Quantstamp Security Assessment Certificate

August 31st 2021 — Quantstamp Verified

Alkemi

This audit report was prepared by Quantstamp, the leading blockchain security company.

Executive Summary

Type	Lending platform
Auditors	Joseph Xu, Technical R&D Advisor Ed Zulkoski, Senior Security Engineer Sebastian Banescu, Senior Research Engineer
Timeline	2021-07-06 through 2021-08-31
EVM	Berlin
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Earn User Guide Alkemi Earn Contracts
Documentation Quality	Low
Test Quality	Low

Source Code

Repository	Commit
alkemi-earn-contracts (audit)	03479f9
alkemi-earn-contracts (re-audit)	69a1f54

Total Issues	33 (28 Resolved)
High Risk Issues	6 (6 Resolved)
Medium Risk Issues	8 (8 Resolved)
Low Risk Issues	11 (9 Resolved)
Informational Risk Issues	6 (4 Resolved)
Undetermined Risk Issues	2 (1 Resolved)

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
Undetermined	The impact of the issue is uncertain.
Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Resolved	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

Initial audit (Commit `03479f9`):
Quantstamp has performed an audit of the Alkemi Earn Contracts, which is a DeFi lending protocol with KYC integration. We have found 33 issues in total with 6 issues marked as "High" severity. Notably, one of the "High" severity issue is the same vulnerability that has been exploited in the Lendf.me protocol, which has lead to the loss of $25 million in funds. We also document a variety of vulnerabilities that may arise from handling of decimal units, which are not handled in a very consistent manner throughout the codebase. There are also additional issues related to the ALK token distribution and user verification processes.

We strongly recommend addressing all findings, including the best practice issues mentioned in this report. In particular, we recommend investing time into developing a comprehensive technical documentation of the contracts. Having a robust technical specification, including detailed outlines of the set of operations to be performed in each core functionality, will greatly help reduce the risk of exploit and improve code maintainability.

Re-audit (Commit `69a1f54`):
A great majority of the issues are now Fixed in the re-audit, including most of the "High" severity issues. The "High" severity decimal standardization issue is marked as Mitigated, with the Alkemi team providing additional code comments. Remaining issues are marked as Acknowledged, but these are all "Low", "Informational", or "Undetermined" severity issues. The auditors note that the Alkemi team has communicated with the auditors actively during the re-audit process and clearly demonstrated their interest in resolving as many issues as possible.

During the re-audit, the Alkemi team introduced changes to the `RewardControl.sol` contract. The auditors reviewed the changes and suggested several fixes. These fixes are included in the re-audit commit `69a1f54`. However, this review was not conducted with the full team and should be considered as out of the formal audit scope.

ID	Description	Severity	Status
QSP-1	Reentrancy in functions related to the main lending operations	High	Fixed
QSP-2	Lack of standardization or documentation on decimal units	High	Mitigated
QSP-3	The oracle contract can return token value in USD term but the core contracts assume price in ETH term	High	Fixed
QSP-4	The oracle contract returns price feed with inconsistent decimal units	High	Fixed
QSP-5	The price feed may experience integer underflow if an asset has more than 18 decimals	High	Fixed
QSP-6	ALK token allocation does not account for token decimals or prices	High	Fixed
QSP-7	Customer KYC status can be changed by anybody	Medium	Fixed
QSP-8	Unverified users can perform liquidation	Medium	Fixed
QSP-9	Ambiguous naming and usage of functions and structs in core math contracts	Medium	Fixed
QSP-10	Possible integer underflow in core math contracts	Medium	Fixed
QSP-11	Oracle prices used could be stale	Medium	Fixed
QSP-12	No check for paused oracle	Medium	Fixed
QSP-13	`_withdrawEquity()` changes the utilization rate but does not update the interest rate	Medium	Fixed
QSP-14	Incorrect supply rate decimal units in the return value of `getSupplyRate()`	Medium	Fixed
QSP-15	Removing a market does not reset its ALK token distribution speed to zero	Low	Fixed
QSP-16	ALK speeds are insufficiently updated	Low	Acknowledged
QSP-17	`_withdrawEquity()` does not account for interest accrual	Low	Fixed
QSP-18	`USDETHPriceFeed` assumed to have 8 decimals	Low	Fixed
QSP-19	Missing input validations	Low	Fixed
QSP-20	Inaccurate number of blocks per year	Low	Fixed
QSP-21	Interest rate model may be degenerate	Low	Fixed
QSP-22	Interest rate model ignores return values from `CarefulMath`	Low	Fixed
QSP-23	Changes in interest rates could decrease user balances	Low	Acknowledged
QSP-24	For-loop / Gas issues	Low	Fixed
QSP-25	References to test contract in production code	Low	Fixed
QSP-26	Removing an asset price feed does not reset flag	Informational	Fixed
QSP-27	Minimum and maximum limits are not fixed	Informational	Fixed
QSP-28	Updated cash values may be incorrect for deflationary or fee-based tokens	Informational	Acknowledged
QSP-29	Unlocked pragma	Informational	Fixed
QSP-30	Clone-and-own	Informational	Fixed
QSP-31	Privileged roles and ownership	Informational	Acknowledged
QSP-32	Mismatch in inline requirements, comments, and implementation	Undetermined	Fixed
QSP-33	Very old compiler version	Undetermined	Acknowledged

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.8.0
- Mythril c0.22.21

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`
1. Install the Mythril tool: `pip3 install mythril`
2. Run the Mythril tool on each contract: `myth analyze path/to/contract`

Findings

QSP-1 Reentrancy in functions related to the main lending operations

Severity: *High Risk*

Status: Fixed

File(s) affected: `AlkemiEarnPublic.sol`, `AlkemiEarnVerified.sol`

Description: A reentrancy vulnerability is a scenario where an attacker can repeatedly call a function from itself, unexpectedly leading to potentially disastrous results. The main contracts `AlkemiEarnPublic.sol` and `AlkemiEarnVerified.sol` are both vulnerable to the same reentrancy attack vector experienced by the Lendf.me hack.

The reentrancy vulnerability stems from the fact that the functions handling the main lending operations `supply()`, `withdraw()`, `borrow()`, `repayBorrow()`, `liquidateBorrow()` all maintain their intermediate balance updates within respective local variables. These local variables updates are eventually written to the `Market` struct and each user's `Balance` struct, but these writes occur after transfers are performed using `doTransferIn()`, `doTransferOut()`, `withdrawEther()`, or `revertEtherToUser()`. This makes it possible create a reentrant transaction through fallback function or ERC-777 hook before the market and user balances are updated (i.e., the in-line comment `EFFECTS & INTERACTIONS` is incorrect, since interactions are taking place before effects).

Fortunately, the Alkemi protocol does not currently support ERC-777 tokens (the aforementioned attack was carried out using the imBTC token, which follows the ERC-777 standard). However,

the contracts should be updated to eliminate potential reentrancy to mitigate the risk of financial loss.

Recommendation: Modify the functions so that the `Market` and user `Balances` structs are updated before transfers take place. For additional security, consider adding an additional lock mechanism such as OpenZeppelin's `ReentrancyGuard` to the functions that carry out any token transfers.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/9. The main lending operations are changed to adhere to the checks-effects-interactions pattern. `ReentrancyGuard.sol` is used and the `nonReentrant` modifier is applied to functions `supply()`, `withdraw()`, `borrow()`, and `repayBorrow()`. The `nonReentrant` modifier was not applied to `liquidateBorrow()` due to contract size limit and "stack too deep" error. While `liquidateBorrw()` should also ideally be modified with `nonReentrant`, this issue is considered Fixed overall based on changes to the lending operation procedures.

QSP-2 Lack of standardization or documentation on decimal units

Severity: *High Risk*

Status: Mitigated

File(s) affected: `AlkemiEarnPublic.sol`, `AlkemiEarnVerified.sol`, `AlkemiRateModel.sol`, `ChainLink.sol`

Description: The contracts in this protocol use a variety of decimal units across variables and function return values. A list of the decimal units used in the contracts is as follows:

- `expScale`: Used for most calculations and for generic token quantities (1 = 10^18)
- `token.decimal`: Used for token quantities whose decimal is not in `expScale`
- `doubleScale`: Used for value = price * quantity (1 = 10^36)
- Price scale : Used for the oracle price feed (1 = `10**18 * (10**(18 - token.decimal))`)
- Unit scale: Equivalent to `expScale`, but used for the interest rate model (1 = 10^18)
- Percentage scale: Used for interest rate model (1 = 10^20)
- Basis point scale: Indicated in the interest rate model comments, but does not seem to be actually used (1 = 10^22)

The usage of the decimal units are not standardized within or across contracts. While certain variables and functions have in-line comments that indicate the intended decimal units, there are no comprehensive documentation or enforcement on how the contracts handle the decimal units across variables and functions. In this report, we identify several High to Medium severity issues whose root cause is the mis-handling of the decimal units. More generally, the current approach to handling decimal units makes the contracts extremely prone to serious computation errors that can put protocol funds at risk.

