ClarityAlliance

ZEST PROTOCOL v2 (UPGRADE) SECURITY REVIEW

Conducted by:
KRISTIAN APOSTOLOV, ALIN BARBATEI (ABA), SILVEROLOGIST

DECEMBER 3RD, 2025

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

1. About Clarity Alliance

Clarity Alliance is a team of expert whitehat hackers specialising in securing protocols on Stacks.

They have disclosed vulnerabilities that have saved millions in live TVL and conducted thorough reviews for some of the largest projects across the Stacks ecosystem.

Learn more about Clarity Alliance at claritylliance.org.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

2. Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Clarity Alliance to perform a security assessment.

This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Clarity Alliance’s position is that each company and individual are responsible for their own due diligence and continuous security. Clarity Alliance’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree
to analyze.

The assessment services provided by Clarity Alliance are subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis.

Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third parties. Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Clarity Alliance does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

3. Introduction

A time-boxed security review of Zest Protocol, where Clarity Alliance reviewed the scope and provided insights on improving the protocol.

4. About Zest Protocol

Zest Protocol is the DeFi protocol built for Bitcoin. Fully on-chain and open-source, it is building the future of Bitcoin finance.

We've launched Zest Protocol Borrow, enabling users to unlock liquidity by borrowing against their assets.

Live on Stacks—the leading Bitcoin Layer 2—Zest is now the top DeFi protocol on the network. Through the Stacks Market, users can deposit idle assets such as STX, sBTC, stSTX, USDC, and others to earn yield, accumulate points, and access overcollateralized loans..

Zest exists to make Bitcoin productive—every sat of it. The goal is to build a vibrant borrowing and lending ecosystem around Bitcoin as an asset.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

5. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1 Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2 Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3 Action required for severity levels

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

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

6. Security Assessment Summary

Scope

The following contracts, located in the zest-core repository, were in the scope of the security review:

- `dao/dao-multisig.clar`
- `dao/dao-executor.clar`
- `dao/dao-treasury.clar`
- `dao/traits.clar`
- `market/market.clar`
- `market/market-vault.clar`
- `registry/egroup.clar`
- `registry/assets.clar`
- `registry/reserve-calculator.clar`
- `vault/vault-stx.clar`
- `vault/vault-sbtc.clar`
- `vault/vault-ststx.clar`
- `vault/vault-usdc.clar`
- `vault/vault-usdh.clar`
- `vault/traits.clar`

Initial Commit Reviewed:
80f5da77fbcb917958a0e3f64c4bb0e87832492b

Intermediate Commits Reviewed:
496f774576c6e2aa42ee6a634cd6daf94060f0d0

eb99c6f8acf89b6d86ede97173179a8a8b1e25c8

Final Commit After Remediations:
fab7cdf569b4165b2c0bd47fd7ff46717d5e8b43

CONTENTS

ClarityAlliance
Security Review

Zest Protocol v2 Upgrade

7. Executive Summary

Over the course of the security review, Kristian Apostolov, Alin Barbatei (ABA), Silverologist engaged with - to review Zest Protocol. In this period of time a total of **40** issues were uncovered.

Protocol Summary

Protocol Name	Zest Protocol
Date	December 3rd, 2025

Findings Count

Severity	Amount
Critical	1
High	4
Medium	7
Low	7
QA	21
Total Findings	**40**

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

Summary of Findings

ID	Title	Severity	Status
[C-01]	stSTX Vault Cannot Withdraw Tokens	Critical	Resolved
[H-01]	Efficiency Groups Cannot Be Updated	High	Resolved
[H-02]	DAO Implementation Cannot Be Updated	High	Resolved
[H-03]	Disabled Debt Not Accounted For In Notional Debt	High	Resolved
[H-04]	Self-Liquidation Market Draining Attack via Egroup LTV Downgrading	High	Resolved
[M-01]	Positions With An Empty Safe Mask Are Not Fully Supported	Medium	Resolved
[M-02]	Missing Grace Period After Vault Repayment Pause	Medium	Resolved
[M-03]	Lack of Slippage on Liquidations	Medium	Resolved
[M-04]	Ambiguous EGroup Defaulting Logic	Medium	Resolved
[M-05]	Dangerous Market Account Behavior	Medium	Resolved
[M-06]	Inability to Liquidate Positions Using zToken Collateral Within the Same Vault FlashLoan Context	Medium	Resolved
[M-07]	Repay Health Check May Block Insolvent Users From Avoiding a Full Liquidation	Medium	Resolved
[L-01]	Threshold Changes Can Invalidate Pending Executable Proposals	Low	Acknowledged
[L-02]	Vault Names, Symbols, and URI Require Sanitization	Low	Resolved
[L-03]	Market Vault Funds Retrieval Bypasses Clarity 4 Security Enhancements	Low	Resolved
[L-04]	Reducing Collateral Liquidation LTV Ratios May Instantly Liquidate Users	Low	Acknowledged
[L-05]	Significant Absence of Emitted Events	Low	Resolved
[L-06]	Maximum Liquidation Penalty Is Not Capped	Low	Resolved
[L-07]	Avoid Using Unwrap Panic	Low	Resolved

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

Summary of Findings

ID	Title	Severity	Status
[QA-01]	Inline Reserve Calculator Contract	QA	Resolved
[QA-02]	Missing Flash Loan Features	QA	Resolved
[QA-03]	Optimization of Market Asset Retrieval	QA	Resolved
[QA-04]	Eliminate Redundant Contract Caller Authentication in Vault DAO Operations	QA	Resolved
[QA-05]	Function Naming Ambiguities Severely Decrease Code Readability	QA	Resolved
[QA-06]	General Code Style Improvements	QA	Resolved
[QA-07]	Optimization for Enabling and Disabling Assets	QA	Resolved
[QA-08]	Promote Debug Getters in eGroup to Production	QA	Resolved
[QA-09]	Simplify Nonce to a uint to Reduce Complexity	QA	Resolved
[QA-10]	Code Constants Usage Ambiguities	QA	Resolved
[QA-11]	Simplification of Retrieving Liquidation Position	QA	Resolved
[QA-12]	Optimization of Borrower Scaled Debt Retrieval	QA	Resolved
[QA-13]	Improvements Needed for Mask Market Contract Operations	QA	Resolved
[QA-14]	Function check-egroup-invariant Contains Inefficiency and Redundancies	QA	Resolved
[QA-15]	Redundant Parameter Fragment	QA	Resolved
[QA-16]	Enhance Market Contract External Interface	QA	Resolved
[QA-17]	Create Market Trait	QA	Resolved
[QA-18]	Implement a Majority Rule-Based Multisig	QA	Acknowledged
[QA-19]	Integrate Max Staleness into Asset Oracle Data Entry	QA	Resolved
[QA-20]	Remove Unused Market Contract Code Artifacts	QA	Resolved
[QA-21]	Isolate stSTX Price Resolution from resolve-ztoken	QA	Resolved

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

8. Findings

8.1. Critical Findings

[C-01] stSTX Vault Cannot Withdraw Tokens

Description

Clarity 4 introduces significant changes to the `as-contract` logic:

- The `as-contract` has been removed.
- `as-contract?` now exists and imposes full restrictions on all passed tokens by default.

All Zest vaults utilize the `send-underlying` function to initiate the transfer of underlying tokens. This function is generally implemented as follows:

```
(define-private (send-underlying (amt uint) (account principal))
  (begin
    (try! (contract-call? .sbtc transfer amt current-contract account none))
    (ok true)))
```

However, the `vault-stx` is an exception, as it requires permission to pass `amt` tokens of STX.

```
(define-private (send-underlying (amt uint) (account principal))
  (begin
    (try! (as-contract? ((with-stx amt))
      (try! (contract-call? .wstx transfer amt tx-sender account none))
      true))
(ok true)))
```

This is a special case due to the functionality of the wrapped STX contract.

Apart from the `vault-stx`, all other vaults theoretically function correctly because the underlying `SIP-10::transfer` function allows the transfer if the caller is either the `contract-caller` or `tx-sender`.

For example, the sBTC token is implemented as follows:

```
(define-public (transfer (amount uint) (sender principal)
  (recipient principal) (memo (optional (buff 34))))
        (begin
                (asserts! (or (is-eq tx-sender sender)
    (is-eq contract-caller sender)) ERR_NOT_OWNER)
                (try! (ft-transfer? sbtc-token amount sender recipient))
                (match memo to-print (print to-print) 0x)
                (ok true)
        )
)
```

However, not all tokens support both modes of authorization. Only more recent tokens have started to address this, while older tokens

still rely on `tx-sender` for authorization.

Among the implemented vaults, the `vault-ststx` contract, which wraps the stSTX token, is the only vault that does not accept authorization by `contract-caller` :

```
(define-public (transfer (amount uint) (sender principal)
  (recipient principal) (memo (optional (buff 34))))
  (begin
    (asserts! (is-eq tx-sender sender) (err ERR_NOT_AUTHORIZED))

    (match (ft-transfer? ststx amount sender recipient)
      response (begin
        (print memo)
        (
          print{action:"transfer",
          data:{sender:tx-sender,
          recipient:recipient,
          amount:amount,
          block-height:block-height}}
        )
        (ok response)
      )
      error (err error)
    )
  )
)
```

This means that while depositing in the `stSTX` vault is allowed, withdrawing from it will fail, effectively blocking user funds in the vault.

Recommendation

To resolve the issue specifically for the `stSTX` vault, use the `with-ft` keyword to permit the movement of the `stSTX` tokens:

```
(define-private (send-underlying (amt uint) (account principal))
  (begin
    (try! (as-contract? ( (with-ft .ststx "ststx" amt)  )
      (try! (contract-call? .ststx transfer amt tx-sender account none))
      true))
    (ok true)))
```

To avoid concerns about future implementations of underlying token vaults, this logic can be applied to all vaults.

