Ern Smart Contract

August 2025 to November 2025

Made with ❤ by the following Creed authors: Dominik Muhs

Table of Contents

Executive Summary 4

Scope and Objectives 5

Audit Artifacts 6

Findings 7

- [Major] Dangerous Arithmetic Without Decimal Normalization
- [Medium] aToken Donations Can Become Stranded
- [Medium] Unclaimed Aave Rewards
- [Minor] Unused Code
- [Minor] Potential Harvest Griefing
- [Minor] Potential Redundant Reward Token Swap
- [Minor] Hardcoded Underlying Decimals
- [Minor] Paused Pools Break Harvest Check Assumptions
- [Minor] Missing Aave `supplyCap` Validation
- [Minor] Hardcoded Aave Pool Address

Disclaimer 19

Executive Summary

This report presents the results of our engagement with Sumcap to review their Ern smart contract.

The review was conducted over one week, from August 27, 2025, to September 3, 2025, by Dominik Muhs. A total of five person-days were spent.

Overall, we identified 10 findings: 1 major, 2 medium, and 7 low-severity items. The most significant findings concern internal arithmetic and the integration with Aave. The low-severity items focus on defense-in-depth, maintainability, and laying the groundwork for future iterations.

At the developer team's request, the report also includes a design considerations section with forward-looking recommendations and notes on potential risk areas. Given the time-boxed scope and the focus on the code base, this section should be read as good-faith guidance rather than a comprehensive risk assessment.

Furthermore, a mitigations review was conducted over three days, from November 14, 2025, to November 17, 2025, by Dominik Muhs. A total of three person-days were spent. All findings were either acknowledged with a statement or validated as fixed by the reviewer.

Scope and Objectives

Our review focused on the commit hash `095ac6f3c77b932707746e68093fd56ea06019c8`.

Together with the Ern developer team, we identified the following priorities for our review:

- Validate interactions with external protocols are free of unintended side effects.
- Verify accounting correctness across user interactions with the `Ern` contract.
- Review access control and operational safeguards and potential risks stemming from their absence.
- Ensure that the system is implemented consistently with the intended functionality, and without unintended edge cases.
- Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Security Field Guide, and the ones outlined in the EEA EthTrust Security Levels Specification.

On November 7, 2025, the report was edited to reflect changes in the code base, using the following commit hash: `914dfcc22944edb0b44a7281f0e74226756cca39`. Findings that were addressed are marked accordingly.

During the mitigations review, the following revisions were factored in:

- `main` branch at `8f4146245d5eabf771416ce139cc48546b249126`
- PR #99 at `63a47d2ceb14ef133a0cc7262cd02092d386088d`
- PR #100 at `8dfb41b8d8b62e9320b0e258caf2a8f5792ab16e`

Audit Artifacts

Design Considerations

During the kickoff call, the client asked the assessment team to provide additional considerations around future Ern iterations. This mainly concerned the wrapping of DeFi protocols other than Aave, and the use of non-stablecoins. In the following sections, we will provide some high-level considerations that do not constitute security issues per se and should be regarded as opinionated thoughts.

When integrating with other tokens, we recommend a solution where token contracts are vetted on a case-by-case basis before being made available to users. Especially custom tokens, as well as non-ERC20 implementations that provide the following features, should be scrutinized:

- Pauseability
- Blacklist
- Transfer fees
- Hooks and callbacks

When supporting additional token implementations, the fact that user rewards are also settled during deposits and withdrawals may become an issue, as the token may prevent users from updating their positions. Thus, rewards-related logic should be decoupled from position updates so deposits and withdrawals only adjust accounting, with token transfers performed via explicit claims.

Furthermore, deposits and withdrawals currently assume nominal token amounts reflect actual balance changes. With fee-on-transfer or otherwise non-standard ERC-20s, the `Ern` contract in its current implementation would mint or burn shares against incorrect deltas and create silent deficits. If these tokens have to be supported, we recommend basing accounting on observed balance changes (pre/post) rather than the requested amounts. Extra accounting logic will be required to align shares and assets.

Findings

Major Dangerous Arithmetic Without Decimal Normalization

Fixed

> This finding has been addressed in PR #73 at `ccca0e5b63c3c2812645f508a01707cb12c3d321`. The decimals are now initialized in the contract's constructor based on the underlying asset, and `minYieldAmount` is initialized in a normalized fashion alongside it. The underlying decimals are correctly reflected in the overridden `ERC20.decimals()` view.

The `Ern` contract is initialized with its shares as a standard ERC20 token with 18 decimals. The underlying pool token, as indicated by the minimum yield values, is supposed to have six decimals.