Recommendation: We strongly recommend writing a rigorous documentation on the handling of decimal units at the variable and the function level. Each variable, including intermediate values in calculation, should have unambiguous labeling of decimal units. Functions should also have its input parameters and return value decimal units referenced in an easily accessible manner. We recommend limiting the decimal units to `expScale` and `doubleScale` throughout the contracts as much as possible.

Having a proper documentation on decimal unit handling will help significantly reduce the risk of severe bugs and also increase the maintanability of the contracts. Test codes should also be written with decimal unit checks in mind.

Update: Mitigated in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/24 and https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/31. The Alkemi team addressed this issue mainly through additional code comments at the function level. While this does improve the situation, we still recommend a refactor to standardize decimal units (in future versions of the protocol) or a separate document that comprehensively track the decimal usages.

QSP-3 The oracle contract can return token value in USD term but the core contracts assume price in ETH term

Severity: *High Risk*

Status: Fixed

File(s) affected: `AlkemiEarnPublic.sol`, `AlkemiEarnVerified.sol`, `ChainLink.sol`

Description: In the oracle contract, setting `assetsWithPriceFeedBasedOnUSD[asset] = true` will return the token price in token/USD term at `expScale` (1 = 10^18). However, all of the calculations in the core contracts `AlkemiEarnPublic.sol` and `AlkemiEarnVerified.sol` assume that the oracle price feed returns in token/ETH term at `10**18 * (10**(18 - token.decimal))` scale. This is according to the comment on the last line in the `getPriceForAssetAmount()` function: `assetAmountWei * oraclePrice = assetValueInEth`. The same issue applies to the `getAssetAmountForValue()` function and other functions which uses `fetchAssetPrice()` assuming the price returned by that function is in token/ETH term.

Introducing a market whose asset price feed is in token/USD term by `assetsWithPriceFeedBasedOnUSD[asset] = true` can have severe consequences for the protocol.

Recommendation: Remove the code related to `assetsWithPriceFeedBasedOnUSD` from the oracle contract so that the oracle contract can only return token price in ETH terms.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/3.

QSP-4 The oracle contract returns price feed with inconsistent decimal units

Severity: *High Risk*

Status: Fixed

File(s) affected: `ChainLink.sol`

Description: The decimal units of the return value from `getAssetPrice()` function can vary based on the asset and the mapping `assetsWithPriceFeedBasedOnUSD[asset]`.

1. If the `asset` is WETH (condition on L116) it returns `1e18`.
2. If the `asset` has an associated price feed AND `assetsWithPriceFeedBasedOnUSD[asset] = true` (condition on L121 and L132), then it returns the token's USD price aligned to 18 decimals (if the `asset` has less than 18 decimals).
3. If the `asset` has an associated price feed, but the `assetsWithPriceFeedBasedOnUSD[asset] = false` (L121 and `else`-branch on L146), then it returns the token's ETH price inflated by `10**(18 - token.decimal)`.

The 3rd case enumerated is most problematic because the `asset` has may not have 18 decimals (e.g., 6 decimals in case of USDC). However the USDC/ETH price feed has 18 decimals, which means `price` on L149 has 18 decimals and the operation on L150 increases it to 30 decimals. For tokens with 18 decimals, the 3rd case still returns 18 decimals.

While the calculations on the main contract appear unaffected since the oracle return values are in ETH terms and consistently used to generate `doubleScale` (1 = 10^36), this approach can be extremely risky because the main contract may not be aware of the token decimal adjustment that took place in the oracle contract.

Recommendation: We recommend always returning a consistent decimal units for the price feed. Magnification by token decimals should take place in the main contract instead of in the oracle contract.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/8.

QSP-5 The price feed may experience integer underflow if an asset has more than 18 decimals

Severity: *High Risk*

Status: Fixed

File(s) affected: `ChainLink.sol`

Description: If an asset has more than 18 decimals, the subtraction on L150 will underflow and produce abnormally large values.

Recommendation: Use `SafeMath.sub()` instead of subtracting . Having said that, we recommend removing this operation so that the oracle contract can return consistent decimal units in `getAssetPrice()`.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/8 and https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/30.

QSP-6 ALK token allocation does not account for token decimals or prices

Severity: *High Risk*

Status: Fixed

File(s) affected: `RewardControl.sol`

Description: The function `refreshAlkSpeeds()` calculates the proportion of ALK token distribution that should be allocated to each market. However it only considers the total supply and borrow amounts in terms of the token quantity.

There are two issues:

1. This unfairly favors tokens that have more decimals. For example, DAI with 18 decimals is favored over USDC with 6 decimals, assuming the same value of total liquidity.
2. Even if the number of decimals are equivalent, prices must be considered. For example, 1 WETH should be weighted much higher than 1 DAI, even though both have 18 decimals.

Recommendation: Incorporate token decimals and total value of the liquidity when determining ALK token allocations.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/8, https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/19, and https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/29.

QSP-7 Customer KYC status can be changed by anybody

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `AlkemiEarnVerified.sol`

Description: In the function `_changeCustomerKYC()`, the following lines (L601-606) simply emit an event and do not cause the function to revert. This allows any address to update the mapping `customersWithKYC`.

```
if (!KYCAdmins[msg.sender]) {
    emitError(
        Error.KYC_ADMIN_CHECK_FAILED,
        FailureInfo.KYC_ADMIN_CHECK_FAILED
    );
}
```

Recommendation: Revert or return from the function if `msg.sender` is not a KYC admin.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/4.

QSP-8 Unverified users can perform liquidation

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `AlkemiEarnVerified.sol`

Description: In the function `_changeCustomerKYC()`, the following lines (L2461-2466) simply emit an event and do not cause the function to revert. This allows any address to act as a liquidator.

```
if (!liquidators[msg.sender]) {
    emitError(
        Error.LIQUIDATOR_CHECK_FAILED,
        FailureInfo.LIQUIDATOR_CHECK_FAILED
    );
}
```

Note: This check is implemented as a modifier in the older live contract, which seems to work as intended.

Recommendation: Revert or return from the function if `msg.sender` is not a liquidator.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/5.

QSP-9 Ambiguous naming and usage of functions and structs in core math contracts

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `CarefulMath.sol`, `Exponential.sol`

Description: In `Exponential.sol`, there are two functions `addExpNegative()` and `subExpNegative()`, which are used in the interest rate model. Despite similar naming, there are differences in the behavior of these functions:

- `addExpNegative()` takes an `Exp` struct with `uint256 mantissa` and an `ExpNegative` struct with `int256 mantissa` to return a `Exp` struct with `uint256 mantissa`.

- `subExpNegative()` takes two `Exp` structs with `uint256 mantissa` to return a `ExpNegative` struct with `int256 mantissa`.

The `ExpNegative` structure has one attribute `int256 mantissa`, which can be a positive integer. However, since there is another structure called `Exp` which has one attribute `uint256 mantissa`, we assume that the `ExpNegative` should only hold negative exponent values. This is not enforced in `subExpNegative` and if `a.mantissa > b.mantissa` when calling this function, then the output will be a `ExpNegative` structure with a positive `mantissa`. Similar issues exist in functions `addInt()` and `subInt()` within `CarefulMath.sol`, which are called by `Exponential.addExpNegative()` and `Exponential.subExpNegative()` respectively. These functions have similar naming, but their argument and return value types differ.

Recommendation: The return value of `subExpNegative()` may be either `Exp` or `ExpNegative`. Similarly, the return value of `addExpNegative` may be either `Exp` or `ExpNegative`. Update the function names to better match their usage. Otherwise, enforce that the output of `subExpNegative()` should only be negative.

It is probably best to create a more general structure that can accommodate both positive and negative exponents and use that instead of having separate structures for positive and negative exponents.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/10. Struct `ExpNegative`, functions `CarefulMath.subInt()`, `CarefulMath.addInt()`, `Exponential.addExpNegative()`, and `Exponential.subExpNegative()` are removed. The interest rate calculations are updated to eliminate uses of these struct and functions.

QSP-10 Possible integer underflow in core math contracts

Severity: Medium Risk

Status: Fixed

File(s) affected: `CarefulMath.sol`, `Exponential.sol`

Description: The `Exponential.addExpNegative()` function is subject to an integer underflow because the output `result` of the `CarefulMath.addInt()` function, which is of type `int256` (signed integer), may be negative. However, `result` is cast to a `uint256` (unsigned integer) in the `return` statement on L72.

