Arkis smart contracts
Security Review

Solo review by:
M4rio.eth, Lead Security Researcher

May 8, 2025

Contents

1 Introduction

1.1 About Cantina

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

1.2 Disclaimer

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

1.3 Risk assessment

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

1.3.1 Severity Classification

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

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

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

2 Security Review Summary

Arkis is a digital asset prime brokerage protocol for institutional borrowers and lenders to interact in a zero-trust environment.

From Mar 30th to Apr 6th the security researchers conducted a review of arkis-smart-contracts-raw.a0 on commit hash ecc68075. A total of **20** issues were identified:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	5	3	2
Low Risk	2	1	1
Gas Optimizations	4	4	0
Informational	8	4	4
Total	**20**	**13**	**7**

3 Findings

3.1 High Risk

3.1.1 Missing checkpoint on APY change will create unfair interest and rewards

Severity: High Risk

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

Description: The agreement uses the APY to calculate the interest accrued for both borrowers and lenders:

```
function _pendingTPS(StorageStaking storage $) private view returns (uint256) {
    if ($.totalDeposited == 0) return $.tps;
    /* solhint-disable-next-line not-rely-on-time */
    uint256 secondsPassed = block.timestamp - $.lastCheckpoint;

    // We need double rounding here to avoid situations where borrowers repay less
    // than what is due to lenders. With division in formulas, a surplus is unavoidable,
    // but it must favor the pool and not the lender, because in the latter case
    // we simply cannot pay off.
    uint256 r1 = ($.apy * secondsPassed * $.utilization) / (APY_DENOMINATOR * 365 days);
    uint256 r2 = (r1 * TPS_DENOMINATOR) / $.totalDeposited;

    return $.tps + r2;
}

function _pendingInterest(
    StorageStaking storage $,
    address borrower
) private view returns (uint256) {
    Debt storage debt = $.debts[borrower];
    /* solhint-disable-next-line not-rely-on-time */
    uint256 secondsPassed = block.timestamp - debt.timestamp;
    return (debt.borrowed * secondsPassed * $.apy) / (APY_DENOMINATOR * 365 days);
}
```

The problem arises when the APY is changed:

```
function _setApy(uint32 apy) internal {
    _setApy(_storageStaking(), apy);
}
// ...
function _setApy(StorageStaking storage $, uint32 apy) private {
    $.apy = apy;
}
```

No checkpoint is made before applying the new APY, it affects all interest retroactively from the last checkpoint. This results in unfair interest accumulation:

- For lenders: Their rewards calculation via TPS will be retroactively changed.
- For borrowers: The interest on their loans will be calculated using the new rate for the entire loan period.

Recommendation: Call the `_checkpoint` function before setting the new APY. For individual borrowers, maintain a history of APY changes along with their timestamps and calculate accrued interest for each change interval accordingly.

Arkis: Fixed in commit 5abd2a4c.

Cantina Managed: Fix verified.

3.2 Medium Risk

3.2.1 The `info` function in the `AgreementStaking` will run out of gas

Severity: Medium Risk

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

Description: The `info` function in `AgreementStaking` iterates through all lenders and borrowers, returning them all at once:

```
function info()
    external
    view
    override
    returns (Metadata memory metadata, Whitelist memory whitelist)
{
    Storage storage $ = _storage();
    (metadata.leverage, metadata.totalDepositThreshold, metadata.apy) = _info();
    metadata.collaterals = $.collaterals.values();
    metadata.lenders = _getRoleMembers(LENDER_ROLE);
    metadata.borrowers = _getRoleMembers(BORROWER_ROLE);
    IWhitelistingControllerAgreement wc = IWhitelistingControllerAgreement(compliance);
    whitelist = Whitelist(wc.getTokens(address(this)), wc.getOperators(address(this)));
    return (metadata, whitelist);
}
```

The problem is that as the list of borrowers and lenders grows, the function may eventually exceed the gas limit and revert. This issue becomes serious as its used in the `ThresholdsVerifier.verifyThresholds` which will result in DoSing the verification.

Recommendation: Consider splitting this into another function that gets only the metadata without lenders/borrowers so that could be used in the `ThresholdsVerifier` without DoSing and a function that can be called offchain which could return everything.