Note #1: The `.ststx` is used locally for testing. In a production environment, the principal should be changed to `SP45ZE494VC2YC5JYG7AYFQ44F5Q4PYV7DVMDPBG.ststx-token`. In the context of the `with-ft` command, the `UNDERLYING` constant can be used (which is currently unused).

Note #2: To ensure this case is covered by tests, modify the local `ststx` utility token contract to behave like the production one:

```
(define-public (transfer (amount uint) (sender principal)
   (recipient principal) (memo (optional (buff 34))))
   (begin
-    (asserts! (or (is-eq tx-sender sender)
- (is-eq contract-caller sender)) err-not-token-owner)
+    (asserts! (is-eq tx-sender sender) err-not-token-owner)
     (ft-transfer? ststx amount sender recipient)))
```

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

8.2. High Findings

[H-01] Efficiency Groups Cannot Be Updated

Description

The current implementation of Efficiency Groups allows any group to be updated using the `egroup::update` function. However, this function contains an incorrectly reversed "no changes" check:

```
;; --- early end if no mask update ---
(asserts! (is-eq prev-MASK new-MASK) (ok true))
```

As it stands, if the previous mask is NOT equal to the new mask (the `(is-eq prev-MASK new-MASK)` condition), the `asserts!` exits with the `(ok true)` error. The logic should be reversed to only pass the assertion check if the previous mask IS equal to the new mask.

Recommendation

If an early exit (without reverting) is still intended, modify the `asserts!` logic to only pass if `prev-MASK` is not equal to `new-MASK`.

Example implementation:

```
;; --- early end if no mask update ---
(asserts! (not (is-eq prev-MASK new-MASK)) (ok true))
```

Additionally, consider actually reverting execution if the masks are equal, instead of allowing a pass-through.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[H-02] DAO Implementation Cannot Be Updated

Description

The DAO executor contract is tasked with executing proposals once they have garnered sufficient approvals from signers. A crucial parameter, `impl`, holds the principal of the DAO multisig contract, which is responsible for proposal creation and signer approvals.

The executor includes a function designed to update the implementation's principal:

```
(define-public (set-impl (new-impl principal))
  (begin
    (try! (IMPL))
    (var-set impl (some new-impl))
    (ok true)))
```

In practice, this function is not callable.

The DAO multisig interacts with the executor solely through `execute-proposal`, which executes proposals within the context of the proposal script rather than directly as the multisig contract.

Since `IMPL` requires the caller to be the current implementation, and `dao-multisig` lacks a method to invoke `set-impl`, the functionality to update the implementation is effectively inaccessible.

Recommendation

To enable the missing functionality, two approaches can be considered:

1. Introduce functionality in the `dao-multisig` contract that calls `dao-executor::set-impl` to update the implementation. This method can incorporate a timelock by implementing a two-step process with a hardcoded, enforced delay.
2. Adjust the authorization of `dao-executor::set-impl` to utilize the same logic as the `DAO` validations found in other parts of the code:

```
(define-private (DAO)
  (begin
    (asserts!
      (is-eq tx-sender .dao-executor)
      ERR-AUTH)
(ok true)))
```

While Clarity does not permit reentrancy within the same function, it does allow reentrancy within the same contract, thus enabling the proposed fix. However, in this scenario, a timelock cannot be enforced.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[H-03] Disabled Debt Not Accounted For In Notional Debt

Description

During the liquidation process in `market::liquidate` , the liquidation context is initially retrieved using `liquidation-context` . This context is then used to compute the notional values of both collateral and debt, which are essential for determining the position's Loan-to-Value (LTV) ratio.

The current logic retrieves context and information for all enabled collateral and debt. However, this approach is flawed because disabling debt is intended only to prevent the addition of new debt in that asset, not to exclude it during health checks or the liquidation of existing loans.

The issue arises from the filtering performed in `user-safe-mask` , where disabled debt assets are omitted. Consequently, disabled debt is excluded from the LTV calculation, which artificially lowers the LTV for positions that include such assets and affects general health checks.

This problem impacts all market operations that depend on accurate LTV values.

Recommendation

Modify the `market::user-safe-mask` function to ensure it does not disabled debt assets.

Example implementation:

```
(define-private (user-safe-mask (mask-user uint) (mask-enabled uint))
  (let ((enabled-collateral (bit-and mask-enabled MAX-U64))
-       (enabled-debt (/ (bit-and mask-enabled DEBT-MASK) (pow u2 DEBT-OFFSET)))
        (user-collateral (bit-and mask-user MAX-U64))
        (user-debt (/ (bit-and mask-user DEBT-MASK) (pow u2 DEBT-OFFSET)))
        (collateral-match (bit-and user-collateral enabled-collateral))
-       (debt-match (bit-and user-debt enabled-debt)))
+       )
-   (bit-or collateral-match debt-match)))
+   (bit-or collateral-match user-debt)))
```

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[H-04] Self-Liquidation Market Draining Attack via Egroup LTV Downgrading

Description

The Zest v2 codebase utilizes an elevation group system where bundles of assets are associated with specific Loan-to-Value (LTV) parameters. More details can be found in the official documentation.

Within this framework, LTV groups are valid only if certain conditions are met regarding borrowing, partial liquidation, and full liquidation. Specifically, a group that is a subset of another must have its borrow, partial liquidation, and liquidation LTVs higher than or equal to those of the superset. This means that the more unique assets a user holds (either as collateral or debt), the less their underlying collateral is valued.

The `market::collateral-add` function adds collateral to a user's position. Since this operation may change the user's LTV group if the collateral is new, a user's overall health status can actually deteriorate.

This situation can occur both naturally for legitimate users and can be exploited in an attack.

Consider the following scenario for a typical user:

- The user has $1,000 in notional value from four collaterals added at a 60% LTV group.
- The user also has debt equivalent to $550 in notional value.
- In this case, if the user accrues $50 more in debt, they will be liquidated.
- To avoid liquidation, the user wants to add some new collateral and adds $50 of a new collateral token to their position.
- By adding new collateral, the user is moved to a 50% LTV group.
- The new notional value is $1,050; however, since the user is now in a 50% LTV group, their position is valued at $525, which is less than their $550 debt.
- The user becomes liquidatable simply by adding more collateral.

In an attack scenario, a threat actor may:

- Deposit one type of collateral.
- Maximize borrowing against it, bringing the liquidation point to the borrow LTV.
- Add multiple small amounts of all other available collaterals, making themselves fully liquidatable.
- Liquidate themselves for all existing collateral.
- Due to the penalty discount, some debt remains unpaid, which must be socialized.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

The attacker profits from the difference generated by the penalty discount after accounting for execution fees, provided that the egroups have sufficiently high differences between LTVs and high enough penalties.

The condition is not only for the user to be fully liquidatable but also for the penalty to be high enough such that the debt plus penalty exceeds the available collateral.

Recommendation

Modify the `collateral-add` function to perform an ending health check only if adding a new collateral asset for that user, ensuring that the new health status is at least the same as the current status. This means the new collateral's total notional value, multiplied by the new LTV evaluation, must be equal to or better than the previous status, regardless of whether the position is healthy or not.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

8.3. Medium Findings

[M-01] Positions With An Empty Safe Mask Are Not Fully SupportedAre Not Fully Supported

Description

The `market::assets` function is responsible for retrieving asset information for a user's assets, including all debt assets and enabled collateral assets. During this process, the `assets::status-multi` function is called, which contains the following logic:

```
(define-read-only (status-multi (ids (list 64 uint)))
 (let ((enabled-mask  (get-bitmap))
       (mask          (uint-to-list-u64 enabled-mask)))
   (map status ids mask)))
```

In this code, `mask` is always a non-empty list. However, if `ids` is an empty list, the `map` function will fail, causing `status-multi` to abort.

Consider the following scenario:

- Alice adds USDC as her only collateral asset.
- Later, USDC is disabled as a collateral asset.
- Alice attempts to withdraw her collateral but fails.

Once USDC is disabled, Alice's safe mask (her user mask with disabled collaterals removed) becomes empty. Consequently, when the system `status-multi` , `ids` will be an empty list, leading to a map error and preventing her from ids her funds.

Recommendation

Modify `status-multi` to explicitly handle cases where `ids` is empty. In such instances, it should return an empty list.

Example implementation:

```
(define-read-only (status-multi (ids (list 64 uint)))
 (let ((enabled-mask  (get-bitmap))
       (mask          (uint-to-list-u64 enabled-mask)))
-   (map status ids mask)))
+   (if (is-eq (len ids) u0) (list ) (map status ids mask))))
```

[M-02] Missing Grace Period After Vault Repayment Pause

Description

Each vault has an independent mechanism to pause repayments, which also implicitly pauses liquidations for that specific vault. This functionality is distinct from the market-wide liquidation pause, which affects all vaults simultaneously.

If repayments are paused for only a subset of vaults, the global liquidation pause is not activated, allowing other vaults to continue with normal liquidation activities. However, when repayments are resumed, there is no grace period applied.

Consequently, if a vault has its repayments paused for a certain period, any positions that become liquidatable during this pause can be immediately liquidated once repayments are unpaused, without giving users a chance to repay first. This behavior can lead to unfair losses for the holders of affected positions.

Recommendation

Enable the DAO to set a liquidation grace period for vaults. Specifically:

- Allow the DAO to establish a vault liquidation grace period within the `market` contract, necessitating a new function.
- This grace period should be applied on a per-vault basis and affect the `market::liquidate` function.
- Transform the `liquidation-grace-end` variable into a map with vault-id → grace end, while using a special ID for the market contract itself, e.g., 100.
- Extend the `market::is-liquidation-paused` function to also accept the debt asset ID as an argument and perform two checks:
  - Check if there is an end time entry for the new `liquidation-grace-end` map for the market itself, e.g., key 100, to verify a global repay pause.
  - Check if there is an entry for the asset-id in the new map.

This approach ensures that for each liquidation, it is verified whether either the global liquidation or the specific vault is in a grace period.

The proposed feature should be included in a proposal where the logic first unpauses the repayment for a specific vault and then marks the market contract to indicate that the vault now has a liquidation grace period.