Recommendation: The return value of `addExpNegative()` may be either `Exp` or `ExpNegative`. It is probably best to create a more general structure that can accommodate both positive and negative exponents and use that instead of having separate structures for positive (`Exp`) and negative exponents (`ExpNegative`).

In addition, we recommend adding over-/underflow checks in the `CarefulMath.addInt()` and `CarefulMath.subInt()` functions and corresponding error-handling in `Exponential.sol` functions.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/10. Struct `ExpNegative`, functions `CarefulMath.subInt()`, `CarefulMath.addInt()`, `Exponential.addExpNegative()`, and `Exponential.subExpNegative()` are removed. The interest rate calculations are updated to eliminate uses of these struct and functions.

QSP-11 Oracle prices used could be stale

Severity: Medium Risk

Status: Fixed

File(s) affected: `ChainLink.sol`

Description: The `ChainLink.getAssetPrice()` function does not check the return values of the `priceContractMapping[asset].latestRoundData();` function call on L128. This approach is vulnerable to stale prices according to the Chainlink documentation under current notifications: "if answeredInRound < roundId could indicate stale data." The `timestamp` and `answerInRound` values are not consulted beyond checking that `timestamp > 0`.

The same applies to the `USDETHPriceFeed.latestRoundData();` function call on L140.

Recommendation: We recommend adding require statements to ensure that `timestamp` and `answerInRound` were updated recently based on some tolerance.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/6 and https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/31. The price is considered stale if not updated for > 1 hour OR answered in less than `roundID`.

QSP-12 No check for paused oracle

Severity: Medium Risk

Status: Fixed

File(s) affected: `AlkemiEarnPublic.sol`, `AlkemiEarnVerified.sol`, `ChainLink.sol`

Description: The oracle contract returns a price of 0 for all assets if paused. However, the main contracts' call to the oracle contract `fetchAssetPrice()` does not check for paused oracle. While zero price is outside of the `fetchAssetPrice()` function using `isZeroExp()`, this is not always the case. In the main contracts, functions `getAssetAmountForValue()` and `liquidateBorrow()` do not use `isZeroExp()` to check for zero price returned by the oracle.

Recommendation: Add a check for paused oracle in `fetchAssetPrice()`. Alternatively, handle all zero price result in the `fetchAssetPrice()` function as opposed to outside of the call to the oracle contract.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/7.

QSP-13 `_withdrawEquity()` changes the utilization rate but does not update the interest rate

Severity: Medium Risk

Status: Fixed

File(s) affected: `AlkemiEarnPublic.sol`, `AlkemiEarnVerified.sol`

Description: When the admin address withdraws equity of an asset using `_withdrawEquity()`, the protocol's cash balance for the asset is changed. However, this does not trigger updates to the interest rate, even though the interest rate is dependent on the current cash balance for the asset.

Recommendation: Add interest rate update procedures in `_withdrawEquity()`.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/12.

QSP-14 Incorrect supply rate decimal units in the return value of `getSupplyRate()`

Severity: Medium Risk

Status: Fixed

File(s) affected: `AlkemiRateModel.sol`

Description: The intermediate results of `getSupplyRate()` is consistently in terms of percentage units (1% = 10^18) but the return value of this function must be in `expScale` (1% = 0.01 = 10^16). However, the actual return value `supplyRate.mantissa` is on the scale 1% = 10^12 due to an extra division by 10000.

This can be seen by tracing the computations in `getSupplyRate()` from L296-323, shown below:

```
(err1, temp1) = getExp(100, 1);    // L296 -- temp1 = 100% = 1e20, so temp1: 1% = 1e18
[...]
(err1, oneMinusSpreadBasisPoints) = subExp(temp1, SpreadLow);    // L298 -- temp1: 1% = 1e18; SpreadLow: 1% = 1e18; oneMinusSpreadBasisPoints: 1% = 1e18
[...]
(err1, temp1) = mulExp(utilizationRate0, annualBorrowRate);    // L306 -- utilizationRate0: 1% = 1e18; annualBorrowRate: 1% = 1e16; temp1: 1% = 1e16
[...]
(err1, temp1) = mulExp(temp1, oneMinusSpreadBasisPoints);    // L315 -- temp1: 1% = 1e16; oneMinusSpreadBasisPoints: 1% = 1e18; temp1: 1% = 1e16
[...]
(Error err4, Exp memory supplyRate) = divScalar(
    temp1,    // L321 -- temp1: 1% = 1e16
    10000 * blocksPerYear
); // basis points * blocks per year -- supplyRate: 1% = 1e12
```

The resulting `supplyRate` is expressed in 1% = 10^12 scale. This is due to the `annualBorrowRate` having already been adjusted down to the 1% = 10^16 scale in the `getUtilizationAndAnnualBorrowRate()` return value. The comment appears to believe that `temp1` is actually expressed in basis point scale where 1% = 0.01 = 10^20.

Recommendation: For a simple fix, the final division should be by `blocksPerYear` in `getSupplyRate()`. Having said that, we recommend a refactor where the return value `Exp({mantissa: annualBorrowRateScaled / 100})` of `getUtilizationAndAnnualBorrowRate()` (L261) is not divided by 100; this adds confusion by having the returns values -- the utilization rate and the borrow rate -- in different scales. Scaling should be done in `getSupplyRate()` and `getBorrowRate()` after all intermediate results are expressed in 1% = 10^18 scale. Dividing by `100 * blocksPerYear` at the end would put all the results in line with the desired 1% = 10^16 scale.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/11.

QSP-15 Removing a market does not reset its ALK token distribution speed to zero

Severity: *Low Risk*

Status: Fixed

File(s) affected: `RewardControl.sol`

Description: Removing a market using the `removeMarket()` function will not reset the value of the corresponding market under the mapping `alkSpeeds` to zero. If the market still has non-zero supply or borrow amounts, the ALK speeds will no longer be correct. This could affect users that invoke `claimAlk()` which does not perform a refresh.

Recommendation: When removing a market, make sure to zero the distribution speed for the removed market and call `refreshAlkSpeeds()` to update the distribution speed for the remaining markets.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/20 and https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/29.

QSP-16 ALK speeds are insufficiently updated

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `RewardControl.sol`

Description: The functions `refreshAlkSupplyIndex()` and `refreshAlkBorrowIndex()` only update the market currently being engaged by the supplier/borrower. This may cause the overall ALK distribution to deviate from `alkRate`.

Exploit Scenario: Consider the following extreme scenario:

1. Suppose a market `M` has not had any interactions for a large number of blocks `B` and has an average `alkSpeed` of `S` during this window, such that `M` should be alloted `B * S` ALK tokens for this time.
2. After `B - 1` blocks in this window, suppose a substantial supply of tokens enter a different market. This causes all market speeds to refresh, which would reduce the allocation for `M` to some `s` such that `s < S`.
3. A user supplies a negligible amount of tokens to market `M` at block `B`, causing `updateAlkSupplyIndex(M)` to be invoked. This results in the supply index state of `M` being updated based on `s * B`, even though it should be allotted `S * (B - 1) + s * 1`.

The converse scenario could also occur if a large number of tokens withdraw from other markets after `B - 1` blocks. In this scenario, `M` would be alloted more tokens than earned. Note that since these two functions are invoked in the beginning of `supply()`, it appears that flash loan attacks are at least mitigated for this issue.

Recommendation: Consider whether the potential deviation from `alkRate` is likely to be substantial in practice. While refreshing all markets for every interaction would resolve the issue, this would cause substantial gas increases and may not be feasible.

Update: The team indicates that this is a design decision to minimize gas consumption. The assumption is that the markets will be used frequently enough to allow suitable updates of `alkRate` (or minimal deviation, if any).

QSP-17 `_withdrawEquity()` does not account for interest accrual

Severity: *Low Risk*

Status: Fixed

File(s) affected: `AlkemiEarnPublic.sol`, `AlkemiEarnVerified.sol`

Description: `_withdrawEquity()` computes the equity available to the admin by simply using the asset's cash balance, the current total borrows (`markets[asset].totalBorrows`) and the current total supply (`markets[asset].totalSupply`). However, the real-time equity should account for the interest accrued.

Recommendation: Add interest rate accrual update in `_withdrawEquity()` before computing the available equity.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/13.

QSP-18 `USDETHPriceFeed` assumed to have 8 decimals

Severity: *Low Risk*

Status: Fixed

File(s) affected: `ChainLink.sol`

Description: The `ChainLink.getAssetPrice()` function contains an implicit assumption that the `USDETHPriceFeed.latestRoundData()` function call on L140 will always have 8 decimals. However, this assumption can be broken because the admin is free to set the `USDETHPriceFeed` address to any value from this list.

Moreover, the name of this state variable is misleading because it might give the impression that the correct feed should be a USD-pegged coin to ETH (which has 18 decimals), while the actual intention from its usage on L143 in `ChainLink.sol` indicates that it should actually be a ETH/USD price feed (which has 8 decimals). This name gives way to human error from the admin's side.

