Lydian
Protocol Audit

0f.studio

Date: 17/02/2022

Git Revision: 6f88efc32055841ba1b49a6ba61f74dffdb5a1a5

Protocol Summary

Lydian is a decentralized reserve currency protocol based on the Lydian (LDN) token. The Lydian token is backed by assets, including USDA & wLDN/USDA LP tokens, in the Lydian treasury. Users can benefit from the protocol and gain rewards either through staking their Lydian tokens or purchasing bonds. Users can wrap their Lydian tokens and provide liquidity to the wLDN/USDA pool on Arkadiko whose initial liquidity is provided by the Lydian team. Besides, users can buy wrapped Lydians (wLDN) on Arkadiko and stake them on Lydian.

During the protocol bootstrapping phase, users can participate in an auction to receive Lydian tokens through committing USDA tokens.

The protocol has a governance process in place through which users can submit proposals that run for 10 days and vote for proposals using staked or lydian tokens. The approved proposals are executed through the DAO.

Protocol Actors

User	Any user that uses the protocol for staking or buying bonds
DAO	A contract that executes proposals approved through the active governance
Depositor	User added by the DAO with the ability to deposit tokens to the Lydian treasury and get Lydians in return
Debtor	User added by the DAO with the ability to incur treasury debt and receiving treasury assets to be used in yield farming
Minter	A contract enabled to mint/burn tokens

User Stories

Below is the set of user stories, grouped by contract, the audit was based on. Listed next to a user story are the findings associated with it, if any.

Auction

- **User can commit USDA, *0F-LDN-006***
- **User can withdraw LND once auction successfully ended, *0F-LDN-004*, *0F-LDN-006***
- **User can withdraw committed USDA if auction failed or canceled**
- **DAO can cancel the auction, *0F-LDN-004***
- **DAO can transfer tokens from auction contract**
- **DAO can add an auction, *0F-LDN-004***

Bond Depository

- **User can deposit token and set slippage, receives sLDN after vesting**
- **DAO can add and update a bond**
- **DAO can set active bond teller**
- **DAO can set active treasury**

Bond Teller

- **User can redeem LDN after vesting ends**
- **DAO can set active bond depository**
- **DAO can set active staking**
- **DAO can set active treasury**
- **DAO can enable/disable contract**
- **DAO can migrate funds**

Staking

- **User can stake LDN and will receive same amount of sLDN**
- **User can unstake sLDN and will receive same amount of LDN**
- **User can execute rebase if epoch-end-block reached**
- **DAO can update epoch length and end-block**
- **DAO can set active staking distributor**
- **DAO can enable/disable contract**
- **DAO can migrate funds, *0F-LDN-003***

Staking Distributor

- **Active recipient can get new rewards according to rate schedule**
- **DAO can set active treasury**

- DAO can add recipient with reward rate
- DAO can set adjustment rate and target

Treasury

- User can audit reserve token to save total reserves
- Depositor can deposit token and get LDN
- Depositor can withdraw token for LDN
- Debtor can incur debt, *0F-LDN-001*, *0F-LDN-003*, *0F-LDN-005*
- Debtor can repay debt
- DAO can enable reserve token
- DAO can disable reserve token
- DAO can set active minter
- DAO can set enable/disable debtor for certain token
- DAO can set enable/disable depositor for certain token
- DAO can enable/disable contract
- DAO can transfer token, *0F-LDN-007*
- DAO can migrate funds, *0F-LDN-003*
- Active minter can mint LDN if enough excess reserves, *0F-LDN-007*

Value Calculator

- DAO can add token info

DAO

- Active governance can execute proposal as DAO
- Proposal executed by DAO can update active governance

Wrapped Lydian Creator

- User can wrap sLDN into wLDN
- User can unwrap wLDN to sLDN

Governance

- User can create proposal that runs for 10 days, *0F-LDN-003*
- User can vote for/against a proposal by sending LDN or sLDN, *0F-LDN-010*
- User can claim back tokens after a proposal ended
- User can execute proposal once
- Contract owner can create proposal that runs for a certain amount of days
- Contract owner can execute proposal immediately, *0F-LDN-009*
- DAO can enable/disable contract
- DAO can migrate funds, *0F-LDN-003*

Lydian Token

- SIP-010
- User can burn owned LDN
- Active minter can mint new LDN for any wallet
- Active minter can burn LDN for any wallet
- DAO can set active minter
- DAO can set token URI

Staked Lydian Token

- SIP-010, *0F-LDN-003*, *0F-LDN-008*
- User can claim rebase rewards
- User can burn owned sLDN, *0F-LDN-002*, *0F-LDN-005*
- DAO can set active staking
- DAO can set active treasury
- DAO can enable/disable contract
- DAO can migrate funds, *0F-LDN-003*
- DAO can mint/burn tokens from any wallet, *0F-LDN-002*, *0F-LDN-008*
- Active staking can rebase, *0F-LDN-003*

Wrapped Lydian Token

- SIP-010
- User can burn owned wLDN
- DAO can set active minter
- Active minter can mint/burn tokens for any wallet

Findings

Findings Severity Classification

A finding is an issue in the protocol that might get exploited or cause the protocol not to function properly or as expected. Findings are classified as per the below:

Critical: A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality and we recommend these issues to be fixed before moving to a live environment.