While there is no on-chain enforcement to ensure that vault repayment must be linked to a market liquidation grace, there is no simpler way to implement this feature. Adding a grace period within the vault itself on `system-repay` would limit both liquidations and repayments, forcing users to add collateral if they wish to save their position.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[M-03] Lack of Slippage on Liquidations

Description

When a liquidation is initiated, the liquidator calls the `market::liquidate` or `market::liquidate-multi` function with the desired collateral token to be liquidated and specifies the maximum debt repayment amount they are willing to offer for the discounted collateral.

The amount of collateral (at a discount) a liquidator would receive and the debt they must repay is determined by the liquidation penalty factor, which depends on the LTV group, as well as the user's health status (partial versus full liquidation).

Due to the implementation of the liquidation system, a liquidator may call `market::liquidate` expecting a specific profit after liquidation but may receive less due to several factors:

- The user, either intentionally or unintentionally, has front-run the liquidation by adding more collateral or repaying to reduce the amount they are liquidated for.
- A price update has occurred in the same block before the liquidator's call, and the new prices are unfavorable.
- A separate user's debt asset was liquidated, moving the user to a different LTV group where the liquidation penalty discount is less favorable than the original one.

In all these scenarios, the liquidator receives less value than expected. In certain cases, the liquidator might not have initiated the liquidation if they had up-to-date state information, due to a lack of profitability.

Recommendation

Allow liquidators to specify a `minimum-collateral-received` amount when calling `liquidate`. If the resulting `coll-final` from the `liquidate` call is not at least equal to this amount, then revert the liquidation. If liquidators do not wish to use this option, they can simply set it to 0.

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[M-04] Ambiguous EGroup Defaulting Logic

Description

The Zest v2 codebase utilizes an efficiency group system where bundles of assets are linked to specific LTV parameters. More details can be found in the official documentation.

A user's position is associated with one of these maps/groups if it is a subset of it (closest match, see a walkthrough here). If no group is found, the default EGroup is used.

The default group values are:

```
;; default
(define-constant DEFAULT-MASK-ID u255)

(map-insert registry (uint-to-buff1 DEFAULT-MASK-ID)
                          {
                            id: (uint-to-buff1 DEFAULT-MASK-ID),
                            MASK: MAX-U128,
                            LIQ-CURVE-EXP: (uint-to-buff2 u10000),
                            LIQ-PENALTY-MIN: (uint-to-buff2 u100),
                            LIQ-PENALTY-MAX: (uint-to-buff2 u1000),
                            LTV-BORROW: (uint-to-buff2 u0),
                            LTV-LIQ-PARTIAL: (uint-to-buff2 u0),
                            LTV-LIQ-FULL: (uint-to-buff2 u0)
                          })
```

The current default values were chosen to prevent normal execution flow with them. However, this behavior is not consistently enforced, leading to unusual situations:

1. **Adding Collateral Works with Any Group**

   The current `market::add-collateral` function does not check any group logic, allowing users to add any collateral token as long as it is an approved collateral asset.

2. **Removing Collateral Works in Any Group if No Debt**

   Removing collateral via the `market::collateral-remove` function does not revert for any group if the health check passes. For the default group, the health check can only pass if the user has no debt.

3. **Repaying is Allowed in Some Cases**

   When repaying a position via the `repay` function, if the full debt amount associated with a token is not repaid, no health check is performed, and repayment is allowed. If the full debt is repaid, a health check is conducted with the new LTV group. If the new user mask still has no superset and the user has other debt, the position reverts due to low health. If this was the last user debt, the repayment is successful. If the new user mask has a superset, normal LTV health logic is applied, and the position may or may

not be healthy.

4. Borrowing Cannot Be Done Within the Default Group

The `borrow` function performs two health checks. The first check is for the existing user group, which defaults to group 0, setting all collateral to no value. This check can be passed if the user has no debt. However, the second check, done with the updated mask, keeps the user in the default group, causing a revert due to the newly added debt.

5. Liquidations Cannot Be Done Within the Default Group

Liquidating a position fails because, during liquidation, the protocol calls `calc-liq-factor`:

```
(define-private (calc-liq-factor (ltv-curr uint) (ltv-liq-partial uint)
  (ltv-liq-full uint))
  (min BPS (div-bps-down (- ltv-curr ltv-liq-partial)
    (- ltv-liq-f ull ltv-liq-partial))))
```

If both `ltv-liq-full` and `ltv-liq-partial` are 0, this calculation reverts due to division by zero.

The current implementation of the `egroup` contract allows any group to be modified through the `egroup::update` function. A potential side effect of changing a group's mask is that any existing user positions, which initially mapped to the old mask (as a subset), will now use the default group values.

Considering the above, several issues or odd cases arise from the current egroup defaulting mechanism:

A. If the team does not ensure that all potential user position masks are covered by existing egroup masks, users will default to the default group.

B. If the team updates the mask of an existing egroup without considering existing user positions, users with active loans will use the default egroup.

C. The default group is inconsistent: adding collateral is permitted, removing collateral is permitted if no debt, borrowing fails intentionally, but liquidations fail coincidentally due to division by zero.

D. A user defaulted to the EGroup may have a chance to exit the system if the conditions elaborated in repaying, point `(3)`, are met.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

Recommendation

Most issues can be resolved by ensuring a uniform default group behavior that disallows any entry point operations if the user is or enters the default group. Instead of returning a default group, the `egroup::resolve` function should revert if no group is found. Using this new function, ensure all functions that modify the user mask check that they have not reached a default group or revert.

The only issue not covered is `(B)`, where the team does not ensure all existing user positions are covered when updating an existing mask. This is not something that can easily be implemented on-chain. However, with the other mentioned changes, users in this group would be as if they were paused (as no operations would be allowed).

In this situation, the team would need to quickly add an egroup mask to map these users. By doing so, the issue will be resolved without any problem. However, if this operation takes too long, users might become liquidatable. To mitigate this, the team can only pause liquidations per vaults/global, not per egroup. The team can manage the crisis by pausing liquidations for a short interval while they donate funds to affected parties to repay part of their debt. Alternatively, if repaying is modified to work on behalf of others, they can directly repay part of the targeted user debt.

Adding an on-chain mechanism here would not benefit anyone. Similar to how the `Reducing Collateral Liquidation LTV Ratios May Instantly Liquidate Users` issue can be avoided by incremental team changes, this issue can be better mitigated off-chain or using other mechanisms rather than implementing restrictions on egroup mask updates.

To conclude, our recommendation is to:

- Remove the default egroup and ensure all market operations revert if a user position (old and new) does not map to any egroup.
- Ensure no egroup mask updates leave any users in the default egroup.
- Prepare a contingency strategy for cases where, by mistake, an egroup update throws users into no group.

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[M-05] Dangerous Market Account Behavior

Description

The current Zest `market` contract includes entry points for all relevant market operations. All non-liquidation functions allow an `account` principal:

```
(define-public (collateral-add (ft <ft-trait>) (amount uint)
  (account principal))
(define-public (collateral-remove (ft <ft-trait>) (amount uint)
  (account principal))
(define-public (borrow (ft <ft-trait>) (amount uint) (account principal))
(define-public (repay (ft <ft-trait>) (amount uint) (account principal))
```

In each of these cases, the account must be either the `tx-sender` or `contract-caller`. This is enforced through checks:

```
(asserts! (or (is-eq account tx-sender)
  (is-eq account contract-caller)) ERR-AUTH)
```

However, the underlying funds logic or accounting only utilizes `tx-sender`. Here's a detailed explanation for each situation:

1. Adding Collateral

When adding collateral via `market::collateral-add`, the `market-vault::collateral-add` function is called with the `account`, leading to `market-vault::receive-tokens`.

```
(define-public (collateral-add (account principal) (amount uint)
  (ft <ft-trait>) (asset-id uint))
  ;; ...
  (receive-tokens ft amount account)
```

In `receive-tokens`, the `asset::transfer` call can only succeed if the account is `tx-sender`, as per SIP-10 transfer logic.

```
(define-private (receive-tokens (asset <ft-trait>) (amount uint)
  (account principal))
  (unwrap-panic
    (contract-call? asset transfer amount account current-contract none)))
```

It cannot function with `contract-caller` since that value is set to the `market` contract, which called the `market-vault` itself. This means the `tx-sender` can decide whether to add collateral to the `contract-caller` or themselves.

2. Removing Collateral

When removing collateral via `collateral-remove`, the collateral accounting is done concerning the passed `account`, and the `account` is the one receiving the tokens. However, this allows any downstream `contract-caller`(s) that do not change `tx-sender` to remove

collateral to a point where the position is barely healthy. At that point, the account can be easily liquidated, even if it still has its collateral.

3. Borrowing Assets

The `market::borrow` function calls `vault-*::system-borrow` to obtain the borrowed funds, but this function implicitly sends the funds to the `tx-sender`. However, it adds the debt to the passed account.

This means the `tx-sender` can receive the funds while the `contract-caller` assumes the debt.

4. Repaying Debt

Repaying the debt via `market::repay` presents another odd situation. To repay debt, the `vault-*::system-repay` function is called, which takes the underlying tokens from the `tx-sender`. While the funds are always taken from the `tx-sender`, the `account` is marked as having the debt paid.

This allows a situation where the `tx-sender` pays, but the `contract-caller` has the debt reduced.

5. Liquidations

As liquidations are currently implemented, the `tx-sender` repays the debt and receives the collateral. This could be better changed to the `contract-caller`.

All these situations arise due to ambiguity regarding the allowed caller versus benefactor versus payer.

Recommendation

For removing collateral, modify the `collateral-remove` function as follows:

- Ensure collateral removal only works for the `contract-caller`, removing the `account` parameter from the function prototype and adding it locally in the `let` declaration as `(account contract-caller)`.
- The function now sends the removed collateral of the `contract-caller` to itself, but integrators may wish to have an optional receiver, defaulting to the contract-caller if not specified.