Arkis: We acknowledge this issue and have decided to maintain the current implementation for now. We will consider addressing this matter in future releases.

Cantina Managed: Acknowledged.

3.2.2 Missing slippage check on curve's remove liquidity

Severity: Medium Risk

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

Description: The Compiler can remove liquidity from a Curve pool by using `CurveFiEvaluator.evaluate` for the `DecreasePositionRequest`. The issue is that this constructs a `remove_liquidity` call with all `minAmounts` set to zero:

```
bytes4 private constant REMOVE_LIQUIDITY_2 =
    bytes4(keccak256("remove_liquidity(uint256,uint256[2])"));
bytes4 private constant REMOVE_LIQUIDITY_3 =
    bytes4(keccak256("remove_liquidity(uint256,uint256[3])"));
bytes4 private constant REMOVE_LIQUIDITY_4 =
    bytes4(keccak256("remove_liquidity(uint256,uint256[4])"));
bytes4 private constant REMOVE_LIQUIDITY_ZAPPER =
    bytes4(keccak256("remove_liquidity(address,uint256,uint256[4])"));

function _constructNCoinsDecreasePosition(
    uint256 nCoins,
    uint256 burnAmount
) private pure returns (bytes memory) {
    if (nCoins == 2) {
        return abi.encodeWithSelector(REMOVE_LIQUIDITY_2, burnAmount, [0, 0]); // <== example
    } else if (nCoins == 3) {
        return abi.encodeWithSelector(REMOVE_LIQUIDITY_3, burnAmount, [0, 0, 0]);
    } else {
        return abi.encodeWithSelector(REMOVE_LIQUIDITY_4, burnAmount, [0, 0, 0, 0]);
    }
}
```

This opens the possibility for the remove command to be sandwiched, where the expected token ratios do not match what would be received under normal conditions. The risk is heightened if a specific token in the pool enters a depletion scenario, where MEV bots compete to drain it. Since the current implementation accepts any ratio, this can result in significant losses.

Recommendation: Since most parameters are already constructed on the backend, consider including `minAmounts` as an input parameter. This would allow explicit control over slippage tolerance, enabling the setting of either high or low slippage thresholds.

Arkis: Fixed in commit 5abd2a4c.

Cantina Managed: Fix verified.

3.2.3 Missing `executor`/`clipperExchange` check could result in malicious command

Severity: Medium Risk

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

Description: The OneInchV6ValidatorGeneric and OneInchV6ValidatorClipper.sol are used to encode commands for 1Inch Clipper and 1Inch general router. Both of these use an executor variable which will execute the swap.

As we can see both are missing a check to ensure that that is a trustable executor which means that the command can be forged to use a malicious executor which could potentially siphon funds.

```
function swap(
    address executor,
    IGenericRouter.SwapDescription calldata desc,
    bytes calldata permit,
    bytes calldata data
) external payable override refundEth(desc.amount) returns (Command[] memory cmds_) {

    // ...

function clipperSwapTo(
    address clipperExchange, // <== example of exchange param
    address payable recipient,
    Address srcToken,
    address dstToken,
    uint256 inputAmount,
    uint256 outputAmount,
    uint256 goodUntil,
    bytes32 r,
    bytes32 vs
) external payable override refundEth(inputAmount) returns (Command[] memory cmds) {
```

Recommendation: Consider always checking these variables against known addresses, you could create a map which will whitelist executors that can be used.

Arkis: Fixed in commit a4efe829.

Cantina Managed: After a discussion with the 1inch team, it was confirmed that the executor for generic swaps does not need to be whitelisted, as the aggregated router verifies the `minAmount` returned.

However, for Clipper Exchange, the router performs no verification at all, so whitelisting is required.

Following further investigation, the team has decided to remove all swap functions except the generic `swap()` due to concerns that the aggregator router does not verify the `minAmount` for every swap function. This could lead to a complete drain of the swap amount if an attacker injects a malicious pool into the swap path and the swap function does not verify the returned `minAmount`.

This change was implemented in commit a4efe829.