Recommendation: Add an assertion to check that the number of decimals of the `USDETHPriceFeed` contract is equal to 8 inside the `ChainLink.setUSDETHPriceFeedAddress()` function. Or replace `10**8` on L144 with `USDETHPriceFeed.decimals()`.

While it is unlikely that Chainlink price feed decimals will change in the near future (the oracle contract currently assumes USD and ETH price feeds will always remain 8 and 18 decimals respectively), we recommend always checking for the price feed decimal scales in `getAssetPrice()` as an extra precaution.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/3. `USDETHPriceFeed` is removed entirely.

QSP-19 Missing input validations

Severity: *Low Risk*

Status: Fixed

File(s) affected: `AlkemiEarnPublic.sol`, `AlkemiEarnVerified.sol`, `ChainLink.sol`, `Liquidator.sol`, `RewardControl.sol`

Description: The following functions are missing input parameter validation:

1. `AlkemiEarnVerified.setWethAddress()`does not validate the `wethContractAddress` parameter.
2. `AlkemiEarnVerified._adminFunctions()` does not validate the addresses should be checked, as well as the values of both mantissas (which should be `<= 10**18`).
3. `AlkemiEarnVerified._supportMarket()` should ensure that `interestRateModel` is non-zero.
4. `AlkemiEarnVerified._setMarketInterestRateModel()` should ensure that `interestRateModel` is non-zero.
5. `AlkemiEarnVerified._setRiskParameters()` does not validate bounds on `_minimumCollateralRatioMantissa` or `_maximumLiquidationDiscountMantissa`.
6. `ChainLink.addAsset()` does not validate the `assetAddress` and `preiceFeedContract` parameters.
7. `ChainLink.removeAsset()` does not validate the `assetAddress` parameter.
8. `ChainLink.changeAdmin()` does not validate the `newAdmin` parameter.
9. `ChainLink.setWethAddress()` does not validate the `_wethAddress` parameter.
10. `ChainLink.setUSDETHPriceFeedAddress()` does not validate the `_USDETHPriceFeed` parameter.
11. `Liquidator.constructor()` does not validate the `alkemiEarnVerified_` parameter.
12. `RewardControl.initializer()` does not validate the `_owner`, `_alkemiEarnVerified`, and `_alkAddress` parameters.

Recommendation: Add the necessary input parameter validations.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/23.

QSP-20 Inaccurate number of blocks per year

Severity: *Low Risk*

Status: Fixed

File(s) affected: `AlkemiRateModel.sol`, `USDTRateModel.sol`, `USDxInterestRateModel.sol`

Description: The 3 affected contracts all declare a `uint256 constant blocksPerYear = 2102400;`, which assumes that the block time is 15 seconds. In reality the current average block time is 13.3 seconds, while the block time for ETH 2.0 is set to 12 seconds. This means that the could be between 10%-20% more blocks per year than the current implementation assumes, which means the borrow and supply rates could be 10-20% higher than reported.

Recommendation: Update the default number of blocks per year to a number closer to the actual number of blocks in a calendar year. Offer the governance role the possibility of updating this value in case of significant changes in the block time.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/14.

QSP-21 Interest rate model may be degenerate

Severity: *Low Risk*

Status: Fixed

File(s) affected: `AlkemiRateModel.sol`

Description: The current interest rate model implementation does not check for reasonable interest rate model parameters in its constructor and `changeRates()` function. Error codes in the intermediate calculations are overlooked.

Examples of reasonable parameter checks in the current context can include the following items:

- `HealthyMinURActual < HealthyMaxURActual`
- `HealthyMinRate < HealthyMaxRate`
- `MinRate < MaxRate`

While the current interest rate model accounts for the possibility of `SpreadHigh` taking a negative value, but `SpreadMid` may also be negative depending on other parameters.

Recommendation: List out properties of a well-behaved interest rate model and add checks to the constructor and `changeRates()` . Consider using `Struct ExpNegative` for `SpreadMid`.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/10.

QSP-22 Interest rate model ignores return values from `CarefulMath`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `AlkemiRateModel.sol`

Description: As opposed to the `SafeMath` library of OpenZeppelin, the `CarefulMath` library used in this project has functions which do not throw in case an error such as an integer overflow/underflow or a division by zero occurs. Therefore, the implementation should always check the return value of the function calls for potential errors when using `CarefulMath`. This is not currently the case in the `AlkemiRateModel.constructor()` and `AlkemiRateModel.changeRates()`, which leaves room for undetected human error.

Recommendation: Either add assertions to check that no error occurs after each call, or use functions similar to `SafeMath`, which revert in case of an error.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/15 and https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/31.

QSP-23 Changes in interest rates could decrease user balances

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `AlkemiEarnPublic.sol`, `AlkemiEarnVerified.sol`, `InterestRateModel.sol`

Description: If the supply or borrow rate mantissas returned by the interest rate model are reduced from their previous values, the balances of end users could decrease. This could happen either intentionally, or potentially accidentally if the rate model is updated by the admin. It is not clear if this is intended, or if user balances should monotonically increase over time.

Recommendation: Clarify if this should be allowed. If not, add `require()` statements ensuring that the rate never decreases.

Update: The team indicates that this is a confirmed behavior that is inherited from Lendf.me and also seen in other lending protocols. This is a design decision to optimize gas usage. The expectation is that in practice, transactions will be frequent enough and interest rate model updates will be rare enough that user balances will be minimally affected.

QSP-24 For-loop / Gas issues

Severity: *Low Risk*

Status: Fixed

File(s) affected: `AlkemiEarnPublic.sol`, `AlkemiEarnVerified.sol`, `RewardControl.sol`

Description: The following functions contain loops that iterate over potentially large numbers of entries in arrays and could cause a DoS via out-of-gas errors:

1. `AlkemiEarn*.calculateAccountValuesInternal()` iterates over all `collateralMarkets` and performs heavy computations for each market.
2. `RewardControl.refreshAlkSpeeds()` contains 2 for-loops each iterating over `allMarkets`.

Recommendation: Perform a gas analysis and place a hard cap on the number of allowed markets.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/21 and https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/29.

QSP-25 References to test contract in production code

Severity: *Low Risk*

Status: Fixed

File(s) affected: `ChainLink.sol`

Description: The oracle contract calculates the token decimal using a test contract when fetching price in function `getAssetPrice()` (see L120: `uint8 assetDecimals = TestTokens(asset).decimals();`).

Recommendation: Use a standard ERC-20 interface such as `EIP20Interface.sol`

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/16.

QSP-26 Removing an asset price feed does not reset flag

Severity: *Informational*

Status: Fixed

File(s) affected: `ChainLink.sol`

Description: The `ChainLink.removeAsset()` function does not set `assetsWithPriceFeedBasedOnUSD[assetAddress] = false` in case it is set to `true`. There is no immediate impact due to this reason, but this flag should also be reset if needed to avoid any issues in case the code is extended to take decisions based on `assetsWithPriceFeedBasedOnUSD[assetAddress]` in any future versions of the code.

Recommendation: Reset flag when removing asset. Having said that, we recommend removing the mapping `assetsWithPriceFeedBasedOnUSD` from the contract entirely.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/3. The mapping `assetsWithPriceFeedBasedOnUSD`is removed entirely.

QSP-27 Minimum and maximum limits are not fixed

Severity: *Informational*

Status: Fixed

File(s) affected: `AlkemiEarnPublic.sol`, `AlkemiEarnVerified.sol`

Description: The `minimumCollateralRatioMantissa` and `maximumLiquidationDiscountMantissa` state variables of the `AlkemiEarn*.sol` contracts are not immutable. They can be changed along with the collateral ratio and liquidation discount using the `_setRiskParameters()` function. This increases the risk of human error because there is no fixed threshold for either the minimum collateral ratio or the maximum liquidation discount.

Recommendation: Fix the minimum and maximum parameters as constants.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/12 and https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/31.

QSP-28 Updated cash values may be incorrect for deflationary or fee-based tokens

Severity: *Informational*

Status: Acknowledged

File(s) affected: `AlkemiEarnPublic.sol`, `AlkemiEarnVerified.sol`

Description: Certain tokens (e.g., DGX) incorporate fees into token transfers, such that the amount of tokens received will be smaller than the amount sent. This could cause incorrect `updatedCash` computation, such as in `repayBorrow()`:

```
(err, localResults.updatedCash) = add(
    localResults.currentCash,
    localResults.repayAmount
);
```

Recommendation: Consider checking the updated cash value after the transfer is performed.

Update: The team indicates that markets for deflationary or fee-based tokens will not be created. The administrator role will be in control of adding token markets during the early stages of the protocol. There are plans to include explicit code-based solutions to these types of tokens for future protocol updates.