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

```
-(define-public (collateral-remove (ft <ft-trait>) (amount uint)
- (account principal))
+(define-public (collateral-remove (ft <ft-trait>) (amount uint) (receiver
+ (optional principal)))
   (let ((ft-address (contract-of ft))
         (asset (get-asset ft-address))
         (asset-id (get id asset))
+        (account contract-caller)
+        (collateral-receiver (match receiver recv recv contract-caller))
         (is-collateral-enabled (get collateral asset))

         ;; Step 1: Get position WITHOUT resolving prices
@@ -765,7 +767,6 @@
         ;; post-removal calculation
         (removed-asset-value
           (find-and-resolve-asset-value assets asset-id amount true)))

-    (asserts! (or (is-eq account tx-sender)
- (is-eq account contract-caller)) ERR-AUTH)
     (asserts! (> amount u0) ERR-AMOUNT-ZERO)
     (asserts!
       (is-healthy collateral-value debt-value current-ltvb) ERR-UNHEALTHY)

@@ -800,15 +801,16 @@
                     amount
                     ft
                     asset-id
-                    account)))
+                    collateral-receiver)))
```

For `borrowing`, if no changes to the vaults are allowed, remove the `account` completely from the function prototype and set it locally as `tx-sender`. However, this is insufficient as the `contract-caller` can initiate a borrow for the `tx-sender`, so we also need to enforce that `tx-sender` is equal to `contract-caller`.

```
-(define-public (borrow (ft <ft-trait>) (amount uint) (account principal))
+(define-public (borrow (ft <ft-trait>) (amount uint))
   (let ((address (contract-of ft))
         (asset (get-asset address))
         (asset-id (get id asset))
+        (account tx-sender)

         ;; Step 1: Get position WITHOUT resolving prices
         (position (get-position account))
@@ -834,7 +836,7 @@
         (debt-value (get debt notional-valued-assets)))

     ;; preconditions
-    (asserts! (or (is-eq account tx-sender)
- (is-eq account contract-caller)) ERR-AUTH)
+    (asserts! (is-eq contract-caller tx-sender) ERR-AUTH)
```

This constraint of `(is-eq contract-caller tx-sender)` is required because We can't change who receives the funds; implicitly, the funds receiver must be the one who incurs the debt.

If modifying the vault interface is allowed, set the borrower as the `contract-caller` and reuse the recipient logic from `collateral-remove`. The `vault:system-borrow` function would then require changes to allow passing a `recipient` principal. The `market::vault-system-borrow` function would subsequently need changes to accommodate this new parameter.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

When repaying debt via the `repay` function, to avoid having the `tx-sender` as a hard requirement, the funds first need to be transferred to the `market` contract, then within an `as-contract` call, transfer them to the vault. This is necessary as the vaults themselves pass execution to the SIP-10 underlying assets, which use the `tx-sender` for authorization. While further changing the vault may be possible, some vaults may use older tokens which only allow for `tx-sender` authorization, such as the `vault-ststsx`. Meaning the `vault::system-repay` behavior should use, as it is using now, for underlying transfer authorization.

Thus, if we want to have `contract-caller` as the payer, we need to modify the `vault::system-repay` function to accept a `from` parameter, where if we are not transferring the funds from the `tx-sender` (which accounts for the case where the `contract-caller` is also the `tx-sender`) we first need to move the funds to the market contract and then to the vault.

Example implementation for `vault::system-repay`:

```
-(define-private (vault-system-repay (aid uint) (amount uint))
+(define-private (vault-system-repay (asset-id uint) (amount uint)
+ (from principal) (ft <ft-trait>) (ft-address principal))
+  (begin
+    (if (is-eg from tx-sender)
+       (call-system-repay asset-id amount)
+       (begin
+         ;; transfer amount from the "from" principal to the current contract
+         is this allows for contract-caller type authorization on tokens
+         ;; stSTX and wSTX repayments will require contract-caller == tx-sender
+         ;; otherwise this transfer will revert
+         (try! (contract-call? ft transfer amount from current-contract none))
+
+         (if (is-eq ft-address ZEST-STX-WRAPPER-CONTRACT)
+           (as-contract? (with-stx amount))
+             (try! (call-system-repay asset-id amount)))
+           (as-contract? ((with-ft ft-address "*" amount))
+             (try! (call-system-repay asset-id amount))))))))
+`
+(define-private (call-system-repay (aid uint) (amount uint))
   (if (is-eq aid STX (contract-call? .vault-st system-repay amount)
   (if (is-eq aid sBTC) (contract-call? .vault-stc system-repay amount)
   (if (is-eg aid stSTX) (contract-call? vault-stst system-repay amount)
```

Note #1, transferring funds from the `market` contract to the vaults requires the `as-contract?` and implicit restrictions, thus we need to separate in between `with-stx` and `with-ft` branches, since `wSTX` does not work with `with-ft` logic, due to a moving underlying `STX` directly.

And the `repay` implementation itself would be something similar to:

```
-(define-public (repay (ft <ft-trait>) (amount uint) (account principal))
+(define-public (repay (ft <ft-trait>) (amount uint) (on-behalf-of
+ (optional principal)))
   (let ((address (contract-of ft))
         (asset (get-asset address))
         (asset-id (get id asset))
+        ;; defaults to payer (contract-caller) if not specified
+        (account (match on-behalf-of behalf behalf contract-caller))

         ;; Step 1: Get position WITHOUT resolving prices
         (position (get-position account))
@@ -880,15 +913,11 @@
         ;; Check if repaying ALL debt for this asset
         (repaying-all (is-eq repaid-scaled-debt account-scaled-debt)))

-    (asserts! (or (is-eq account tx-sender)
- (is-eq account contract-caller)) ERR-AUTH)
-
     ;; preconditions
     (asserts! (> amount u0) ERR-AMOUNT-ZERO)
     (asserts! (> repaid-scaled-debt u0) ERR-INSUFFICIENT-SCALED-DEBT)

-    ;; repay
-    (try! (vault-system-repay asset-id amount-to-repay))
-
+    (try!
+ (vault-system-repay asset-id amount-to-repay contract-caller ft address))
      ;; update
```

Note #2: repaying debt on behalf of someone has the effect of potentially removing a bit from the user mask (if all debt is paid) which sends them into a different egroup. E groups have an on-chain enforced invariant, that group subsets must have a higher or equal LTV value, which means that, repaying a position cannot reduce a users health. There is also an extra health check, out of abundance of caution regardless. Implicitly there are no abusable scenarios when repaying on behalf of someone.

Regarding the `collateral-add` function. The same changes are needed to bypass the `tx-sender` payer restriction that were applied to the `repay` function. Meaning to move funds to the market contract in case the `tx-sender` is not the `contract-caller`.

In this situation, the `tx-sender` is forced to be the source of the collateral due to how the `market` contract passes to the `market-vault` the transfer logic authentication. As such, the `(is-eq contract-caller tx-sender)` check is a requirement, to ensure the `contract-caller` is not initiating collateral adding.

Example `collateral-add` implementation:

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

```
(define-public (collateral-add (ft <ft-trait>) (amount uint))
  (let ((ft-address (contract-of ft))
        (asset (get-asset ft-address))
        (asset-id (get id asset))
        (account contract-caller))

    (asserts! (get collateral asset) ERR-COLLATERAL-DISABLED)

    (if (is-eg account tx-sender)
        (contract-call? market-vault collateral-add account amount ft asset-id
        (begin
          ;; transfer amount from the "from" principal to the current contract
          ;; this allows for contract-caller type authorization on tokens
          ;; stsTX and wSTX repayments will require contract-caller == tx-sender
          ;; otherwise this transfer will revert
          (try! (contract-call? ft transfer amount account current-contract none))
          (if (is-eq ft-address ZEST-STX-WRAPPER-CONTRACT)
            (as-contract? ((with-stx amount))
              (try!
                (contract-call? .market-vault collateral-add account amount ft asset-id)))
            (as-contract? ((with-ft ft-address "*" amount))
              (try!
                 (contract-call? .market-vault collateral-add account amount ft asset-id))))))
    )
  )
```

Note #3: adding collateral on behalf of someone can introduce health issues, if the donator adds dust, just enough to add the user to higher Egroups, which implicitly have a lower LTV ratio, and may put the user in bad health.

Note #4: there is a lack of health check here that should be done regardless of implementing on-behalf-of, however, if repaying debt is allowed on behalf of someone (which has less potential attack surface), also adding this feature to `collateral-add` is superfluous.

The last point to discuss are liquidations. Liquidations had no ambiguities since `tx-sender` was both the payer and receiver of funds. Adding an optional receiver of funds may be cumbersome, but doable if needed.

However, to maintain consistency and with the new modified `vault-system-repay` function, the `liquidate` function also needs to be slightly modified. If we do apply the same changes and have liquidations done by the `contract-caller` then the changes are needed.

Example modifications to support `contract-caller` payer and receiver.

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

```
@@ -1128,6 +1157,7 @@
                (debt-amount uint)
                (min-collateral-expected uint))
   (let (
+    (liquidator contract-caller)
     (context (get-liquidation-context borrower))
     (position (get position context))
     (pos-full (get-full-position borrower))
@@ -1216,7 +1246,7 @@
     (asserts! (>= coll-final min-collateral-expected) ERR-SLIPPAGE)
     ;; execute liquidation
-    (try! (vault-system-repay debt-aid debt-to-repay))
+    (try!
+ (vault-system-repay debt-aid debt-to-repay liquidator debt-ft debt-address))
     ;; update obligations and socialize bad debt
     (let ((debt-updated (try! (contract-call? .market-vault
@@ -1230,7 +1260,7 @@
                               coll-final
                               collateral-ft
                               coll-aid
-                              tx-sender)))
+                              liquidator)))
           (no-collateral-left (and
```

Note #5: if the liquidator is still intended to be the `tx-sender`, then having the `liquidate` variable set to `tx-sender` is enough.

