeem via institutional flows without having to depend on any retail flows. Institutional flows only allow redeeming via `IpxEth` or `IUpxEth`, not `pxEth`.

The communicated retail-institution separation requirements are: 1) Validators spun up by ETH depositors to institution contracts should be different from retail counterparts 2) Rewards earned by validators spun up via retail and institution should be separately managed via different `rewardRecipient` 3) `pxEth` and `apxEth` tokens should never be held by any institutional depositors or protocol entities interacting with the institutional flows.

- Impact: Using `pxEth` for `pirexFees` during institutional deposits/withdrawals requires later redeeming them using retail flow, which breaks the protocol's institution-retail separation requirement.
- Likelihood: Medium (Requires fees to be non-zero) + Impact: Medium (Breaks protocol's separation requirement) = Severity: Medium.

Recommendation: Consider using wrapped `apxEth` (i.e. `IPxEth`) instead of `pxEth` for `pirexFees`.

Redacted Cartel: Addressed as part of the consolidated changes in PR 57.

Spearbit: Reviewed that PR 57 redesigns the protocol architecture to separate retail and institutional systems. With separated `pxETH` and `pirexFees` contracts this issue is resolved.

5.3.3 Minting `pxEth` to validators breaks protocol's institution-retail separation requirement

Severity: Medium Risk

Context: InstitutionalPirexEthDepositLogic.sol#L444-L451

Description: As a mitigation for deposit front-running exploits from any compromised validators, the protocol uses the notion of pre-deposits to register authorized withdrawal-credentials for pubkeys of all initialized validators. When new validators are spun up for staking, `pxEth` equivalent to any pre-deposited ETH amount is minted to the validator-specified receiver. While this works for the existing retail PirexEth flows, it breaks an important separation requirement of the protocol's institutional setup where institutional participants/components should only be able to redeem via institutional flows without having to depend on any retail flows. Institutional flows only allow redeeming via `IpxEth` or `IUpxEth`, not `pxEth`.

The communicated retail-institution separation requirements are:

1. Validators spun up by ETH depositors to institution contracts should be different from retail counterparts.
2. Rewards earned by validators spun up via retail and institution should be separately managed via different `rewardRecipient`.
3. `pxEth` and `apxEth` tokens should never be held by any institutional depositors or protocol entities interacting with the institutional flows.

- Impact: Minting `pxEth` to institutional validators for any pre-deposits requires them to redeem using retail flow, which breaks the protocol's institution-retail separation requirement.
- Likelihood: Medium (Requires the use of pre-deposits) + Impact: Medium (Breaks protocol's separation requirement) = Severity: Medium.

Recommendation: Consider minting wrapped `apxEth` (i.e. `IPxEth`) instead of `pxEth` towards pre-deposits of validators.

Redacted Cartel: Addressed as part of the consolidated changes in PR 57.

Spearbit: Reviewed that PR 57, as recommended, mints `preDepositAmount` of wrapped `apxEth` to `validatorParams.receiver`, instead of minting `pxEth` to it.

5.3.4 `AutoPxEthGateway.withdraw()` implementation is incorrect

Severity: Medium Risk

Context: AutoPxEthGateway.sol#L103-L111

Description: `AutoPxEthGateway.withdraw()` is expected to withdraw the specified amount of `_assets` from the `AutoPxEth` contract by burning the corresponding `AutoPxEth` shares from `msg.sender` and transferring the `_assets` to the `_receiver`.

However, the current implementation attempts to transfer `_assets` from `msg.sender` and withdraw the same to the `_receiver`, while ignoring the `_shares`:

```
_shares = autoPxEth.previewWithdraw(_assets);
autoPxEth.safeTransferFrom(msg.sender, address(this), _assets);
autoPxEth.withdraw(_assets, _receiver, address(this));
```

- Likelihood: Low (`AutoPxEthGateway.withdraw()` is currently unused in favor of `redeem()` by `PIREX_ETH_ROLE`) + Impact: High (Failed/Incorrect withdrawals) = Severity: Medium.

Recommendation: Consider either changing L109 to be `autoPxEth.safeTransferFrom(msg.sender, address(this), _shares);` or replacing L108-L110 with `_shares = autoPxEth.withdraw(_assets, _receiver, msg.sender);`

Redacted Cartel: Addressed as part of the consolidated changes in PR 57.

Spearbit: Reviewed that PR 57 has removed this contract entirely in its redesign.