QSP-29 Unlocked pragma

Severity: *Informational*

Status: Fixed

Description: Most contract files in the repository specify in the header a version number of the format `pragma solidity (^)0.4.24`. The caret (^) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, it is recommended to remove the caret to lock the file onto a specific Solidity version. Later compiler versions may introduce breaking changes that are incompatible with the current contracts.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/17.

QSP-30 Clone-and-own

Severity: *Informational*

Status: Fixed

File(s) affected: `AlkemiWETH.sol`, `CarefulMath.sol`, `Exponential.sol`, `ExponentialNoError.sol`, `SafeMath.sol`

Description: The clone-and-own approach involves copying and adjusting open source code at one's own discretion. From the development perspective, it is initially beneficial as it reduces the amount of effort. However, from the security perspective, it involves some risks as the code may not follow the best practices, may contain a security vulnerability, or may include intentionally or unintentionally modified upstream libraries. Several important contracts in this repository (wrapped Ether and core math operations) are based on cloned code.

Recommendation: Rather than the clone-and-own approach, a good industry practice is to use the Truffle framework for managing library dependencies. Otherwise, include code comments that point to the original source code.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/18 and https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/31.

QSP-31 Privileged roles and ownership

Severity: *Informational*

Status: Acknowledged

File(s) affected: `AlkemiEarnPublic.sol`, `AlkemiEarnVerified.sol`

Description: Smart contracts will often have `admin` address or other roles to designate person(s) with special privileges to make modifications to the smart contract. In the context of this protocol, `admin` can use `_adminFunctions()` to set new fee and oracle values.

Recommendation: The privileges allowed for the `admin` address need to be made clear to the users, especially if strong levels of privileges are allowed. For other roles, carefully document the privileges and capabilities of each role.

Update: The team acknowledges that users will need to trust the administrator, particularly in the early stages of the protocol. The team plans to update the documentation to reflect this. The team has also indicated plans to eventually decentralize the protocol and move away from an explicit administrator role in the future through governance and protocol updates.

QSP-32 Mismatch in inline requirements, comments, and implementation

Severity: *Undetermined*

Status: Fixed

File(s) affected: `AlkemiEarnPublic.sol`, `AlkemiEarnVerified.sol`, `AlkemiRateModel.sol`, `ChainLink.sol`, `RewardControl.sol`, `TestToken.sol`

Description: The following instances of comments being misaligned with the implementation were observed:

1. On L777 of `AlkemiEarnPublic.sol`, the contract states regarding `originationFeeMantissa`: "The de-scaled value must be >= 1.1". First, no limit is enforced in the code (as mentioned in an above issue). Second, it is not clear if this value is correct, since `defaultOriginationFee = (10**15)`, which would have de-scaled value of 0.001.
2. On L866 of `AlkemiEarnPublic.sol` in reference to `getAccountLiquidity()`, we have the comment: "returns account liquidity in terms of eth-wei value, scaled by 1e18". However the return value does not appear to be scaled due to the use of `truncate`.
3. For the function `AlkemiEarnPublic._withdrawEquity()`, the comment on L1182 states that we should check "` amount` <= equity. Equity= cash - (supply + borrows)". However, because of the order-of-operations in `addThenSub`, the function instead checks if `amount > cash + borrow - supply`.
4. On L1944 of `AlkemiEarnPublic.sol`, the comment states that "// In the case of borrow, we multiply the borrow value by the collateral ratio". This operation does not occur here, but in the calling function `calculateAccountLiquidity()`.
5. In `AlkemiRateModel.getBorrowRate()`, the comment "basis points * blocks per year" does not seem to match the code, as `10000` is not included in the denominator.
6. The comment on the `ChainLink.constructor()` indicates that: "Sets the initial assets and admin". However, only the admin is set inside the constructor implementation.

7. In `RewardControl.initialize()`, regarding `alkRate = 4161910200000000000`, we have the comment "8323820396000000000 divided by 2 (for lending or borrowing)". However, `8323820396000000000 / 2 = 4161910198000000000`.
8. The comment on L6 in `TestToken.sol` indicates that: "maximum is 18 decimals". However, this is not enforced anywhere in the code. Hence the number of decimals of a token could be higher than 18.

Recommendation: Ensure that the code implementation adheres to the proper documentation. Clarify the intended range for `originationFee`, and enforce it through a `require` statement.

Update: Fixed in https://github.com/AlkemiNetwork/alkemi-earn-contracts/pull/22.

QSP-33 Very old compiler version

Severity: *Undetermined*

Status: Acknowledged

Description: The compiler version used (`0.4.24`) is quite old and may be susceptible to bugs. For a list of critical bugs in Solidity, refer to the Solidity Security Alerts page.

Recommendation: Consider development with a later version of Solidity.

Update: The team plans to use later versions of Solidity in the V2 release.

Automated Analyses

Slither

Slither analyzed 27 contracts with 75 detectors, returning 615 results. These results were inspected manually and were deteremined to be either false positive or insignificant. The Slither log is provided to the Alkemi team as a separate document due to its length (4662 lines).

Mythril

Mythril reported a total of 17 issues of Low or Medium severities across `AlkemiEarnPublic.sol`, `AlkemiEarnVerified.sol`, `ChainLink.sol`, `RewardControl.sol`, and `RewardControlStorage.sol`. The results were reviewed manually and were determined to be either false positive or insignificant.

Code Documentation

1. **[Fixed]** On L40 of `AlkemiEarnPublic.sol`, the comment "oracle must be configured via _setOracle" should instead refer to `_adminFunctions`. It should also mention that `rewardControl` and `wethAddress` are not set here either.
2. **[Fixed]** On L550 of `AlkemiEarnPublic.sol`, the comment "we multiply the most recent supply index..." should not say "supply", as the function is also used for "borrows".
3. **[Fixed]** In `AlkemiEarnVerified.sol`, the definition on L84 is recursive and therefore unclear: "interestIndex = the total interestIndex as calculated after applying the customer's most recent balance-changing action".
4. **[Fixed]** In `AlkemiEarnVerified.sol`, add the mathematical formulas for how the `principal` and `interestIndex` are exactly computed in the code comments.
5. **[Fixed]** In `AlkemiEarnVerified.sol`, `Market.interestRateModel` is not documented in comment above the structure definition.
6. **[Fixed]** On L57 of `SafeToken.sol`, "complaint" should be "compliant".

Adherence to Best Practices

1. **[Fixed]** Test files should not be in the same directory with production code. Test files should be moved to a dedicated `test/` sub-folder inside the `contracts/` folder.
2. **[Fixed]** TODO comments should be removed from production code.
3. **[Fixed]** Event parameters of type `address` should be indexed. The following instances of un-indexed parameters were found:

    . The `Liquidator` parameter inside the `LiquidatorAdded` event on L357 in `AlkemiEarnVerified.sol`.

    . The `Liquidator` parameter inside the `LiquidatorRemoved` event on L358 in `AlkemiEarnVerified.sol`.

    . The `KYCAdmin` parameter inside the `KYCAdminAdded` event on L523 in `AlkemiEarnVerified.sol`.

    . The `KYCAdmin` parameter inside the `KYCAdminRemoved` event on L524 in `AlkemiEarnVerified.sol`.

    . The `KYCCustomer` parameter inside the `KYCCustomerAdded` event on L525 in `AlkemiEarnVerified.sol`.

    . The `KYCCustomer` parameter inside the `KYCCustomerRemoved` event on L526 in `AlkemiEarnVerified.sol`.

    . Both parameters of `assetAdded` on L36 in `ChainLink.sol`.

    . The `assetAddress` parameter inside the `assetRemoved` event on L37 in `ChainLink.sol`.

    . Both parameters of `adminChanged` on L38 in `ChainLink.sol`.

    . The `wethAddress` parameter inside the `wethAddressSet` event on L39 in `ChainLink.sol`.

    . The `USDETHPriceFeed` parameter inside the `USDETHPriceFeedSet` event on L40 in `ChainLink.sol`.

    . All parameters of type `address` of all events from `AlkemiEarn*.sol`

4. **[Fixed]** Magic numbers should be replaced by named constants whose names indicate the semantic meaning of the constant. The following magic numbers have been observed:

. `100` is used multiple times in `AlkemiRateModel.sol`

. `10**8` on L144 in `ChainLink.sol`

. `18` on L150 in `ChainLink.sol`

. `10**18` is used in `Exponential.sol` on L210. However it could be replaced by `expScale`.

5. **[Acknowledged]** Unused variables or constants should be removed:

   . `mantissaOneTenth` declared on L24 in `Exponential.sol` is never used by any contract.

6. **[Fixed]** Reduce chances for human error (e.g. typos) by writing a simple expression instead of constants with large values. On L117 in `ChainLink.sol` the `1000000000000000000` is too large and should be replaced by a constant such as `DECIMALS_18 = 1e18`.