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[M-06] Inability to Liquidate Positions Using zToken Collateral Within the Same Vault FlashLoan Context

Description

When a flash loan is taken from a vault, a specific reentrancy flag, `in-flashloan`, is set to prevent users from depositing back into the vault.

This flag is specifically checked in the following scenarios:

- `vault::transfer`
- `vault::deposit`
- `vault::redeem`
- `vault::flashloan`

Among these scenarios, by blocking vault share transfers, operations such as using a flash loan of the underlying asset to liquidate a user and requesting the vault LP as collateral cannot be performed.

This is a specific use case; however, the team has expressed interest in this functionality.

Recommendation

Remove the `in-flashloan` check from the `vault::transfer` function in all vaults.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[M-07] Repay Health Check May Block Insolvent Users From Avoiding a Full Liquidation

Description

The Zest v2 codebase utilizes an elevation group system where bundles of assets are linked to specific Loan-to-Value (LTV) parameters. More details can be found in the official documentation.

Within this framework, the LTV groups are valid only if certain conditions are met concerning borrowing, partial liquidation, and full liquidation. Specifically, a group that is a subset of another must have its borrow, partial liquidation, and liquidation LTVs equal to or higher than those of the superset. This implies that the more unique assets a user holds (either as collateral or debt), the less their underlying collateral is valued.

In the `market::repay` function, a health check is performed at the end of the function when repaying the full debt asset.

This leads to a situation where any underwater position with two or more debt assets cannot fully repay any debt associated with one asset if the resulting position remains underwater.

In practice, users at risk of full liquidation cannot simply repay all debt associated with one asset if they remain unhealthy after the repayment. To protect themselves in this scenario, they would need to repay slightly less than their full debt on each asset individually to avoid triggering the health check.

This results in a poor user experience, although it only affects unhealthy positions where repayments still leave them unhealthy. In extreme cases, if users or third-party integrators are unaware of this, they may end up liquidated if this behavior is not clearly communicated.

While the health check during debt repayment has some merit, in practice, due to the setup of the egroup invariant, the likelihood of a healthy position becoming unhealthy after repayment is minimal. This scenario is theoretically possible only if a user belongs to an egroup whose mask was altered, and a new egroup with a completely separate mask (not a subset or superset of any existing masks) is added, with a lower LTV than the original group. For this to occur, the Zest protocol team would need to introduce it mistakenly.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

Recommendation

We propose two options:

1. Modify the `repay` function to perform the health check only when fully repaying an asset debt and only if the current position was healthy. This means skipping the ending health check if the position was not healthy before repayment, as it blocks partial repayments.

2. Remove the health check altogether, since realistically, due to the on- chain egroup invariant, this scenario is unlikely to occur in practice.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

8.4. Low Findings

[L-01] Threshold Changes Can Invalidate Pending Executable Proposals

Description

The `dao-multisig` contract maintains a list of signers who can create and approve proposals. All proposals and their associated data are stored in the `proposals` map.

A proposal becomes executable when, among other conditions, the number of approvals meets or exceeds the threshold value. Each proposal also includes an expiration timestamp.

Consider the following scenario:

- A proposal is created with an expiration timestamp `ET`, and the current `threshold = 2`
- The proposal receives its second approval shortly before `ET`.
- Before the proposal is executed, the DAO increases the approval threshold.

Although the proposal was executable after the second approval, it becomes non-executable following the threshold change. Since this update occurs close to the expiration time, there may not be enough time for the additional required signer(s) to approve, effectively blocking execution.

This creates a timing-dependent inconsistency where proposals can become invalid due to configuration updates that occur between approval and execution.

Recommendation

One possible solution is to include a `threshold` field within the `proposals` map to store the threshold value at the time of proposal creation. When verifying execution conditions, use the minimum of the stored threshold and the current threshold. However, this has implications and should be correlated with the number of signers at that time.

Ultimately, a fool-proof solution would imply substantial overhead and is, objectively, not worth the benefit it adds. Thus, we recommend acknowledging this finding.

[L-02] Vault Names, Symbols, and URI Require Sanitization

Description

The current vault implementations comply with SIP-10 and, as such, have distinct names, symbols, and token URIs.

All of these are constants and are as follows:

Contract	Name	Symbol	URI
vault-sbtc	zest sbtc	zsbtc	none
vault-ststx	zest ststx	zststx	none
vault-stx	Zest STX	zSTX	none
vault-usdc	zest usdc	zusdc	none
vault-usdh	zest usdh	zusdh	none

It is evident that, except for `vault-stx`, all others are in lowercase. This inconsistent formatting is not standardized and may negatively impact third-party UI elements. Additionally, all vaults lack the capability to modify the URI, which could be beneficial if customization is ever required.

Recommendation

Adjust the names and symbols of the mentioned vaults to adhere to more formal standards.

Consider enabling the URI to be modified through a `DAO`-gated function.

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[L-03] Market Vault Funds Retrieval Bypasses Clarity 4 Security Enhancements

Description

Clarity 4 introduces significant changes to the `as-contract` logic:

- The `as-contract` has been removed.
- `as-contract?` now exists with default full restrictions on all passed tokens.

The Zest v2 market vault utilizes the `send-tokens` function to transfer stored tokens to the caller.

```
(define-private (send-tokens (asset <ft-trait>) (amount uint)
  (account principal))
  (unwrap-panic
    (as-contract? ((with-all-assets-unsafe))
      (unwrap-panic
        (contract-call? asset transfer amount tx-sender account none)))))
```

Although the specific asset to be transferred is known during the call, the `as-contract?` logic is invoked with a `with-all-assets-unsafe` allowance, permitting all transfers.

The new `as-contract?` behavior would block all transfers of any other assets if used correctly. However, in this scenario, if a malicious asset is ever introduced, invoking the `SIP-10::transfer` on it within the `market-vault::send-tokens` context would allow the complete draining of all tokens in the vault.

Note that adding a malicious asset token would require several critical system compromises.

Recommendation

To better protect user funds, modify the `market-vault::send-tokens` allowance expression to specifically allow transfers of the specified token.

This should typically involve modifying the following line:

```
-    (as-contract? ((with-all-assets-unsafe))
+    (as-contract? ((with-ft (contract-of asset) "*" amount))
```

However, since `STX` (wrapped) is one such token, a particular issue arises with some existing `STX` wrappers, which actually move `STX` and lack a backing fungible token. To address this, the code would also require a `(with-stx amount)` , but this would leave a vulnerability for extracting stx the event of a malicious token hack (similar to the ALEX hack).

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

The purpose of using an `STX` wrapper token is to enable its use in the context of other fungible tokens without requiring special code. This was achieved until now and with Clarity 4. With the `with-stx` differentiation, the simple wrapper version, which just wraps `stx-transfer?` commands, cannot be used without special code.

Therefore, we recommend creating an `STX` wrapper that functions as a normal FT token, where users would need to call `wrap` to convert their `STX` to `wSTX` and `unwrap` to convert `wSTX` to `STX` in a 1:1 ratio. The wrapper should also have no permissioned role, no way to extract funds, be as simple as possible, and allow for incorrectly transferred `STX` (instead of wrapped) to be accounted for.

With this new version of the `STX` wrapper, all inline code logic that used `with-stx` (including in the `vault-stx` vault) can be removed in favor of normal fungible token transfer authorization.

Thus, the recommendation is to:

- Create an `STX` wrapper that stores `STX` and wraps/unwraps as needed.
- Use this wrapper in the codebase.
- By doing this, the `(as-contract? ((with-ft (contract-of asset) "*" amount))` version in the `market-vault::send-tokens` can be retained, and in the `vault-stx`, the `as-contract` from the `send-underlying` function can be removed completely, as the underlying would act like any other normal SIP-10 token, allowing `contract-caller` as authorization.

Another option is to specifically check if the asset contract is the Zest STX wrapper contract and differentiate behavior accordingly.

Example `market-vault` implementation:

CONTENTS


```
(define-constant PRECISION u100000000)
 (define-constant BPS u10000)
+(define-constant ZEST-STX-WRAPPER-CONTRACT .wstx)

 ;; pack utilities - inlined to avoid contract call overhead
@@ -291,9 +292,12 @@
   (unwrap-panic
     (contract-call? asset transfer amount account current-contract none)))

(define-private (send-tokens (asset <ft-trait>) (amount uint)
   (account principal))
-  (unwrap-panic
-    (as-contract? ((with-all-assets-unsafe))
-      (unwrap-panic
- (contract-call? asset transfer amount tx-sender account none)))))
+  (let ((asset-contract (contract-of asset)))
+    (if (is-eq asset-contract ZEST-STX-WRAPPER-CONTRACT)
+      (as-contract? ((with-stx amount))
+           (try! (contract-call? asset transfer amount tx-sender account none)))
+      (as-contract? ((with-ft asset-contract "*" amount))
+          (try!
+ (contract-call? asset transfer amount tx-sender account))))))

 (define-private (refresh
   (mask uint)) { mask: mask, last-update: stacks-block-time })

@@ -332,7 +336,7 @@
     (asserts! (> amount u0) ERR-AMOUNT-ZERO)

     (insert updated-entry)
-    (send-tokens ft amount recipient)
+    (try! (send-tokens ft amount recipient))
     (ok remaining)))
```

Note: The actual address must be set in the `ZEST-STX-WRAPPER-CONTRACT` and be the same as used in the `vault-stx` contract.

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[L-04] Reducing Collateral Liquidation LTV Ratios May Instantly Liquidate Users

Description

The DAO has the ability to modify the supported collateral Loan-to-Value (LTV) ratios by invoking the `egroup::update` function.

There are two LTVs stored for liquidation purposes: partial and full liquidation LTVs. These thresholds determine when users begin to face liquidation.

If a token is deemed unsuitable as collateral from a market or economic standpoint, or if the current LTVs are considered excessively high and need reduction, governance can call `egroup::update` with more appropriate liquidation LTVs.