5.3.5 Requiring `INVESTOR_ROLE` for institutional redeem flows will prevent composability and allow censorship

Severity: Medium Risk

Context: InstitutionalPirexEth.sol#L191, InstitutionalPirexEth.sol#L229, InstitutionalPirexEth.sol#L261, InstitutionalPirexEth.sol#L284, InstitutionalPirexEth.sol#L311

Description: While only authorized institutional users are allowed to deposit via `InstitutionalPirexEth`, enforcing the `INVESTOR_ROLE` for institutional redeem flows will prevent institutional users from participating in DeFi protocols using their wrapped `apxEth` shares as an asset or collateral. This is because non-institutional users will be considered as unauthorized by Institutional Pirex ETH protocol to redeem any wrapped `apxEth` shares they obtain from institutional users in other DeFi protocols or via a simple trade. This will prevent composability of the Institutional Pirex ETH protocol within the DeFi ecosystem and create a closed system, which is not the desired goal.

Requiring `INVESTOR_ROLE` for institutional redeem flows will also allow censorship from protocol governance by allowing the revocation of `INVESTOR_ROLE` role after deposits to lock them out of redeems.

- Likelihood: High (DeFi composability is broken) + Severity: Low (Loss of yield from lack of DeFi composability) = Impact: Medium.

Recommendation: Consider removing the `INVESTOR_ROLE` for institutional redeem flows.

Redacted Cartel: Addressed as part of the consolidated changes in PR 57.

Spearbit: Reviewed that PR 57, as recommended, removes `onlyRole(Constants.INVESTOR_ROLE)` modifier for redeem flows at `initiateRedemption()`, `redeemWithIUpxEth()`, `bulkRedeemWithIUpxEth()` and `instantRedeemWithIPxEth()`.

5.4 Low Risk

5.4.1 `topUpStake()` can be called when ETH deposits are paused

Severity: Low Risk

Context: InstitutionalPirexEthDepositLogic.sol#L162-L185, InstitutionalPirexEthValidationLogic.sol#L216-L222

Description: Unlike depositPrivileged(), which checks if ETH deposits are paused, `topUpStake()` does not validate `pirexEthValidatorVars.depositEtherPaused == Constants._PAUSED`, which may lead to unexpected behavior.

- Impact: Topping up stake when the depositing ETH is paused may lead to unexpected behavior.
- Likelihood: Low (execution is possible only by privileged KEEPER role) + Impact: Medium (ETH may be deposited when deposits are paused) = Severity: Low.

Recommendation: Validate the state of ETH deposits being paused while topping up stake.

Redacted Cartel: Acknowledged with no action taken. By design it's done this way so we can act swiftly (as this would only be called on lowBalance or slashed validators scenario). Proper handling will be enforced before calling `topUp`.

Spearbit: Acknowledged.

5.4.2 Not updating `outstandingRedemptions` in `emergencyWithdraw()` may leave protocol insolvent

Severity: Low Risk

Context: InstitutionalPirexEthWithdrawLogic.sol#L122-L149

Description: Similar to M-01 of the previous audit which raised the issue of non-updation of `pendingDeposit` in `emergencyWithdraw()` (and has since been mitigated), not updating `outstandingRedemptions` in `emergencyWithdraw()` may leave protocol insolvent if the protocol were to be resumed after such an emergency withdrawal.

- Impact: `outstandingRedemptions` may not be backed by sufficient ETH in the protocol for redemptions after an emergency withdrawal and therefore leave protocol insolvent.
- Likelihood: Very Low (`emergencyWithdraw()` is meant as an extreme measure) + Impact: High (Protocol may be insolvent) = Severity: Low.

Recommendation: Evaluate if prioritizing updation of `outstandingRedemptions` over `pendingDeposit` in `emergencyWithdraw()` is better to allow withdrawals to continue instead of deposits.

Redacted Cartel: Addressed as part of the consolidated changes in PR 57.

Spearbit: Reviewed that PR 57, as recommended, considers `outstandingRedemptions` in the calculation of `remainingBalance` to update `pendingDeposit` accordingly in `executeEmergencyWithdraw()`.

5.4.3 `AutoPxEth.previewWithdraw()` rounds in a direction that disadvantages the vault

Severity: Low Risk

Context: AutoPxEth.sol#L401-L416, ERC4626.sol#L146-L150, ERC4626.sol#L124-L128

Description: `AutoPxEth.previewWithdraw()` overrides ERC4626 to implement a "modified version takes into consideration the withdrawal fee". However, unlike `ERC4626.previewWithdraw()` which rounds up the required user shares via `assets.mulDivUp(supply, totalAssets())`, this uses `ERC4626.convertToShares()` which rounds down the required user shares via `assets.mulDivDown(supply, totalAssets())` and thus effectively requiring less shares from the user in cases where rounding up/down matters, i.e. resulting in slight loss of vault funds during withdrawal.