3.2.4 The Risk Operator can improperly trigger liquidation

Severity: Medium Risk

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

Description: We can observe that both the liquidation and close account operations conceptually perform the same action on the account:

```
function close(
    Script[] calldata closeStrategy
)
    external
    override
    onlyDispatcher
    onlyState(STATE_OPENED | STATE_SUSPENDED)
    returns (bool success)
{
    return _runScript(closeStrategy);
}

function liquidate(
    Script[] calldata liquidationStrategy
)
    external
    override
    onlyLiquidator
    onlyState(STATE_OPENED | STATE_SUSPENDED)
    returns (bool success)
{
    return _runScript(liquidationStrategy);
}
```

The difference is that they can be called by different operators: the risk operator or the liquidation operator. The issue is that there is nothing preventing the risk operator from executing liquidations on any account. This is a critical operation within Arkis and should be treated as high-risk. This problem is further amplified because the liquidation process is not checked on-chain at all. While we trust the liquidator to act in good faith—meaning they perform the necessary checks before liquidating—the same cannot be guaranteed for the risk operator.

Recommendation: The optimal solution would be to verify on-chain whether an account is eligible for liquidation. This would prevent a risk operator from liquidating a healthy account.

Arkis: We have institutional clients with signed agreements and completed KYC. The risk factor calculation is documented, all parameters are public, a snapshot of each calculation is sent to both the clients and us, and it's also stored in an archive with all parameters for conflict resolution.

Cantina Managed: Acknowledged.

3.2.5 Lenders will lose everything if they get removed

Severity: Medium Risk

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

Description: In the Agreement contract, role-based access control (RBAC) is enforced on all functions, including `withdraw` and `claim`:

```
function withdraw(uint128 amount) external override onlyRole(LENDER_ROLE) {
    // ...
}

function claim() external override onlyRole(LENDER_ROLE) {
    // ...
}
```

The issue arises because the Agreement Factory has the ability to remove any lender's role without restriction. If a lender is removed from the `LENDER_ROLE`, they lose access to both their deposited funds and any rewards, since they can no longer call `withdraw` or `claim`.

Recommendation: Consider decoupling access control from fund ownership. One approach is to allowing them to call `withdraw` and `claim` even if they are removed. If the need to actually freeze the funds we can introduce a blacklist functionality which can then blacklist a lender for accessing his funds or rewards.

Arkis: Fixed in commit 5abd2a4c.

Cantina Managed: Fix verified.

3.3 Low Risk

3.3.1 The `Pauser` can run out of gas DoSing the pause functionality

Severity: Low Risk

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

Description: The `Pauser` contract serves as a central entity that can pause multiple contracts simultaneously:

```
function pauseAll() external override onlyRole(ARKIS_ALERT_OPERATOR_ROLE) {
    Storage storage $ = _storage();
    bool anyPaused = false;
    uint256 length = $.pausables.length();
    for (uint256 i = 0; i < length; i++) {
        address pausable = $.pausables.at(i);
        try IPausable(pausable).pause() {
            anyPaused = true;
        } catch (bytes memory reason) {
            emit FailedToPause(pausable, reason);
        }
    }
    if (!anyPaused) revert AlreadyUpToDate();
}
```

The problem is that it iterates through all the pausable contracts at once, pausing them one by one. If the list grows large enough, this could lead to a denial of service (DoS), impacting both pause and unpause functionality, as both iterate through all components.

Recommendation: Consider adding a separate function for pause/unpause that processes components in a paginated manner.

Arkis: We acknowledge your concern regarding the gas usage of `PauseAll()` and `UnpauseAll()`. Currently, with 6 modules, `PauseAll()` consumes 212,356 gas, and `UnpauseAll()` uses 117,977 gas. Given the current block gas limit of 36 million, we can theoretically support approximately 169 modules for `PauseAll()` (36,000,000 / 212,356 $\approx$ 169) and 305 modules for `UnpauseAll()` (36,000,000 / 117,977 $\approx$ 305) before reaching the limit. Since the `Pauser` contract is upgradeable, we can modify and redeploy it if gas issues arise in the near future.

Cantina Managed: Acknowledged.