However, reducing these LTVs can immediately decrease the collateral value of all existing borrowing positions secured by it, potentially leading to instant liquidation of users.

Recommendation

An on-chain solution would involve modifying the `egroup::update` function to include an LTV ramp duration when adjusting the liquidation LTVs (both partial and full). This ramp would represent a linear decrease from the time of the update to the desired values over the ramp period. If no ramp duration is specified, the change would be immediate. The ramp should only impact liquidations.

By implementing this approach, users would still reach the liquidation point, but not instantaneously, providing them a fair opportunity to unwind their positions. This method has been adopted by some projects over time.

However, incorporating this mechanism into the existing system would introduce significant overhead. Therefore, we recommend acknowledging this issue and, when reducing liquidation LTVs, to do so gradually (e.g.,reducing by 1% every 24 hours until a 10% intended reduction is achieved).

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[L-05] Significant Absence of Emitted Events

Description

The entire codebase is completely devoid of any `print` statements.

This greatly restricts off-chain monitoring and integration capabilities.

Recommendation

Incorporate print events into all public functions and entry points within the codebase.

A standardized print/event structure can be implemented to facilitate off-chain processing. An example of such a structure is:

```
(print {
  action: "<function-name or action>",
  caller: <caller>,
  data: {
    <key1>: <value1>,
    <key2>: <value2>,
    ...
    <keyN>: <valueN>
  }
})
```

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[L-06] Maximum Liquidation Penalty Is Not Capped

Description

When setting an elevation group, the maximum liquidation penalty bounds are specified. These bounds are defined within `[LIQ-PENALTY-MIN, LIQ-PENALTY-MAX]` and are measured in basis points.

Although there is a validation ensuring that the minimum penalty is less than the maximum (`< LIQ-PENALTY-MIN LIQ-PENALTY-MAX)`) there is no validation to ensure that the maximum penalty itself is less than 100%.

A liquidation penalty exceeding 100% should not be permitted.

Recommendation

In the `egroup::SERIALIZE-LEGAL` function, include an additional check to ensure that `LIQ-PENALTY-MAX` is less than `BPS` .

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[L-07] Avoid Using Unwrap Panic

Description

The codebase contains several instances where `unwrap-panic` is utilized.

The use of `unwrap-panic` is discouraged because it complicates debugging in the event of an error and makes it more challenging for external integrators to work with the code. A panic revert would terminate transactions, preventing third parties from handling specific error codes.

Recommendation

Whenever possible, replace `unwrap-panic` with `unwrap!` and include a separate error code.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

8.5. QA Findings

[QA-01] Inline Reserve Calculator Contract

Description

The `reserve-calculator` contract includes functionality that is invoked multiple times by the vault contracts. In Clarity, calling external contracts leads to a notable increase in read count, which significantly raises the block cost overhead.

Recommendation

Wherever possible, inline operations from the `reserve-calculator` contract.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[QA-02] Missing Flash Loan Features

Description

The protocol documentation describes flash loan functionality that has not been implemented:

- A whitelist for callers
- A percentage fee for the treasury
- Custom fees for specific callers

Recommendation

Implement the missing functionality in all vault contracts.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[QA-03] Optimization of Market Asset Retrieval

Description

In the `market` contract, the `asset` function retrieves the status of a given asset principal by querying the `assets` contract:

```
(define-private (asset (a principal))
  (let ((id       (contract-call? .assets get-reverse a))
        (final-id (buff-to-uint-be id)))
    (contract-call? .assets get-status final-id)))
```

Currently, this process involves two calls to the `assets` contract, with a basic conversion occurring between these calls.

Recommendation

To enhance efficiency, implement a function within the `assets` contract that directly retrieves the status of an asset using its principal. This function should then be called within the `market::asset` function.

Example implementation in the `assets` contract:

```
(define-read-only (get-asset-status (address principal))
  (let ((id (get-reverse address))
        (final-id (buff-to-uint-be id)))
    (get-status final-id)))
```

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[QA-04] Eliminate Redundant Contract Caller Authentication in Vault DAO Operations

Description

The current governance logic executes DAO operations through a proposal system, where each proposal is a contract in itself. To facilitate this, the `dao-executor::execute-proposal` function is invoked:

```
(define-public (execute-proposal (script <proposal-script>))
  (begin
    (try! (IMPL))
    (try! (as-contract? ((with-all-assets-unsafe))
      (try! (contract-call? script execute))
      true))
    (ok true)))
```

This function ensures that the current `tx-sender` is set to the `dao-executor` itself. Consequently, the proposal contract can execute all DAO-related operations, which verify the legitimacy of such actions. The check used to verify the DAO call in some parts of the codebase is as follows:

```
(define-private (DAO)
  (begin
    (asserts!
      (or (is-eq tx-sender .dao-executor)
          (is-eq contract-caller .dao-executor))
      ERR-AUTH)
(ok true)))
```

The `(is-eq tx-sender .dao-executor)` part is correct. However, the `(is-eq contract-caller .dao-executor)` is unnecessary since it can never be reached. The `dao-executor` contract does not have any function that can (or should) directly call system operations.

Recommendation

Utilize only the `(is-eq tx-sender .dao-executor)` logic check in the vault contracts where this modified `DAO` check exists.

Example implementation:

```
(define-private (DAO)
  (begin
    (asserts!
      (is-eq tx-sender .dao-executor)
      ERR-AUTH)
(ok true)))
```

[QA-05] Function Naming Ambiguities Severely Decrease Code Readability

Description

Throughout the codebase, there are instances of ambiguous, misleading, or unhelpful function names.

A more prominent issue is the frequent use of function names that are nouns representing the result of their actions, rather than verbs. This is a significant anti-pattern that greatly reduces code readability:

Examples in the `market` contract:

- `asset` instead of `get-asset`. Note that because the function itself is named `asset`, naming variables that contain the result of the function call becomes more complex.
  - Instead of using `(asset (get-asset address))`, the developer is forced to use `(a (asset addr))` or employ other naming artifices to differentiate it from the function itself, such as adding a leading underscore or other characters.

- `assets` instead of `get-assets`
- `position` instead of `get-position`
  - This also results in the need for awkward variable names, e.g., `(pos(position account))` instead of the more readable `(position (get-position account))`.

- `notional` instead of `get-notional-assets`
- `egroup` instead of `get-egroup`
- `enabled-mask` instead of `get-enabled-mask`
- `position-liq` instead of `get-liquidation-position`
- `position-full` instead of `get-full-position`

Another pattern that severely decreases code readability, sometimes stemming from the overlap with noun-named functions, is the use of abbreviated names. For example:

- `calc-asset-notional` instead of `calculate-asset-notional`
- `pos` instead of `position`
- `a` instead of `asset`
- `addr` instead of `address`
- `aid` instead of `asset-id`
- `liq-penalty` instead of `liquidation-penalty`

Recommendation

Rename all functions that have nouns as names to verbs, simply by prepending `get-` to them. Search for all occurrences of short variable or function names and write them out fully.

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[QA-06] General Code Style Improvements

Description

The codebase contains several opportunities for enhancing readability that can be easily implemented:

1. Use standard indentation instead of aligned indentation.

For example, instead of aligning, which both decreases readability and increases code size (implicitly affecting real-time execution costs):

```
(context              (liquidation-context borrower))
(pos                  (get position context))
(pos-full             (position-full borrower))
(alist                (get assets context))
(mask                 (get mask pos))
(group                (egroup mask))
```

Apply standard formatting:

```
(context (liquidation-context borrower))
(pos (get position context))
(pos-full (position-full borrower))
(alist (get assets context))
(mask (get mask pos))
(group (egroup mask))
```

This also applies to function arguments. Instead of:

```
(define-read-only (SERIALIZE-LEGAL
                (this uint)
                (args
                {
                  MASK           : uint,
                  LIQ-CURVE-EXP  : uint,
                  LIQ-PENALTY-MIN: uint,
                  LIQ-PENALTY-MAX: uint,
                  LTV-BORROW     : uint,
                  LTV-LIQ-PARTIAL: uint,
                  LTV-LIQ-FULL   : uint,
                 }))
```

Use:

```
(define-read-only (SERIALIZE-LEGAL
                (this uint)
                (args {
                  MASK: uint,
                  LIQ-CURVE-EXP: uint,
                  LIQ-PENALTY-MIN: uint,
                  LIQ-PENALTY-MAX: uint,
                  LTV-BORROW: uint,
                  LTV-LIQ-PARTIAL: uint,
                  LTV-LIQ-FULL: uint,
                 }))
```

2. Use lowercase for function names.

There are several functions within the codebase that are in uppercase without any apparent reason:

- `calc-asset-notional` shoud be `serialize-and-validate-input`

- `DAO` should be `check-dao-auth`
- `dao-executor::IMPL` should be `check-impl-auth`
- `dao-multisig::SIGNER` should be `check-signer-auth`
- `market-vault::INTERNAL` should be `check-impl-auth`
- `market-vault::REFRESH` should be `refresh`
- `vault::SYSTEM` should be `check-caller-auth`

3. Avoid declaring `let` variables for values retrieved from tuples only once.

For example:

```
(define-private (position (account principal)) ;; enabled only
  (let ((mask (enabled-mask)))
    (contract-call? .market-vault position account mask)))
```

You can remove the `let` and have a single call as:

```
(define-private (position (account principal)) ;; enabled onLy
    (contract-call? .market-vault position account (enabled-mask)))
```

Another example, instead of:

```
(define-private (liquidation-context (account principal))
  (let ((pos (position-liq account))
        (mask (get mask pos))
        (alist (assets mask)))
    {
      position: pos,
      assets: alist
    }))
```

You can have:

```
(define-private (liquidation-context (account principal))
  (let ((pos (position-liq account))
    {
      position: pos,
      assets: (assets (get mask pos)))
    }))
```

4. Use the new style tuple declaration instead of the old style.

Declaring a tuple in Clarity is allowed using both the old-style `tuple` keyword and curly brackets `{}`. The codebase mainly uses curly brackets but occasionally uses the tuple version, e.g.:

```
collateral:  (list 64 (tuple (aid uint) (amount uint))),
debt       : (list 64 (tuple (aid uint) (scaled uint)))
```

Change this to use only the new-style declaration, which uses curly brackets.