- Impact: `AutoPxEth.previewWithdraw()` rounds in a direction that disadvantages the vault.
- Likelihood: Low (Applies to withdrawal when rounding up/down matters) + Impact: Low (Minor loss of vault funds during withdrawals) = Severity: Low.

Recommendation: Consider using `assets.mulDivUp(supply, totalAssets())` as in `ERC4626.previewWithdraw()` for share conversion. Evaluate if this mitigation warrants a redeployment of `AutoPxEth`.

Redacted Cartel: Addressed as part of the consolidated changes in PR 57. Since we will doing 2 token system to alleviate various issue with commingling between retail and institutional, which requires deployment of both a separate `pxEth` and `autoPxEth` just for institutional, we will be implementing the rounding fix above just for it. Note that we won't be redeploying the existing `autoPxEth` currently live for `pxEth`

Spearbit: Reviewed that PR 57, as recommended, replaces the use of `ERC4626.convertToShares()` with `assets.mulDivUp(supply, totalAssets())` in `previewWithdraw()`. This updated `AutoPxEth` will be used for the institutional side whereas the deployed retail one will not be modified.

5.4.4 Allowing resetting of key protocol addresses may result in unexpected behavior

Severity: Low Risk

Context: InstitutionalPirexEthValidators.sol#L298-L308, InstitutionalPirexEthConfigurationLogic.sol#L124-L205

Description: `setContract()` allows protocol governance to set/reset key protocol addresses of `UpxEth`, `PxEth`, `AutoPxEth`, `InstitutionalPxEth`, `OracleAdapter` and `RewardRecipient`. Resetting these protocol addresses has to be performed with great caution because they could lead to unexpected behavior for those interacting/integrating with the older addresses. For example, if the `RewardRecipient` is reset then the expectation, among others, will be that Keepers will keep harvesting all balance ETH rewards sent to the older/replaced `RewardRecipient`.

- Impact: Allowing resetting of key protocol addresses may result in unexpected behavior unless all interactions/integrations are carefully considered.
- Likelihood: Very Low (Resetting should rarely happen) + Impact: Medium (Unexpected behavior if not all interactions/integrations are carefully considered) = Severity: Low.

Recommendation: Carefully consider all interactions/integrations and only allow resetting of select addresses as anticipated. For all other addresses, programmatically prevent resets.

Redacted Cartel: Acknowledged with no action taken for now.

Spearbit: Acknowledged.

5.4.5 Returning `postFeeAmount` instead of `apxEthShares` during deposits may break any integrations

Severity: Low Risk

Context: InstitutionalPirexEthDepositLogic.sol#L108-L147

Description: While retail deposit flow mints `postFeeAmount` of `pxEth` for the receiver, the equivalent institution deposit flow mints `postFeeAmount` of `pxEth`, deposits that into the `AutoPxEth` vault to receive `apxEthShares` of `apxEth` and wraps that 1:1 for `institutionalPxEth` or `IPxEth` to the receiver. So the institutional receiver receives `apxEthShares` worth of `IPxEth`. However, the institutional deposit function returns `postFeeAmount` (similar to the retail deposit function) instead of returning `apxEthShares` to indicate the correct amount of minted shares, .

- Impact: Any DeFi integrations with Institutional Pirex protocol that expect a return value of `apxEthShares` will break.
- Likelihood: Low (Requires an integration with this expected output) + Impact: Low (Broken integration) = Severity: Low.

Recommendation: Consider returning `apxEthShares` instead of `postFeeAmount`.

Redacted Cartel: Addressed as part of the consolidated changes in PR 57.

Spearbit: Reviewed that PR 57, as recommended, has `executeDeposit()` returning `apxEthShares` amount.

5.4.6 `emergencyWithdraw()` breaks critical protocol invariant of 1:1 `pxEth` peg

Severity: Low Risk

Context: InstitutionalPirexEthWithdrawLogic.sol#L136-L145

Description: `emergencyWithdraw()` is meant to be an emergency withdrawal mechanism of ETH and all tokens (except `pxEth`) by protocol governance. However, when ETH is removed using this mechanism and if/when protocol continues operations, it does not burn a corresponding amount of `pxEth`.

- Impact: This breaks a critical protocol invariant of 1:1 `pxEth` peg, which means that there will be more `pxEth` in the protocol than what can be redeemed against protocol-accounted ETH.
- Likelihood: Very Low (`emergencyWithdraw()` is meant as an extreme measure) + Impact: High (Critical protocol invariant violation) = Low severity.

Recommendation: Consider burning from governance approved/managed burner accounts an amount of `pxEth` equivalent to ETH withdrawn using `emergencyWithdraw()`.

Redacted Cartel: Addressed as part of the consolidated changes in PR 57.