3.3.2 The `PendleValidatorMisc` will revert if ETH is needed

Severity: Low Risk

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

Description: In PendleValidatorMisc.sol#L160, the `callAndReflect` function aggregates three calls—two optional internal calls and one reflect call—into a single `multicall` invocation:

```
function callAndReflect(
    address payable,
    bytes calldata selfCall1,
    bytes calldata selfCall2,
    bytes calldata reflectCall
)
    external
    payable
    override
    onlyValidReflectCallData(reflectCall)
    refundEth(msg.value)
    returns (Command[] memory cmds)
{
    uint256 length = selfCall2.length > 0 ? 3 : 2;
    IPActionMiscV3.Call3[] memory calls = new IPActionMiscV3.Call3[](length);
    calls[0].callData = selfCall1;
    if (selfCall2.length > 0) calls[1].callData = selfCall2;
    calls[length - 1].callData = reflectCall;

    bytes memory payload = abi.encodeCall(IPendleValidatorMisc.multicall, (calls));
    bytes memory result = address(this).safeDelegateCall(
        payload.appendWord(getOperator(Support.NotRequired))
    );
    cmds = abi.decode(result, (Command[]));
}

function multicall(
    IPActionMiscV3.Call3[] calldata calls
) external payable override refundEth(msg.value) returns (Command[] memory cmds) {
```

The issue here is that both `callAndReflect` and `multicall` use the `refundEth(msg.value)` modifier. Since `callAndReflect` performs a `delegatecall` to `multicall`, they share the same execution context and thus the same `msg.value`. As a result, both will attempt to refund the same ETH amount, causing the second refund attempt to fail and revert the entire transaction. This breaks the expected behavior, especially when ETH is legitimately required by the calls being aggregated.

Recommendation: To avoid multiple refund attempts on the same `msg.value`, remove the `refundEth` modifier from the `multicall` function and retain it only in `callAndReflect`. This ensures that ETH is refunded exactly once after the entire composite call finishes execution.

Arkis: Fixed in commit 5abd2a4c.

Cantina Managed: Fix verified.

3.4 Gas Optimization

3.4.1 Gas optimizations in `Agreement`

Severity: Gas Optimization

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

Description: We can find some gas optimizations in the agreement:

- `APY_DENOMINATOR * 365 days` could be made a constant and avoid doing the calculations all the time.
- AgreementStaking.sol#L99 the `_claim` is already calling `_checkpoint`.

Recommendation: Consider making these gas optimizations.

Arkis: Fixed in commit 5abd2a4c.

Cantina Managed: Fix verified.

3.4.2 Gas Optimization in the `CurveFiExchange`

Severity: Gas Optimization

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

Description: The CurveFiExchange we can see a gas optimization: Line 40 and 41 do the same thing:

```
CurveFiPool storage basePool = pool.basePoolId != 0 ? getPool(pool.basePoolId) : pool;
if (pool.basePoolId != 0) basePool = getPool(pool.basePoolId);
```

Recommendation: Consider the second line.

Arkis: Fixed in commit 5abd2a4c.

Cantina Managed: Fix verified.

3.4.3 Gas Optimization in `Pauser`

Severity: Gas Optimization

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

Description: The Pauser.sol#L53 checks if it can pause or not a component, if not then it sets a variable named `hasRole` to false and then it reverts. This can be optimized to revert directly if you can not pause a component.

```
try IAccessControl(pausable).hasRole(PAUSER_ROLE, address(this)) returns (
    bool result
) {
    hasRole = result;
} catch {
    hasRole = false;
}
if (!hasRole) revert MissingPauserRole(pausable);
```

Recommendation: Consider reverting automatically in the catch, discarding the `hasRole`.

Arkis: Fixed in commit 5abd2a4c.

Cantina Managed: Fix verified.

3.4.4 Gas Optimization in `ThresholdsVerifier`

Severity: Gas Optimization

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

Description: The ThresholdsVerifier.sol#L27 does a check on all the collaterals to see if they are accepted collaterals in an agreement:

```
for (uint256 x; x < _collaterals.length; x++) {
    Asset calldata c = _collaterals[x];
    bool isValidCollateral = false;
    for (uint256 y = 0; y < metadata.collaterals.length; y++) {
        if (c.token == metadata.collaterals[y]) isValidCollateral = true;
    }
    if (!isValidCollateral) revert WrongCollateral(agreement, c.token);
    if (c.amount == 0) revert AmountMustNotBeZero(c.token);
}
```

We can see an optimization in the second for loop, if we find a collateral we can set it to `true` and then break to avoid unnecessary loops.