5. Improve switch-like statements.

Throughout the codebase, there are instances of switch-like code statements implemented using multiple `if-else` clauses.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

From a formatting point of view, instead of `if-else` with continuous indenting to the right:

```
(if (is-eq type TYPE-PYTH)
      (resolve-pyth ident)
      (if (is-eq type TYPE-DIA)
          (resolve-dia ident)
          ERR-TYPE)))
```

They can be better formatted on each line, as such:

```
(define-private (resolve-price (type (buff 1)) (ident (buff 32)))
  (if (is-eq type TYPE-PYTH) (resolve-pyth ident)
  (if (is-eq type TYPE-DIA) (resolve-dia ident)
  ERR-TYPE)))
```

Recommendation

Apply the mentioned style changes.

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[QA-07] Optimization for Enabling and Disabling Assets

Description

In the `assets` contract, the current process for enabling or disabling an asset involves:

- Retrieving registry data, including the `collateral` and `debt` status booleans, using the `status` method.
- Calculating the updated bitmap to represent the new state.
- Confirming that the action is authorized by the DAO.
- Ensuring the asset's current state is different from the intended new state.
- Updating the bitmap accordingly.

However, retrieving registry data and computing additional fields is unnecessarily complex and costly in terms of execution.

Recommendation

Simplify the logic in `assets::enable` by only checking if the current bitmap differs from the new one.

Example implementation:

```
(define-public (enable (asset principal) (collateral bool))
  (let ((id              (get-reverse asset))
        (final-id        (buff-to-uint-be id))
        (enabled-mask    (get-bitmap))
-       (a               (status final-id enabled-mask))
-       (c               (get collateral a))
-       (b               (get debt a))
        (pos             (mask-pos final-id collateral))
        (updated-bitmap (bit-or enabled-mask (pow u2 pos)))
     ;; --- dao auth ---
     (try! (DAO))

     ;; --- preconditions ---
     (asserts!
-      (if collateral
-          (not c)  ;; collateral must not already be enabled
-          (not b)) ;; debt must not already be enabled
+      (not (is-eq enabled-mask updated-bitmap))
       ERR-ALREADY-ENABLED)

      ;; --- enable ---
      (var-set bitmap updated-bitmap)
      (ok true)
     ))
```

Apply the same optimization to `assets::disable`.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[QA-08] Promote Debug Getters in eGroup to Production

Description

The `egroup` contract includes getter functions labeled as debug: `debug-get-popbucket` , `debug-get-bucket` , and `debug-get-reverse` . Facilitating the retrieval of on-chain data from an off-chain context will aid third-party integrations and monitoring systems.

Recommendation

Rename all debug getter functions to remove the `debug-` prefix and consider them as a standard part of the contract.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[QA-09] Simplify Nonce to a uint to Reduce Complexity

Description

The `nonce` variable is currently stored as a `(buff 4)`, necessitating conversion operations each time it is accessed or incremented. The optimization of `read_length` has surpassed the point of diminishing returns in terms of added complexity.

Recommendation

Convert all instances of `nonce` from `(buff 4)` to `uint` in the `assets` contract, and from `(buff 1)` to `uint` in the `egroup` contract.

After making these changes, remove the `uint-to-buff4` and `uint-to-buff1` conversions if they are no longer in use.

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[QA-10] Code Constants Usage Ambiguities

Description

The codebase contains several instances where constants are either misplaced, unused, or inconsistently used.

Consider the following issues:

1. Unused Market Error Code

The following error codes in the `market` are unused and should be removed:

```
(define-constant ERR-EXCESSIVE- LIQUIDATION (err u400019))
(define-constant ERR-EXCESSIVE-COLLATERAL-SEIZURE (err 400020))
(define-constant ERR-SKIPPED-NO-BALANCE     (err 4400021))
```

Additionally, the `MAX-U64` constant is defined in the `assets` contract but is never referenced in the codebase. It should be removed.

2. Misplaced Constant Definition Positioning

The constants `BPS`, `PRECISION`, and `INDEX-PRECISION` are used throughout the `market` contract but are defined in the `health` section of the file, midway through it.

```
;;-------------------------------------------------------------------
;; health
(define-constant BPS       u10000)
(define-constant PRECISION 100000000)
(define-constant INDEX-PRECISION 1000000000000) ;; 1e12 for index calculations
```

`BPS` and `PRECISION` should be defined in the `oracle constants and errors` section of the `market`.

3. Constant Name Case Mismatch

In the `market` contract, the names of constants related to assets and their zToken counterparts have inconsistent uppercase usage.

Example of naming:

```
(define-constant SBTC u1)
(define-constant zsBTC u6)
```

Options:

1. Use uppercase for the asset name and keep the `z` prefix lowercase, format: `z[NAME]`; e.g., `SBTC` and `zSBTC`.
2. Alternatively, use constant names to mimic the token symbol; e.g., `sBTC` and `zSBTC`.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

Recommendation

Implement the suggested changes for each situation.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[QA-11] Simplification of Retrieving Liquidation Position

Description

The `market::position-liq` function is currently used to obtain the collateral amounts (that are currently enabled) and all debt amounts (both enabled and disabled) for an account that is to be liquidated. The function is implemented as follows:

```
(define-private (position-liq (account principal)) ;; Liquidation specific
  (enabled collateral + all debt)
  (let ((enabled (enabled-mask))
        ;; Extract only collateral bits (0-63)
        (enabled-coll (bit-and enabled MAX-U64))
        ;; ALL debt bits set to 1 (64-127)
        (all-debt DEBT-MASK)
        ;; Combine: enabled collateral + all debt
        (liq-mask (bit-or enabled-coll all-debt)))
    (contract-call? market-vault position account liq-mask)))
```

After obtaining the currently enabled mask, the function sets all debt bits (leading 63-127 bits) to 1 and then calls `market-vault::position`.

This approach is redundant because the market-vault: :position function inherently returns the full aggregated debt, regardless of whether it is disabled:

```
(d
  (lookup-debt id mask MAX-U128))) ;; debt is always aggregated even if disabled
```

Recommendation

To streamline the `position-liq` function, directly call `market-vault::position` using the currently enabled market bitmap. Additionally, consider renaming the `position-liq` function to something more descriptive, such as `get-liquidation-position`.

Example implementation:

```
(define-private (position-liq (account principal))
  (contract-call? .market-vault position account (enabled-mask)))
```

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[QA-12] Optimization of Borrower Scaled Debt Retrieval

Description

In the `market` contract, both the `liquidate` and `repay` functions retrieve the current scaled account debt through two separate calls to the `market-vault::debt-scaled` function.

In the `liquidate` function, it is done as follows:

```
(ob (contract-call? .market-vault resolve borrower))
(oid (get id ob))
(curr-scaled (contract-call? .market-vault debt-scaled oid debt-aid))
```

In the `repay` function, the retrieval process is more complex but ultimately involves the same operations:

```
(define-private (resolve (account principal)) ;; obligation
  (contract-call? .market-vault resolve account))

;; ...

(ob      (resolve account))
(oid     (get id ob))
;; ...
(scurr (contract-call? .market-vault debt-scaled oid aid))
```

It is important to note that the `market::resolve` function is only invoked from the `repay` function and solely in this context.

The current implementation requires two calls to the same `market-vault` contract, with a value being passed between these calls. This redundancy introduces unnecessary overhead and can be streamlined.

Recommendation

Introduce a function in the `market-vault` contract named `get-account-scaled-debt`, which takes the borrower principal and the debt asset ID as parameters and returns the scaled debt value.

Utilize this new function in both the `market::repay` and `market::liquidate` functions. Additionally, eliminate the `market::resolve` function entirely, as it will no longer be necessary.

[QA-13] Improvements Needed for Mask Market Contract Operations

Description

In the `market` contract, there are several operations that could be enhanced both logically and in terms of execution fees:

1. Reuse the `min` function when selecting the smallest number:

```
(dec (if (> sdelta scurr) scurr sdelta))
```

2. Utilize standard Clarity bitwise operators for mask manipulation.

There are two instances of inlined bitwise operations that can be simplified using bitwise operators:

- In `repay`:

```
(debt-bit-pos (+ aid u64))
(div (pow u2 debt-bit-pos))
(future-mask (- mask div))
```

This can be rewritten as: `future-mask (bit-and mask (bit-not (pow u2 (+asset-id DEBT-OFFSET))))`

- In `collateral-remove`:

```
(let ((coll-bit-pos aid)
      (div (pow u2 coll-bit-pos))
      (future-mask (- mask div)))
```

This can be written as: `(let ((future-mask (bit-and position-mask (bit-not(pow u2 asset-id)))))`

- In `borrow`:

```
(debt-bit-pos (+ aid u64))
(future-mask (let ((div (pow u2 debt-bit-pos))
                   (shiftr (/ mask div))
                   (bit (mod shiftr u2))
                   (base (if (is-eq bit u0) div u0)))
                   (+ mask base)))
```

This can be rewritten as: `(future-mask (bit-or mask (pow u2 (+ asset-id DEBT-OFFSET) )))`

Recommendation

Implement the suggested changes.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[QA-14] Function `check-egroup-invariant` Contains Inefficiency and Redundancies

Description

The implementation of `check-egroup-invariant` suffers from deep nesting of unoptimized condition checks and lacks early returns. Examples include:

- The `valid` and `exclude-id` variables are used only once and can be inlined.
- The checks for `valid`, `over max-id`, and `exclude ID` can be combined into a single `if` condition.
- Both `new-is-superset` and `existing-is-superset` variables include a `(not(is-eq existing-mask new-mask))` check. This implies that if they are equal, the `invariant-holds` variable defaults to `true`, making the check redundant. This check can be separated and moved to the beginning of the code block, alongside other skip-this-iteration checks.
- Since both `new-is-superset` and `existing-is-superset` variables are used only once, they do not need to be declared. Their new forms, `(subset existing-mask new-mask)` for `new-is-subset` and `(subset new-mask existing-mask)` for `existing-is-superset`, can be inclined.