7. **[Fixed]** The use of `Exp` is sometimes inconsistent. For example, the function `AlkemiEarnPublic.calculateAccountValuesInternal()` returns `(error code, sum ETH value of supplies scaled by 10e18, sum ETH value of borrows scaled by 10e18)`; it is not clear why these scaled values are `uint256` instead of `Exp` values.

8. **[Fixed]** In `AlkemiEarnVerified.sol`, it is unclear why the following `require` statement is needed on L562-565: `require(customersWithKYC[msg.sender], "Customer is not KYC Verified");`, given that it is on the `else`-branch of the following `if`-statement: `if (!customersWithKYC[msg.sender])`.

9. **[Acknowledged]** Consider using more helper functions to reduce the length of functions such as `AlkemiEarnPublic.liquidateBorrow()`.

10. **[Acknowledged]** In `AlkemiEarnPublic.sol`, `managers` is never used.

11. **[Fixed]** In `AlkemiEarnPublic.sol`, regarding their comment: "TODO: Should we add a second arg to verify, like a checksum of `newAdmin` address?" -- since the pending admin still needs to call `_acceptAdmin`, this additional check doesn't feel necessary. Having said that, the proposed check would prevent is somehow pasting a malicious user's addresss into the `newPendingAdmin` argument somehow.

12. **[Fixed]** In `AlkemiEarnPublic.getPriceForAssetAmountMulCollatRatio()`, the `isZeroExp(assetPrice)` check on L662 is redundant to the check in `fetchAssetPrice()` and could be removed.

13. **[Fixed]** In `AlkemiEarnPublic.liquidateBorrow()`, `msg.sender` is used in some locations instead of the alias `localResults.liquidator`.

14. **[Fixed]** It is not clear why `AlkemiEarnPublic.supplyEther()` has a `user` parameter as it is unused.

15. **[Fixed]** On L1238-1240 of `AlkemiEarnPublic.sol`, it is not clear why `withdrawalerr != 0` is checked, since `withdrawEther()` can only return `NO_ERROR`. Further, the comment "success" on L1239 does not align with the check.

16. **[Fixed]** In `AlkemiRateModel.sol`, the `constructor()` could simply call `changeRates()` after setting the owner and contract name.

17. **[Fixed]** In `AlkemiRateModel.sol`, the variable name `Hund[e]redMantissa` contains a typo.

18. **[Fixed]** In `RewardControl.sol`, `getAlkRewards()` and `refreshAlkSpeeds()` could benefit from a shared helper function rather than cloning the code.

19. **[Acknowledged]** In `SafeToken.sol` on L165 of the function `doApprove`, the error code `Error.TOKEN_TRANSFER_OUT_FAILED` should instead relate to a failed approval.

Test Results

Test Suite Results

All tests (264 test cases) are passing using `npm test`. The test suite output is provided to the Alkemi team as a separate document due to its length (1293 lines).

Code Coverage

Code coverage is omitted. The unit tests include `.sol` files, which cannot be handled by `solidity-coverage`.

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

```
dc30aa39f7f6d397538b9a12b5fada560c52781572ba610f5de77d408c133e89 ./contracts/AggregatorV3Interface.sol
5fbf7ec3f26726a109efdb85455626a8bdfede298620f162eac682110b98b015 ./contracts/AlkemiEarnPublic.sol
b5efac7bd5a43bc425accfd413e786ba2465f7bf7200cfe44c0de22fbc9176a6 ./contracts/AlkemiEarnVerified.sol
308affcfab05eabdb1c76cfbe3aec99627f63f9a3380210919867541d5f321d8 ./contracts/AlkemiRateModel.sol
baecaaf70e551f73d8d6c6ec33f6f72d35e31b7c14e491c9b01ba1c3f87c5257 ./contracts/AlkemiWETH.sol
cb43924ff0af83a4e28fd31ecafe508f910d4bde6ccca995c8fb308fb6018f89 ./contracts/CarefulMath.sol
61aff625b6f456e2c39f6a97c551cc8c02006fe004d6ec4b67865112c5887475 ./contracts/ChainLink.sol
743839757fb9b3bc802fb694a1ca44698812cb306cd2067f57eddbb4984131a8 ./contracts/EIP20Interface.sol
686ac790407107ef0f634dafa4ef6c36aec72c34c8e3f2af0e565ff2c7a0c557 ./contracts/EIP20NonStandardInterface.sol
89dc0e8ad92c04e423d85cf8b2a82d00771d117b2b78624ad384dbcedc99dac6 ./contracts/ErrorReporter.sol
```

c9ae1564ea66358d4d819e74792c550621af67d7dd43bff2a5e2fcfbb2a549e2 ./contracts/Exponential.sol

eaeb864d924628839a93ea71cfbd92d86e2cebf57941dd2da5eddd8d59510f93 ./contracts/ExponentialNoError.sol

e6e3dc1d2535539cd3337243ff77912689d0c78c9b69b7947ee28dad00cc7545 ./contracts/InterestRateModel.sol

a1b883c765e7970e04ad435fe3f410f812843994e1fcefb988bc7f2e7ae71226 ./contracts/Migrations.sol

386308c846ce34d796f4c528d4c14a11385ad24b712a09d4c0478ea1830c5e8a ./contracts/ReentrancyGuard.sol

4b5a8e7d5351ec6a60599bea3cb4399962613c3ca09c3fc30cfafaefc39e6bdb ./contracts/RewardControl.sol

1fdc612a66abe09186a6732adc59177aeea1193b5db6151c2d24709b93d7e780 ./contracts/RewardControlInterface.sol

2c504cfbbd76acf2f433434ac101b01af645662e80baa5a73615d29336d29c95 ./contracts/RewardControlStorage.sol

8bbb7617be4748900f1f26b6a9c8ce939898119ee27921c6464bcfcc74786134 ./contracts/SafeMath.sol

12954b798eb0e4e09bbe3ee47d9d23170dd840d82dc3d9361c5d33ce01271e3d ./contracts/SafeToken.sol

Tests

9061c76faf73cf823bd780d8fe3ece2553d052275c722ba75b9b6791a05463d0 ./contracts/test/LiquidationChecker.sol

3064b2ff34e6a9bb3edc18f55c1ba1aa49f1308d02820007c5739af78f197f1f ./contracts/test/Liquidator.sol

771c5c3bb5c2233b53fd202262c753c3b44965cf25a5bbfcea5b79f550ac3a22 ./contracts/test/PriceOracle.sol

0e1cc1857081cd6dcc3e5bc01afd8dd1ad2e83366efb6a036e847a6a63e00014 ./contracts/test/PriceOracleInterface.sol

f5c87468ca074b7b33001add663cec316fb7c33f7bedda00b690a751c856ad96 ./contracts/test/PriceOracleProxy.sol

81d9322d7e986693ba9400d45a9c20e451e5bbf23378ce5366ba95b79ec8b1f0 ./contracts/test/TestTokens.sol

d7c79987ea8b972656ad7ea17483d56e1d6c0aa67c38a9fcb7c82bd8998f9854 ./test/AddressGenerator.sol

3228b89eb7551a30d7fe89e44a4936d3d3c35affba640f7244303f35d5ca1417 ./test/AlkemiEarnVerifiedGasHarness.sol

7a37736c615936f70a8741bab6ba7f616fba163b24c85fbe5c0d4debc3c1af92 ./test/AlkemiEarnVerifiedGasHarnessTest.js

bf74ada08f811c30edd3c78b3f119452539bdad0dd04b26144b740dc616e0859 ./test/AlkemiEarnVerifiedHarness.sol

75946a1254d933c2ecc093471261b5ebc398e3e124f93200bbb22c4b0a355191 ./test/AlkemiEarnVerifiedHarnessTest.js

f77dd664d1cb026c66a814275b73c18eed7731522d0ba50a9cfe558b582fbef3 ./test/AlkemiEarnVerifiedHarnessTest10.js

248c18019dd38ee6b069d86aa9ad3c41d3ebecc20f2f68f07e3e91bd91fb2d9a ./test/AlkemiEarnVerifiedHarnessTest11.js

63b844d79a04a9a4ce83a36bccc298f1782d242a315a9acc1d4753f3107c914e ./test/AlkemiEarnVerifiedHarnessTest12.js

d4de31e9a85f6a4b57fbbdd902b3a73f000b192123f6798d8491d5ac0e2f18e5 ./test/AlkemiEarnVerifiedHarnessTest13.js

12a9474a229767a59119369fdaf84f6516ad1fd726503e48cb816cbbbaeae7e6 ./test/AlkemiEarnVerifiedHarnessTest14.js

a4e2d2de8e8b41e8df06ae2a4e3af0508992d114a1f9aa843a2f7f74fb9d0344 ./test/AlkemiEarnVerifiedHarnessTest2.js

efe8b436852d3fe9d70d21255bc50c899836502ee76ff157e91be7ef6e3d39fe ./test/AlkemiEarnVerifiedHarnessTest3.js