There are multiple occurrences in the smart contract where arithmetic operations are performed on the `Ern` shares and the Aave tokens that do not perform proper normalization. Consequently, this can result in accounting errors, which end up processing the wrong yield amounts and can result in a de facto loss of user funds.

ern-914dfcc2/contracts/src/Ern.sol

```
uint256 currentBalance = A_UNDERLYING.balanceOf(address(this));
uint256 totalSharesSupply = totalSupply();

// Calculate potential yield amount
uint256 yieldAmount = currentBalance > totalSharesSupply ? currentBalance - tota…
```

ern-914dfcc2/contracts/src/Ern.sol

```
uint256 currentBalance = A_UNDERLYING.balanceOf(address(this));
uint256 totalSharesSupply = totalSupply();

// Calculate yield (current aToken balance - total shares)
if (currentBalance <= totalSharesSupply) return; // No yield to harvest

uint256 yieldAmount = currentBalance - totalSharesSupply;
```

For example:

A user submits 100.1337 `USDC`. Consequently, `100133700` (100.1337 * 10**6) is supplied to the Aave pool. When the mint function is called, that same amount is minted:

ern-914dfcc2/contracts/src/Ern.sol

```
// Mint shares 1:1 with deposited amount
_mint(msg.sender, amount);
```

In the context of the `Ern` ERC20, this is interpreted with 18 decimals, however. So 100.1337 shares of `shUSDC` result in `100133700000000000000` (100.1337 * 10**18)

Recommendation

Whenever token amounts with varying decimals are used, the token amounts should be normalized before any arithmetic operations are performed. Alternatively, the ERC20 `decimals` view could also be overridden to dynamically mirror the underlying token's `decimals` view, which makes all yield calculation operations consistent again.

Medium aToken Donations Can Become Stranded

Acknowledged

> This finding has been acknowledged by the client. They provided the following statement: "This mechanism is not utilized by the specific Aave pools in use by Ern. Due to the extra smart contract complexity, lack of attack vector and the fact that there is no exploit scenario where the action benefits an attacker, this finding did not seem worth the upfront mitigation effort."

Anyone can donate funds to the relevant Aave pool on behalf of the `Ern` contract. This results in an `aToken` balance that is not accounted for by any shares if the contract has just been bootstrapped and no shares are present in the system yet. When an authorized harvester runs `harvest`, the contract will withdraw the full excess from Aave and swap to the reward token.

ern-914dfcc2/contracts/src/Ern.sol

```
AAVE_POOL.withdraw(address(UNDERLYING), yieldAmount, address(this));

// Swap yield to reward tokens
// NOTE: this assumes the tokens are not the same
uint256 rewardReceived = DEX.exactInputSingle(
    IDex.ExactInputSingleParams({
        tokenIn: address(UNDERLYING),
        tokenOut: address(REWARD_TOKEN),
        fee: 3000, // 0.3% fee tier
        recipient: address(this),
        deadline: block.timestamp + 300, // 5 minutes
        amountIn: yieldAmount,
        amountOutMinimum: minOut,
        sqrtPriceLimitX96: 0
    })
);
```

However, since no shares are present in the system, the distribution index will not be updated.

ern-914dfcc2/contracts/src/Ern.sol

```
// Update cumulative reward tokens per share
if (totalSharesSupply > 0) {
    cumulativeRewardPerShare += (userRewards * 1e18) / totalSharesSupply;
}
```

The reward tokens remain idle in the contract and are not attributable to future minters. Although there is no exploit scenario where this action benefits an attacker, the stranded rewards are user-unfriendly and introduce accounting ambiguity.

Recommendation

We recommend preventing harvest altogether when there are no user shares in the system. Alternatively, "pending rewards" can be accumulated off-index to be credited on the first non-zero supply harvest.

Medium Unclaimed Aave Rewards

Acknowledged

> This finding has been acknowledged by the client. They provided the following statement: "This mechanism is not utilized by the specific Aave pools in use by Ern. Due to the extra smart contract complexity this finding did not seem worth the upfront mitigation effort."

When users supply their funds to the `Ern` contract by calling `deposit`, the smart contract supplies them to an Aave V3 pool to generate yield. In regular `harvest` calls, the generated yield is withdrawn from the pool, converted into the specified reward token, and stored as user reward.

However, Aave provides an additional incentive system that allows for distributing additional rewards to users who supply certain assets to the protocol. The `Ern` contract does not provide facilities to claim these funds, leaving them locked indefinitely.

Recommendation

We recommend incorporating a check and subsequent call to the respective `RewardsController` deployment, as well as a setter and validation around changing the controller address. More information on the incentive program and its periphery contracts can be found in the official documentation: https://aave.com/docs/developers/smart-contracts/incentives

Minor Unused Code

Fixed

> This finding was addressed in PR #74 at `f2b448a4ffcefe25a2d5387bbde89b672a090783`. All dead code has been removed.