Spearbit: Reviewed that PR 57, as recommended, burns `amount` of `pxEth` from burner accounts by calling `updateBuffer()` in `emergencyWithdraw()` when token is ETH.

5.5 Gas Optimization

5.5.1 Removing unused code may save gas

Severity: Gas Optimization

Context: InstitutionalPirexEthDepositLogic.sol#L485

Description: The function `getApxEth()` is not used in the `InstitutionalPirexEthDepositLogic` library. A similar function exists in and is used by the `InstitutionalPirexEthWithdrawLogic`.

Recommendation: Consider removing unused code to improve code quality and save gas.

Redacted Cartel: Addressed as part of the consolidated changes in PR 57.

Spearbit: Reviewed that PR 57, as recommended, removes the unused function.

5.5.2 Using constants directly may save gas

Severity: Gas Optimization

Context: InstitutionalPirexEthValidators.sol#L409

Description: The function `slashValidator()` assigns the `DEPOSIT_SIZE` constant to a variable and then uses that variable throughout the function code. This is unnecessary and can be optimized by directly using the constant everywhere.

Recommendation: Avoid reassigning constants to variables and potentially save gas.

Redacted Cartel: Addressed as part of the consolidated changes in PR 57.

Spearbit: Reviewed that PR 57, as recommended, fixes the issue by using the constant instead of reassigning it to a local variable.

5.5.3 Refactoring duplicated code may save gas

Severity: Gas Optimization

Context: validateEmergencyWithdraw, validateDeposit, validateInitiateRedemption, validateRedeemWithIUpx-Eth, validateInstantRedeemWithIPxEth

Description: The same code snippet shown below is repeated in all the above functions:

```
if (receiver == address(0)) revert Errors.ZeroAddress();
if (amount == 0) revert Errors.ZeroAmount();
```

Recommendation: Consider avoiding duplicated code by moving these checks to a common internal function to decrease contract size and potentially save gas.

Redacted Cartel: Acknowledged with no action taken for now.

Spearbit: Acknowledged.

5.6 Informational

5.6.1 Addressing variable naming and comments can improve readability

Severity: Informational

Context: InstitutionalPirexEthConfigurationLogic.sol#L127, InstitutionalPirexEthConfigurationLogic.sol#L142, InstitutionalPirexEthConfigurationLogic.sol#L203, InstitutionalPirexEth.sol#L157, InstitutionalPirexEthDeposit-Logic.sol#L483

Description: There are few places in the codebase where variables can be better named and code comments have typographical errors, which affect readability. Few comments in the institutional contracts incorrectly copy-paste and reference retail variables or flows. Some examples are illustrated below:

1. InstitutionalPirexEthConfigurationLogic.sol#L127: `$contract` may be changed to `_contract` to be consistent with the naming convention.
2. InstitutionalPirexEthConfigurationLogic.sol#L142: `oldVault` may be named as `oldAutoPxEthGateway`.
3. InstitutionalPirexEthConfigurationLogic.sol#L203: `Errors.UnrecorgnisedContract()` should be `Errors.UnrecognisedContract()`.
4. InstitutionalPirexEth.sol#L157: This should say "*Emergency withdrawal for all ERC20 tokens and ETH (except pxETH)*".
5. InstitutionalPirexEthDepositLogic.sol#L483: This should say "*The calculated amount of pxEth*".

Recommendation: Consider addressing variable naming and code/NatSpec comments to improve readability.

Redacted Cartel: Addressed as part of the consolidated changes in PR 57.

Spearbit: Reviewed that PR 57 implements the typographical and readability suggestions.

5.6.2 The token transfer/conversion logic in `instantRedeemWithIPxEth()` can be refactored with a call to `getApxEth()`

Severity: Informational

Context: InstitutionalPirexEthWithdrawLogic.sol#L176-L191, InstitutionalPirexEthWithdrawLogic.sol#L476-L488, InstitutionalPirexEthWithdrawLogic.sol#L320-L324)

Description: The token transfer/conversion logic in `instantRedeemWithIPxEth()` is exactly the same as that implemented in `getApxEth()` and therefore can be replaced with a call to it. This is similar to what's implemented in `executeInitiateRedemption()`.

Recommendation: Consider replacing the token transfer/conversion logic in `instantRedeemWithIPxEth()` with a call to `getApxEth()`. Consider renaming `getApxEth()` to something like `redeemPxEth()` because it transfers in `iPxEth`, unwraps it to `apxEth` and redeems that to `pxEth`.

Redacted Cartel: Addressed as part of the consolidated changes in PR 57.

Spearbit: Reviewed that PR 57, as recommended, resolves the issue by renaming `getApxEth()` to `redeemPxEth()` and calling that to avoid duplicated code.