6091c7267e2b412f92a79a8889440692209744576bcd0a508e01e36f73767090 ./test/AlkemiEarnVerifiedHarnessTest4.js

2de974f292f175927406f1e69e7ad56ce75a3a0faf3c8c4b2f58dcda18f78071 ./test/AlkemiEarnVerifiedHarnessTest5.js

98210164ed7c78d8ca9ae956cfe5e47edc669cc9425c1f0359162aa399da8046 ./test/AlkemiEarnVerifiedHarnessTest6.js

beb051c230942e5e6eacdd4c56742e3550a27b8f8b1a2fef98153868d3fee76e ./test/AlkemiEarnVerifiedHarnessTest7.js

05b56500255ebfe3acbb382f05b9998d591e0dc46d275ace5ac660a0fd7cc497 ./test/AlkemiEarnVerifiedHarnessTest8.js

5b2eef53762c319a9631cdd50d9cea6dfbaaf0644e30e0b8d921bfc16eca71e2 ./test/AlkemiEarnVerifiedHarnessTest9.js

116f342e38b0eadfe3157caca856174cceed1973ac0a586de07d4f6e5c973c82 ./test/AlkemiEarnVerifiedLightHarness.sol

1de4f1da59fb42269952e3751af1f34b93e9153f304cda2889094f030473f249 ./test/AlkemiEarnVerifiedScenario.sol

fa95fba79cebc65de26173db5ed7e489aeef3c5314b900d875c78f8c415319cc ./test/AlkemiEarnVerifiedScenario2.sol

62a8e8f4df81de7b8f499f105a897a1e9b3e518060f9b41c8ecadc4dd0f28527 ./test/AlkemiEarnVerifiedTest.js

4953f03dfe0a1622ddfb6ca773a82ebdcb072e138b7ad9b0a6e8ae6b286378e6 ./test/AssertHelpers.sol

fac8ba9a300d58dba8191f6408b2c0c6a56dc5770ec9e3a5d0a14403c315e6fd ./test/CarefulMathTest.sol

6d7e1575c51041edf3568d4a7ef593f95169703816c991f42b95b639d9431fa8 ./test/Contract.js

4e6f59430986d117da02abd32dff5f5e7ff39e1281de078f847de5f31fcf7dc7 ./test/EIP20Harness.sol

8c66e5070b2ba09245f45b049f83c4344305b4434bfb8503e10daf947afd3e1e ./test/EIP20NonCompliantHarness.sol

de32d8aadab052c78675633ca65f5540a025e2a850909f920bb25780d53ac3e4 ./test/EIP20NonStandardReturnHarness.sol

bd391bd5eec90f7a28759b8c990fc868a3ead1a78fdbe4ce6501e83ee0827cf1 ./test/EIP20NonStandardThrowHarness.sol

57510205f82c7cc16c57e6287c48d169af1121add01952f4fb0c3f1b9ea8fe05 ./test/ErrorReporter.js

c20a4d644ef1b0fa1d9845768451d20aadef14fdf2744d778f506f9d4064c75a ./test/ErrorReporterHarness.sol

fee1a5dc07e8d789daaddd08ecf5b465925dace6364d2cd77694e3e5c9302086 ./test/ErrorReporterHarnessTest.js

b174cbeb599b573329e8a119d661059a6cf5e2b3c8084d020b5acdd333feb16b ./test/ErrorReporterTest.sol

f44858c3c6a1d3a89f7b02ceabee9d95357e34950f5c81465b1b792a9280342e ./test/ExponentialTest.sol

103b23387a78d4974374b7e3cd5abe090ac58c1e8f1cb7e9f08bf9ce739c590d ./test/MathHelpers.sol

61bb51560fba4a493db48cbbc7ff448159c415805b987031a4ebd0bbedf056bb ./test/PriceOracleHarness.sol

f82ea6fdee218c7e99fb3f3dd3d082c8f94d196e3650694c208c2e1730b8af46 ./test/RewardControlHarness.sol

54452d4fc4767ef9f655c3d652b49305f95668ca2b32dbfa9e82a365409c0121 ./test/RewardControlHarnessV2.sol

ec6daf8021ae6dbf75bb7453f432989bee458af46cc82c021f08ff1f73d53ed3 ./test/RewardControlInterfaceTest.js

f047128e4ca32a803132abdcbcc551089c13e145b3f670b132d81b17b8c49f5a ./test/RewardControlInternalTest.js

52f31730b5740f376ddc90dcaa6f47be77e70c0c582ef0f89b4b7e4e02124256 ./test/RewardControlScenario.sol

e5cf437813e27d269e9e6711138ff6aa8a9409eea9e3bcc977f14149714d1cd1 ./test/RewardControlUpgradeTest.js

464cb7c9c0492fd15a854b4df434844445b403bae913e722ee84fd5ccfdbc470 ./test/SafeTokenHarness.sol

c6fbe08a1379e0c64e14a0ff25a23ae327d9e5029fa72406b0e80c4583ede476 ./test/SafeTokenHarnessTest.js

814698d710cb53f4866d67acddaa8045c1714f871c6f13f2d87d20d3de8b33c6 ./test/SafeTokenNonStandardTest.sol

1407ed0683e6caa3b183df551ced5c2c35f43ddc125117ae1dc575b14a01d0a6 ./test/SafeTokenTest.sol

75e87676aadfac707e02d69142c006970d6ea03d6cbce7b877ef0ac4f34ecf5e ./test/Scenario.js

b6a3179d50c8669c9140a6135561ed07c5a2994cb98e4211e769134e4e9f34d9 ./test/ScenarioForRewardDistribution.js

390bcefd75992b793e936a8d8bc888f2bdde14d728fbc2f2caaf8cce809de37b ./test/StableCoinInterestRateModelTest.js

f3726ea9048174adec5bc1b4bc633c199cbb984f9adb1ac845c3281c5753f1e1 ./test/StableCoinInterestRateModelTest.sol

50d548bd2cd366a959ef13e5819241d20a665eba372d76672cd2ad595e0832ce ./test/StandardInterestRateModelTest.js

1b959a09f207871f64cea51531eb7767ff7571022dc4363669da5d3d5759cc18 ./test/StandardInterestRateModelTest.sol

60f462996ea2878ef5ef66c4b29612d1a5ca9f73438170731bca6cb0fc72e53a ./test/testKYC.js

a81dea96478fa12db5d27824419114c4ec461fe5939c7d9d9205364fa911db9a ./test/Utils.js

e893cee972961dd8e569568fba1f01615cc10b782eeae873a0ac91be4d59fdb7 ./test/Web3.js

97519c7e2399db469fd6e87004818595425f5d4672716b0078b9dc5ada8116cd ./test/Tokens/BasicToken.sol

9947a4fe63a768cc5ed7809977ce37044e8b9b98b38506ddfba9704fadbe155b ./test/Tokens/BasicTokenNS.sol

052a17619e9f1724666039b506a066f29aec6f80353e016210f1a9006e54536d ./test/Tokens/ERC20.sol

de764738a91f2d6b0b068ab4af1fa6e27d59a9b600dec9fca57091391563a47b ./test/Tokens/ERC20Basic.sol

122ca0cae2af87255901a618af24fa27e4c565ec36baedba9d57f7927d368dc5 ./test/Tokens/ERC20BasicNS.sol

6a6163145c237f0e46c75976f7dbf10e7ce0f12dc9f1dcafa8ae479a00fbeec3 ./test/Tokens/ERC20NS.sol

bda3b97f5f3a77276fb9abb218e54e412b040e12de674b64c627eb3824cb5fe7 ./test/Tokens/FaucetNonStandardToken.sol

6b6dfc366cf33e7b762188bc03a36dad1b9c6061672fbdad227b6a78bf4a70ea ./test/Tokens/FaucetToken.sol

5deff2031c20b2e2cef1cb6b3ed643bfd9b3d7d5b94703513f0f4cecf8b76c7c ./test/Tokens/NonStandardToken.sol

4720b46cdfa828e2e94fcd79a7a5d1dcb6afc8deee0b3bb77abc8600a8d19677 ./test/Tokens/SafeMath.sol

49a9611d4158f3c35a9d39ebae28954c9e33958b121ad915b501fbfa9123c5cc ./test/Tokens/StandardToken.sol

b12c04edf906e049a99452016461ec6345acceb0a2643ba432830016297b0c8a ./test/Tokens/WrappedEther.sol