There are several occurrences of unused private functions in the code base. These can be safely removed. We recommend giving the project a general scan to look for dead code.

Examples

```
function _requireHarvestCooldown() private view {
    if (block.timestamp < lastHarvest + harvestCooldown) revert HarvestCooldownNotMet();
}
```

```
function _requireSufficientAllowance(uint256 allowed, uint256 amount) private pure {
    if (allowed < amount) revert InsufficientAllowance();
}
```

Minor Potential Harvest Griefing

Acknowledged

> This finding has been acknowledged by the client in the context of PR #76. They provided the following statement: "The business logic already contains a set of conditions for a harvest to happen, as suggested. Furthermore, the "minYieldAmount" value will be tweaked to ensure the system works as intended and is profitable for the harvester."

The `Ern` contract treats any excess aToken balance as harvestable yield. Any third party can donate to the pool reserve on behalf of the contract, raising the `aToken` balance without changing the share supply.

Depending on the internal business logic of the harvester bot, an adversary can repeatedly donate just over `minYieldAmount`, keeping `canHarvest` true and pressuring the bot to execute many small harvests.

ern-914dfcc2/contracts/src/Ern.sol

```
if (!yieldSufficient && !timePassed) {
    return (false, yieldAmount);
} else {
    return (true, yieldAmount);
}
```

Depending on the chain the contract is deployed on, this will waste gas, compound DEX fees, and increase slippage/MEV across fragmented swaps. Consequently, the bot may operate at a loss when per-call gas exceeds fee revenue.

Recommendation

We recommend enforcing a minimum interval between harvests and fine-tuning the `minYieldAmount` value so each harvest amortizes gas and swap costs.

Minor Potential Redundant Reward Token Swap

Acknowledged

> This finding has been acknowledged by the client. They provided the following statement: "Since the underlying and reward are always different assets and such a scenario would not be reached, the finding did not seem worth the upfront mitigation effort."

Currently, the system attempts a swap even when the underlying and reward tokens are identical. This adds unnecessary fees and MEV exposure for no benefit.

ern-914dfcc2/contracts/src/Ern.sol

```
uint256 yieldAmount = currentBalance - totalSharesSupply;

// Withdraw yield from Aave
AAVE_POOL.withdraw(address(UNDERLYING), yieldAmount, address(this));

// Swap yield to reward tokens
// NOTE: this assumes the tokens are not the same
uint256 rewardReceived = DEX.exactInputSingle(
    IDex.ExactInputSingleParams({
        tokenIn: address(UNDERLYING),
        tokenOut: address(REWARD_TOKEN),
        fee: 3000, // 0.3% fee tier
        recipient: address(this),
        deadline: block.timestamp + 300, // 5 minutes
        amountIn: yieldAmount,
        amountOutMinimum: minOut,
        sqrtPriceLimitX96: 0
    })
);
```

Recommendation

We recommend bypassing the swap when the tokens match to remove risk and save gas with a single conditional.

Minor Hardcoded Underlying Decimals

Fixed

> This finding has been addressed in PR #100 at `8dfb41b8d8b62e9320b0e258caf2a8f5792ab16e`. `minYieldAmount` has already been normalized by a prior fix. Furthermore, the contract's `setMinYieldAmount` function now correctly uses the `_decimals` contract variable to normalize the value given by the `newMinYieldAmount` parameter.

The conditions to allow for harvesting accrued yield include a check for a minimum yield amount. The initial value and boundaries for `minYieldAmount` are hardcoded to six decimals, however. This limits the stablecoins that are supported. An overview of stablecoins and their decimals can be found e.g. on Dune: https://dune.com/KARTOD/stablecoins-overview

Examples

ern-914dfcc2/contracts/src/Ern.sol

```
uint256 public minYieldAmount = 10e6; // Minimum yield amount (10 USDC with 6 de…
```

ern-914dfcc2/contracts/src/Ern.sol

```
function setMinYieldAmount(uint256 newMinYieldAmount) external onlyOwner {
    // Reasonable bounds: minimum 1 USDC (1e6), maximum 100,000 USDC (100000e6)
    if (newMinYieldAmount < 1e6) revert MinYieldAmountTooLow();
    if (newMinYieldAmount > 100000e6) revert MinYieldAmountTooHigh();
```

Recommendation

We recommend either making the token decimals configurable or explicitly limiting the kinds of underlying tokens that are supported by checking their decimals or implementing an allowlist approach.