Recommendation: Consider implementing this gas opmitization.

Arkis: Fixed in commit 5abd2a4c.

Cantina Managed: Fix verified.

3.5 Informational

3.5.1 Standardize the usage of `encodeCall` across the codebase

Severity: Informational

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

Description: The `Arkis` infrastructure relies on instructions encoded into `Command` which are then delegate called in various components. We can see that on many parts where we construct the command we use the `encodeWithSelector`:

```
abi.encodeWithSelector(msg.sig, _pool, _burn_amount, i, _min_amount, msg.sender);
```

This won't perform type checking on the parameters which can result in wrong commands being created. Also, we can see that sometimes we have the function signature directly used which is as well a bad practice as it can lead to mistakes:

```
bytes memory data = _receiver == msg.sender
    ? msg.data
    : abi.encodeWithSelector(
        0xc872a3c5, // <==
        _route,
        _swap_params,
        _amount,
        _expected,
        _pools,
        msg.sender
    );
```

Recommendation: While I did not see any type violation across the codebase, it's a good practice to use `encodeCall` instead which does perform the type checking.

Arkis: It's is not always possible, for example in case of overloaded function encodeCall is not supported. in validators I do think use of encodeWithSelector along with msg.sig is safer because this way we make sure signature in our compliance is aligned with intended method, otherwise we might end up calling unwanted method with same set of parameters (there are some).

Cantina Managed: Acknowledged.

3.5.2 Various documentation and minor issues

Severity: Informational

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

Description: We've identified various minor bugs and documentation issues that were aggregated into one issue:

- Across the codebase we can see many instances where we have typos: IWithdrawer.sol#L16.
- NatSpec and documentation are missing in multiple instances; this should be improved, as the architecture is quite complex and difficult to grasp without them.

Recommendation: Consider fixing these issues.

Arkis: Fixed in commit 5abd2a4c.

Cantina Managed: Fix verified.

3.5.3 Improve the error returned on `enforceCanBorrow`

Severity: Informational

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

Description: The `enforceCanBorrow` is used to see if a user can borrow, this includes as well if the agreement is paused. In case that we have the agreement paused, the revert error would be the one from the `Pausable` contract not the `UserCannotBorrow`.

Recommendation: Consider checking the `paused` in the function and return the `UserCannotBorrow` error to be consistent.

Arkis: We confirm that when the `Agreement` contract is paused, any attempt to borrow will revert with an `EnforcedPause` error, even if the address attempting the transaction lacks borrowing permissions in the pool. We believe there's no need to change this behavior, as it aligns with the intended pause functionality.

Cantina Managed: Acknowledged.

3.5.4 Missing `tokenIn` sanity check

Severity: Informational

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

Description: In the PendleEvaluator.sol?lines=38,38 after we select which function we want to execute, the result will contain a `tokenIn` variable. This variable should be sanity checked against the path which should always match.

```
function evaluate(
    address,
    ExchangeRequest calldata request
) external view override returns (Command[] memory cmds) {
    SwitchResult memory result = switchBySelector(
        request.extraData,
        request.amountIn,
        request.minAmountOut,
        request.recipient
    );

    cmds = Command({target: result.target, value: 0, payload: result.data}).populateWithApprove(
        result.tokenIn,
        request.amountIn
    );
}
```

Recommendation: Consider adding the sanity check against the `tokenIn` to avoid any unintended scenarios where a wrong path (with a wrong tokenIn) will be encoded in the command.

Arkis: Fixed in commit 5abd2a4c.

Cantina Managed: Fix verified.

3.5.5 Owner and ACL usage can create confusion

Severity: Informational

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

Description: In BeaconUpgradeable.sol#L36, both OpenZeppelin's `AccessControl` and `Ownable` are used in the same contract. While both patterns serve to manage permissions and access control, using them together can introduce confusion and redundancy.

`Ownable` provides a simple ownership model where a single account has exclusive control, while `AccessControl` offers a more flexible and role-based approach to permissioning. Mixing the two may lead to unclear authority boundaries.