603238582f1095be0a6f488ccf543e95679083f50010d0ee87017801e0c5b9df ./test/Scenario/Action.js

e966932860b0775281408ef48257262aac61e8d6cc4bbfd838a96420fbc29ace ./test/Scenario/AlkemiEarnVerified.js

0758c4d06fa55a022bc0dbc046d214986cdf24698b3cc1ba0bd67ebb60cdc2ad ./test/Scenario/Assertion.js

f9abbd13928211a8727950804c2f3d11c184de404ec73404c5a42229725548a1 ./test/Scenario/Event.js

6a57a0b050854e68a96089be3732cf9e0b6474a2d21dadf13520d8622e253f03 ./test/Scenario/World.js

f095003a01c821ee54a75d670ffcdb54164fb656e2d8a48177168f71022fcbde ./test/Oracle/FixedPriceOracle.sol

53871ffeab402951d0adf7e789dcfac1b4983631a2e8f1074643f6e032c1cc43 ./test/InterestRateModel/AlwaysFailInterestRateModel.sol

a773651d3976b138130843f50c0f03b9fcb441c3f9357f1dee435b2e767252a3 ./test/InterestRateModel/FailableInterestRateModel.sol

cd14263a3f17ed31727738b017298ec4332c0b42b47a6a30a25567e9e1d08a62 ./test/InterestRateModel/FixedInterestRateModel.sol

71c02fc162a2de5f80e696d7429e86559b29f52c2186a2ac2a84748498b29d9b ./test/InterestRateModel/SimpleInterestRateModel.sol

b7e029b77662b56c3858668f02c1f9048c7f7bb82818c63c16fe5f1ef18091db ./test/AlkemiEarnVerified/AlkemiEarnVerifiedNonStandardTest.sol

81055f48fbd1b9018317b32bbe6f553c740b899767578f118a7796c01ba0fe42 ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest.sol

9be6adf9759c65e969ab692efeb0694da50c6de192d091d33055fc9ce13ab9b3 ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_AddCollateralMarket.sol

e4fa03dfe048c9e8f73e5369b56c4e7e802f0f40b2424857adefb24714a66205 ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_AssetPrices.sol

d0029b9d901b18b75e35f80e51f1511ebbb7c86203419cfd235cfe973252d746 ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_Basic.sol

398cde95860a0af01394a61b8e0f80dee4120235164360d5ea4e9ffd8cf3c08c ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_Borrow.sol

7aa5114c42435d7be8cd443d2478fa08b42e3f08a664ea2e0567b205fc8efc1c
./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_CalculateAccountLiquidity.sol

334e0f0fc50e8ca0b1e82f34d6345f6c4a8ca13f413121c0a212e4ff267cd34c
./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_CalculateAccountLiquidity2.sol

786f488c63fa31d5f43e433689cf193c46343203f208859fa1f95d087b940b0d
./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_CalculateAccountLiquidity3.sol

7662d8ee111d8bf606c722ab06502c0bcfcb1c046429d62e37c75dec74e1bf26
./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_CalculateAccountLiquidity4.sol

144181e66b9899a579551c78313781bbacbe8fa1cbd3137f6977d144066e7041
./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_CalculateAccountLiquidity5.sol

8f0cb350c32847b477eb7314e72a9a7eb8d3e70e51cc59d2f0acd615010f3278 ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_CalculateAccountValues.sol

5dcc956cc795e9bfabc9fd863dfc7477b628f9b0f1b37e44ed8d73a00a69cffa ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_CalculateAmountSeize.sol

acb4c176d90d91d93b3534b78bfc88161d85e15bb6e80fbbb2855801a2e06fec ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_CalculateAmountSeize2.sol

e89ea0a90c4990b6e9636e4dbf9d827f568ceda6a2b5884a1399d139fba486ae ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_CalculateBalance.sol

7537fa100eb4bb39f30774305d739bb1e7b27e88aa806ff72c3c148c79423cd0
./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_CalculateBorrowAmountWithFee.sol

2467b1111a7b8995ace81233cd30ac043913c8c76e47d832f4213afaab47be61
./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_CalculateDiscountedBorrowDenominatedCollateral.sol

c33b6927b8fcb361a1f6a9fa5204d3bbadb4d9eabb72bbadd0495cf97c9eb38e
./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_CalculateDiscountedBorrowDenominatedCollateral2.sol

f0653501c160484372bc6be207d43912bdb602af4e37b8e0544c926f5accd667
./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_CalculateDiscountedRepayToEvenAmount.sol

250c0ac6e6f340c7bb891b5e924dbbb3509a71b5ca6b60ca9d01c486fd201433
./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_CalculateDiscountedRepayToEvenAmount2.sol

4662c570f7f7932434631ecc783e9e2166a27708b441113d504e86fd97907c0e ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_CalculateInterestIndex.sol

ff38350b08fcabf244254508abe4063b9f329e709e7ce04da75b1eab104216f1 ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_CalculateInterestIndex2.sol

48502dd4a968a9f769dc4cf063da9c18433b68e237ebbed38b5ff8f0926c40e4 ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_GetAssetAmountForValue.sol

50174549b0d234582f7ccb1159723f4b7368b7e714d208b536548dee490f911e ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_GetPriceForAssetAmount.sol

a0a09cb4ffcdf7ecf879ba5b340303636ff03027c4be3f55f12791999f850ffe
./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_GetPriceForAssetAmountMulCollatRatio.sol

ca7fa2e62e79143870844b7e6e4d6f50840e69613b50f7b3285ff62ee68a599d ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_Oracle.sol

ba410fb2755389d9eb489659fa3c95ff9f3da62c25e21e5018dd03446556fd79 ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_Oracle2.sol

061df9957e205bfe0938fa63955f94f16556515a4d69bdc0e188d70721ef9cc9 ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_RepayBorrow.sol

aab1b57ac6d2499911ac71c34281bf7107f58b2e02a7467cb7c9e887da79e102
./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_SetMarketInterestRateModel.sol

0e1aea15d81807991a3e8b79fa790a13d0a3f1b1608a5ea3cf9d5f39aa09c61a ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_SetOriginationFee.sol

5d6a0884d92e4626d241f658f5bcb08757ed942de4639dbd76299c6397d7f374 ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_SetPendingAdmin.sol

0b7111bed1273710c7c9eb8877c65c6e17a6d6b2ee95d89e177987ba5c3b6f0e ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_SetRiskParameters.sol

ef1a884d4fddb319575cee2ef38253c8491d4045f9039c90bf928df5df6374fa ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_SetRiskParameters2.sol

34a184f0f5cc5d5f7f6ea4840a0fb200a4e97a188fc844695bce6dacff8aee58 ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_SetRiskParameters3.sol

020ef406a92cf6ecbfe0c656e6c30aed037edab24a4a3873ba65d1d72f6aa2ed ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_SetRiskParameters4.sol

263d1799f4a7058bf75bab93fc3966194a3ca3484a98832a4d0f81f2e95fb135 ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_Supply.sol

42594009bd98399932e2452d9c2b42181b5047f5bc0b213f20521ee8203ef1b4 ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_Supply_NonStandard.sol

343428f7a32e063323f27d95fd7205d0e1bf92fcf5ccc481f0ae8bfc7b91a3cf ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_SupportMarket.sol

3b44fbcb879795a12d9f558fbbaae44f3a0d4e8b1abbbc491b461308cccc0ec9 ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_SupportMarket2.sol

9f9b86dbc010782428f400a6d748b69f5befec95dba30fc836185ec02656c573 ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_SupportMarket3.sol

74dc460353af3a6206632f30d181f96952f4f88c4bc5aca40569d55ccfc315c7 ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_SupportMarket4.sol

50e6a7ed133bab81b338dae7688fcecdfa7cbb9653a52d904da5949db5b28fbd ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_SuspendMarket.sol

4b30345167dfddc3a0f4722ecd95d48c05777acffa733d94f5a83768597e556b ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_Withdraw.sol

67bffe944dbd85f20eb769be104a3baf020c3654bcf430e01dc9928807a235b6 ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_WithdrawEquity.sol

ef5f94746a287ba1a96907905fa15bf54fce65d3b4d1ffba60025c6ef36b63ad ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_WithdrawEquity2.sol

0d81ca380846a49b2e42c6f09f87efb43e2ef1b063761a299c6055aa0af5ff2a ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_WithdrawEquity3.sol

8e8dbd8d305573b1a9267c8a9466f9d0ae4d8da584f0b57f2e17a8c56032d0cb ./test/AlkemiEarnVerified/AlkemiEarnVerifiedTest_WithdrawEquity4.sol

86542598ed821221a4d8900f7044207102a88128f82a115fa36ac9e5d67a683a ./test/AlkemiEarnVerified/AlkemiEarnVerifiedWithPriceTest.sol

Changelog

- 2021-07-20 - Initial report (Commit `03479f9`)
- 2021-08-31 - Re-audit report (Commit `69a1f54`)

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.