Minor Paused Pools Break Harvest Check Assumptions

Acknowledged

> This finding has been acknowledged by the client. They provided the following statement: "This check of pool status is already handled off-chain by the bot. If a bot fails to simulate a harvest transaction the team is promptly notified. As such, the finding did not seem worth the upfront mitigation effort."

The `canHarvest` view and its state-changing equivalent `_performHarvest` use the `balanceOf` function to fetch the maximum withdrawable amount. However, the `balanceOf` function of `A_UNDERLYING` does not accurately reflect the withdrawable amount. When the protocol is paused or inactive, withdrawals are impossible; however, the liquidity calculation does not account for this state.

Examples

ern-914dfcc2/contracts/src/Ern.sol

```
uint256 currentBalance = A_UNDERLYING.balanceOf(address(this));
```

ern-914dfcc2/contracts/src/Ern.sol

```
uint256 currentBalance = A_UNDERLYING.balanceOf(address(this));
```

Recommendation

We recommend validating that the pool is in an active state before continuing with any reward calculations based on `A_UNDERLYING`. The public `canHarvest` view should return `false` if the pool is inactive to inform off-chain infrastructure.

Minor Missing Aave `supplyCap` Validation

Acknowledged

> This finding has been acknowledged by the client. They provided the following statement: "This validation on capped pools can be handled off-chain. Both the bot and the frontend are capable of simulating a deposit, as well as tracking supply caps and other parameters directly from Aave's contracts and/or GraphQL API, promptly notifying the team. As such, the finding did not seem worth the upfront mitigation effort."

An Aave pool can come with a capped supply. Once the supply cap is reached, no more liquidity for the given reserve asset can be supplied to the pool. Currently, this cap is unchecked, which means that user deposits that seem valid to the end user can unexpectedly revert on the Aave side.

It is worth noting that this issue only arises on capped pools. Pools with a `supplyCap` of zero are not affected. The validation check is contained in the protocol's `ValidationLogic.validateSupply` library function:

```
    uint256 supplyCap = reserveCache.reserveConfiguration.getSupplyCap();
    require(
      supplyCap == 0 ||
        (
          (IAToken(reserveCache.aTokenAddress).scaledTotalSupply() +
            scaledAmount +
            uint256(reserve.accruedToTreasury)).getATokenBalance(...)
        ) <=
        supplyCap * (10 ** reserveCache.reserveConfiguration.getDecimals()),
      Errors.SupplyCapExceeded()
    );
```

Recommendation

Ensure that the supply cap is honored during a deposit to avoid unexpected reverts. Additionally, a view function and additional changes might be required to inform off-chain infrastructure of the `Ern` instance's underlying pool limits and available capacities.

Minor Hardcoded Aave Pool Address

Fixed

> This finding has been addressed in PR #99 at `63a47d2ceb14ef133a0cc7262cd02092d386088d`. The direct usages of `AAVE_POOL` and `A_UNDERLYING` have been removed and replaced with calls to the Aave address provider `AAVE_ADDRESSES`. Calls have been updated to use `getAavePool` and `getAaveUnderlying` respectively. The scenario where the pool address does in fact change is covered by the admin-only `ensureApprovals` function.

The `AAVE_POOL` contract state variable is marked as `immutable`:

ern-914dfcc2/contracts/src/Ern.sol

```
17 IAavePool public immutable AAVE_POOL;
```

This poses a future risk as pool addresses are typically registered in the `PoolAddressesProvider` protocol contract, which allows for pool upgrades.

https://aave.com/docs/developers/smart-contracts/pool-addresses-provider

Recommendation

The Aave developers recommend that whenever a specific pool address is required, it should be fetched from this smart contract. Thus, a reference to the address provider contract should be stored, and the relevant pool fetched on-demand.

File Hashes

- `contracts/src/Ern.sol`
- `a26c587748ed664fcb32d7a91f0d91159ec6cba8da9f415289e3168f10966ab2`

Mitigations

- `contracts/src/Ern.sol`
- `b3746a33fe312a5931054b26859337e9bda3c9bfc9d854c1b5e48688ed18246b`

Disclaimer

Creed ("CD") typically receives compensation from one or more clients (the "Clients") for performing the analysis contained in these reports (the "Reports"). The Reports may be distributed through other means, including via CD publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of code and only the code we note as being within the scope of our review within this report. Any Solidity code itself presents unique and unquantifiable risks as the Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond specified code that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. In some instances, we may perform penetration testing or infrastructure assessments depending on the scope of the particular engagement.

CD makes the Reports available to parties other than the Clients (i.e., "third parties") – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that CD are not responsible for the content or operation of such Web sites, and that CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. CD assumes no responsibility for the use of third party software on