Recommendation: Since `AccessControl` is already integrated, it would be more consistent and maintainable to eliminate `Ownable` altogether. Instead, define a dedicated role—such as `OWNER_ROLE`—and assign it to the intended owner. Alternatively, you can use the default admin role provided by `AccessControl` as the owner.

Arkis: We acknowledge your recommendation but will retain the current setup for the following reasons: `Ownable` contracts are used for deploy operations, compliance master set edits (primarily during contract upgrades), margin account deployment, and restricting margin account operations to the owner only. ACL is used for all other restricted methods. This balance suits our operational needs, but we'll keep your suggestion in mind for future reviews.

Cantina Managed: Acknowledged.

3.5.6 Error Optimization in `CurveFiValidatorSwapRouter4`

Severity: Informational

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

Description: The CurveFiValidatorSwapRouter4.sol#L84 will underflow and panic if the first pool is `address(0)`.

```
if (_swap_params[i - 1][2] >= 7 && _swap_params[i - 1][2] <= 11)
    validateOperator(_route[(i - 1) * 2 + 1], Support.Required);
_route[i * 2].enforceTokenSupported();
```

Recommendation: As this will revert with a Panic, consider reverting explicitly to debug easier.

Arkis: Fixed in commit 5abd2a4c.

Cantina Managed: Fix verified.

3.5.7 The `refundEth` modifier returns ETH before execution and it requires the sender to receive Ether

Severity: Informational

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

Description: The `refundEth` modifier returns ETH to the sender **before** executing the function body. This is done to verify that the `msg.value` matches the expected amount before proceeding with execution.

```
modifier refundEth(uint256 expectedValue) {
    if (msg.value > 0) {
        if (msg.value != expectedValue) revert MismatchedEthValue(msg.value, expectedValue);
        payable(msg.sender).sendValue(msg.value);
    }
    _;
}
```

However, refunding ETH before the main function logic introduces a reentrancy risk. Since the refund is sent to `msg.sender`, and this action can trigger the `receive()` or `fallback()` function of the sender, it opens the door for unintended behavior or attacks if the sender is a smart contract.

Additionally, the behavior should be clearly documented. If the recipient does not accept ETH (e.g., lacks a `receive` function or reverts on `receive()`), the transaction will fail entirely—even though the ETH was only temporarily sent. Also, the typical expectation is that ETH is sent as part of the encoded command execution, not refunded upfront.

Recommendation: Move the refund logic to the end of the modifier or, preferably, after the function execution. This ensures the core logic executes first and reduces the potential for reentrancy vulnerabilities. Additionally, clearly document the expected behavior and edge cases for refund handling.

Arkis: Fixed in commit 5abd2a4c.

Cantina Managed: Fix verified.

3.5.8 Missing init calls of all the inherited contracts

Severity: Informational

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

Description: In upgradeable contract patterns, it's a widely accepted best practice to explicitly call all inherited `__init__` (or `__*_init`) functions, even if they don't currently include any logic. This ensures forward compatibility and reduces the risk of missing critical initialization logic if those parent contracts are later extended.

In the `Agreement` contract, we can see that this pattern is not fully followed:

```
function __Staking_init(
    address token,
    uint256 threshold,
    uint32 apy
) internal onlyInitializing {
    __Pausable_init();
    StorageStaking storage $ = _storageStaking();
    _setApy($, apy);
    $.token = IERC20(token);
    if (threshold > 0) $.totalDepositThreshold = threshold;
}
```

The `__AccessControlEnumerable_init()` function is missing, even though the contract inherits from `AccessControlEnumerableUpgradeable`. This omission may not cause immediate issues but could lead to unexpected behavior or incomplete initialization in the future.

Recommendation: Review all upgradeable contracts and ensure that all inherited initializer functions are explicitly called within the initializer of the derived contract.

Arkis: We acknowledge your suggestion. However, `AccessControlEnumerableUpgradeable` (like `AccessControlUpgradeable`) doesn't have an `initialize()` method defined, so calling it isn't applicable here. We'll keep this pattern in mind for cases where inherited contracts do introduce initialization logic in the future.

Cantina Managed: Acknowledged.