Recommendation

Implement the suggested optimizations in the `egroup::check-egroup-invariant` function.

Example implementation:

CONTENTS


```
(define-private (check-egroup-invariant
  (id uint)
  (acc { new-mask: uint,
         new-Itv-borrow: uint,
         new-ltv-liq-partial: uint,
         new-ltv-liq-full: uint,
         exclude-id: (optional uint),
         max-id: uint,
         valid: bool }))

;; Sanity check
(if (or (not (get valid acc))
        (>= id (get max-id acc))
        (is-eq (some id) (get exclude-id acc)))
    acc
    ;; Check invariant
    (let ((existing (lookup id))
          (existing-mask (get MASK existing))
          (new-mask (get new-mask acc)))c

      ;; Skip if equal
      (if (is-eq existing-mask new-mask)
          acc
          (let ((existing-ltv-borrow (buff-to-uint-be
            (get LTV-BORROW existing)))
                (existing-ltv-liq-partial (buff-to-uint-be
                  (get LTV-LIQ-PARTIAL existing)))
                (existing-ltv-liq-full (buff-to-uint-be
                  (get LTV-LIQ-FULL existing)))
                (new-ltv-borrow (get new-ltv-borrow acc))
                (new-Itv-liq-partial (get new-ltv-liq-partial acc))
                (new-ltv-liq-full (get new-ltv-liq-full acc))
                ;; Determine relationship
                (holds
                  (if (subset existing-mask new-mask)
                      ;; New is a proper superset / LTVn ‹= LTVe
                      (and (<= new-ltv-borrow existing-ltv-borrow)
                           (<= new-ltv-liq-partial existing-ltv-liq-partial)
                           (<= new-ltv-liq-full existing-ltv-liq-full))
                      (if (subset new-mask existing-mask)
                          ;; Existing is a proper superset / LTVn >= LTVe
                          (and (>= existing-ltv-borrow new-ltv-borrow)

                                 (>= existing-ltv-liq-partial new-ltv-liq-partial)
                               (>= existing-ltv-liq-full new-ltv-liq-full))
                          ;; No subset relationship
                          true))))

          (merge acc { valid: holds }))))))
```

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[QA-15] Redundant Parameter Fragment

Description

The function `price-resolve` accepts `aid` as a parameter, which it subsequently passes to `resolve-callcode` with the following comment explaining its purpose:

```
;; aid is passed so resolve-token can fetch cached lindex
       (final-price (try! (resolve-callcode p callcode aid)))
```

However, `resolve-callcode` does not make use of `aid`, rendering it redundant.

```
(define-private (resolve-callcode (p uint) (calicode (optional (buff 1)))
  (aid uint))
  (let ((cc (unwrap! callcode (ok p))))
    (if (is-eq cc CALLCODE-STSTX)
        (resolve-ststx p)
        (if (is-eq cc CALLCODE-ZSTX)
            (resolve-ztoken p STX)
            (if (is-eq cc CALLCODE-ZSTSTX)
                (resolve-ztoken p STSTX)
                (if (is-eq cc CALLCODE-ZUSDC)
                    (resolve-ztoken p USDC)
                    (if (is-eq cc CALLCODE-ZUSDH)
                        (resolve-ztoken p USDH)
                        ERR-ORACLE-CALLCODE)))))))
```

Recommendation

Remove the `aid` parameter and any related comments from both `price-resolve` and `resolve-callcode`.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[QA-16] Enhance Market Contract External Interface

Description

The `market` contract is designed for use by third-party integrators, providing entry points for borrowing, repaying, collateral management, and liquidations.

At present, all public-facing entry points only return `(ok true)`, which offers no meaningful information to integrators.

Recommendation

To enhance third-party external integration, implement the following changes:

1. In the `market::repay` function, return the `amount-to-repay`, which represents the actual amount repaid.
2. In the `market::liquidate` function (and the subsequent `liquidate-multi`), return a tuple containing the debt repaid and the collateral collected. These values can vary significanctly during the function's execution.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[QA-17] Create Market Trait

Description

The `market` contract is designed for use by third-party integrators. It provides entry points for borrowing, repaying, collateral management, and liquidations.

At present, there is no official trait available for third-party integrators to use.

Recommendation

Develop a market trait for use by external integrators.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[QA-18] Implement a Majority Rule-Based Multisig

Description

In the current design of the DAO multisig contract, signer approvals are necessary for proposals that alter the signer set, whether it involves adding a new signer, removing an existing one, or updating the signer approval threshold.

The threshold is a numerical value, and approval requires reaching this specified count.

This approved count threshold model can lead to certain vulnerabilities. For instance, if just one multisig voter is compromised, the entire protocol could be at risk.

For example, if a signer is compromised, one of the following actions must be taken to mitigate the issue:

- Remove the compromised signer, provided the remaining signers still meet the approval threshold.
- Add a new signer to replace the compromised one.
- Lower the approval threshold so that the compromised signer's approval is no longer necessary.

However, if the approval threshold equals the total number of signers, the proposal system can become deadlocked.

A compromised signer could refuse to approve any proposals, including those necessary to remove themselves from the signer set. This behavior would prevent all proposals from being executed.

In contrast, if the threshold is set as a percentage, such as a hardcoded 66%, the majority will decide regardless of the number of voters. Additionally, the threshold could apply the 50%+1 rule instead of a higher percentage.

Recommendation

Implement a majority rule-based multisig system for enhanced security.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[QA-19] Integrate Max Staleness into Asset Oracle Data Entry

Description

Currently, the oracle staleness configuration is managed through a global `default-max-staleness` data variable and a separate `feed-max-staleness` map for per-feed overrides. This setup introduces unnecessary complexity and requires manual synchronization between asset registration and staleness configuration. Whenever an asset is added, its staleness must be configured separately in a different map using `set-feed-max-staleness`.

Recommendation

Incorporate `max-staleness` as a field within the asset's oracle data structure in the asset registry.

```
oracle: 1
  type: (buff 1),
  ident: (buff 32),
  callcode: (optional (buff 1)),
  max-staleness: uint
}
```

Subsequently, remove the default staleness logic from the `market` and utilize the data from the asset oracle entry.

[QA-20] Remove Unused Market Contract Code Artifacts

Description

In the `market` contract, there are opportunities for minor improvements to enhance code readability and optimize operations:

1. Redundant Map Definition

The `ztoken-asset-ids` map is defined but never utilized within the codebase. This results in unnecessary storage overhead and code clutter without adding any functionality.

```
(define-map ztoken-asset-ids uint bool)
```

It is recommended to remove the unused `ztoken-asset-ids` map definition.

2. Unused Constants Post-Deployment

There are several unused constants that are remnants from development. These will be removed after the production update, but they are noted here for reference:

- `PYTH-STORAGE` constant
- `STSTX-DATA-CORE` constant
- `STSTX-RESERVE` constant

3. Additional Unused Constants

- `ERR-NO-VALID-EGROUP`
- `PRECISION` (replaced by `INDEX-PRECISION`)

4. Irrelevant Comments
Remove the following irrelevant comments:

- L305
- L338
- L343

Recommendation

Remove the specified code artifacts.

CONTENTS

ClarityAlliance
Security Review

Zest Protocol
v2 Upgrade

[QA-21] Isolate stSTX Price Resolution from resolve-ztoken

Description

The `resolve-ststx` function is invoked within `resolve-ztoken` for each price resolution operation. This introduces computational overhead for non-stSTX assets, as the conditional check is evaluated with every call, even when it is not necessary.

```
(final-price (if (is-eq aid STSTX)
    (try! (resolve-ststx p))
    p))
```

Recommendation

Shift the stSTX price resolution logic to the caller level. This allows for direct price resolution for most assets and ensures that the stSTX-specific logic is only executed when required.

```
@@ -360,19 +360,14 @@
   (let ((ratio (unwrap! (call-ststx-ratio) ERR-ORACLE-CALLCODE)))
     (ok (mul-div-down p ratio STSTX-RATIO-DECIMALS))))

-;; ztoken transformation - OPTIMIZATION: Uses cached liquidity index
 (define-private (resolve-ztoken (p uint) (aid uint))
-  (let ((final-price  (if (is-eq aid STSTX)
-                 (try!   (resolve-ststx p))
-                 p))
-        ;; CRITICAL: Fetch lindex from cache instead of cross-contract call
-        (cached (unwrap! (get-cached-indexes aid) ERR-ORACLE-CALLCODE))
+  (let ((cached (unwrap! (get-cached-indexes aid) ERR-ORACLE-CALLCODE))
         (cached-lindex (get lindex cached))
-        (scaled (* final-price cached-lindex)))
+        (scaled (* p cached-lindex)))
     (ok (div-down scaled INDEX-PRECISION))))

 ;; callcode dispatcher
-(define-private (resolve-callcode (p uint) (callcode (optional (buff 1)))
- (aid uint))
+(define-private (resolve-callcode (p uint) (callcode (optional (buff 1))))
   (let ((cc (unwrap! callcode (ok p))))
     (if (is-eq cc CALLCODE-STSTX)
         (resolve-ststx p)
@@ -381,7 +376,7 @@
             (if (is-eq cc CALLCODE-ZSBTC)
                 (resolve-ztoken p SBTC)
                 (if (is-eq cc CALLCODE-ZSTSTX)
-                    (resolve-ztoken p STSTX)
+                    (resolve-ztoken (try! (resolve-ststx p)) STSTX)
                     (if (is-eq cc CALLCODE-ZUSDC)
                         (resolve-ztoken p USDC)
                         (if (is-eq cc CALLCODE-ZUSDH)
```