Medium: The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low: Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational: Informational issues indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

0F-LDN-001 - Debtor can potentially deplete treasury from its assets

Context

The DAO assigns a debtor that is able to incur treasury debt and receive in return treasury assets as long as the debtor has enough staked lydians in his balance. The debtor uses the assets retrieved from the treasury to provide liquidity to pools in other Defi protocols and earn rewards.

Concern

The debtor would be able to deplete treasury from its assets through in-curing debt to receive usda tokens, wldn/usda lp tokens or other supported treasury assets. Then the debtor would use these tokens to stake more Lydians, which gives him the ability to incur more debt.

The debtor can keep on doing so, depleting the treasury from its reserves. Steps leading to this issue:

1. Debtor incur treasury debt through the incur-debt method of the treasury-v1-1 contract.

```
;; ----------------------------------------
;; Debt
;; ----------------------------------------
(define-public (incur-debt (token-trait <ft-trait>) (value-calculator
<value-calculator-trait>) (amount uint))
(let (
(debtor tx-sender)
(token-map (get-reserve-token (contract-of token-trait)))
(value (unwrap-panic (get-token-value token-trait value-calculator amount)))
(debtor-enabled (get enabled (get-debtor-info debtor (contract-of token-trait))))
(available-debt (unwrap-panic (contract-call? .staked-lydian-token get-available-debt
debtor))))
```

2. Debtor gets wrapped lydians on Arkadiko using the Lydian treasury tokens he got in step 1
3. Debtor unwraps wLDN tokens to staked lydians

```
;; wrapped-lydian-creator-v1-1
(define-public (unwrap (amount uint))
  (let (
    ;; Claim rebase if needed
    (claim-result (claim))

    (recipient tx-sender)
    (index (contract-call? .staked-lydian-token get-index))
    (staked-amount (/ (* amount index) u1000000))
  )
    ;; Transfer sLDN to recipient
    (try! (as-contract (contract-call? .staked-lydian-token transfer staked-amount
(as-contract tx-sender) recipient none)))

    ;; Burn wLDN for user
    (try! (contract-call? .wrapped-lydian-token burn recipient amount))

    (ok staked-amount)
  )
)
```

4. Debtor incur more treasury debt now that he has more staked lydians and thus more available debt balance

Recommendation

Use a decentralized process to enable farming while avoiding this scenario. Benefit from the protocol governance process to submit a proposal that would incur treasury debt and directly provide liquidity to liquidity pools in other DeFi protocols upon proposal approval and without the need for the debtor role.

OF-LDN-002- A user can transfer burned staked lydians back to his account upon claiming rebase

Context

Staked lydian token is a SIP-010 token that allows users to burn their tokens through the implementation of a burn method. Users can stake their lydian tokens and get in return rewards through a rebase mechanism that happens every epoch.

The rebase mechanism mainly increases the number of minted staked lydian tokens to match the amount of lydian tokens the staking contract has. This additional amount of staked lydian tokens will go as rewards to users that are staking their tokens. The mechanism represents the user staked lydian token balance in fragments and updates the fragments per token every epoch instead of updating the balances for all users.

Concern

The implemented burn method in the staked-lydian-token contract only burns user tokens without updating user token fragments information.

```
(define-public (burn (recipient principal) (amount uint))
(begin
(asserts!
(or
(is-eq contract-caller .lydian-dao)
(is-eq contract-caller recipient)
)
(err ERR-NOT-AUTHORIZED)
)
(ft-burn? staked-lydian amount recipient)
)
)
```

The user will be able to transfer back his burned tokens upon claiming rebase rewards since the amount of staked Lydians that is claimed depends on the number of fragments the user has which still includes the fragments related to the burned tokens.

```
(define-public (get-claim-rebase (account principal))
(let (
(fragments (get fragments (get-account-fragments account)))
(new-balance (/ fragments (var-get fragments-per-token)))
(current-balance (unwrap-panic (get-balance account)))
(diff (- new-balance current-balance))
)
(ok diff)
```

```
)
)
```

Recommendation

To avoid this behavior, the protocol should update the account-fragments map after burning tokens so that it reflects the new token amounts.

```
(define-public (burn (recipient principal) (amount uint))
(let
(
(fragments (get fragments (get-account-fragments recipient)))
(burn-fragments (/ amount (var-get fragments-per-token)))
(new-fragment-balance (- fragments burn-fragments))
)
(asserts!
(or
(is-eq contract-caller .lydian-dao)
(is-eq contract-caller recipient)
)
(err ERR-NOT-AUTHORIZED)
)
(map-set account-fragments { account: recipient } { fragments: new-fragment-balance })
(ft-burn? staked-lydian amount recipient)
)
)
```

0F-LDN-003 - A user’s staked lydians balance does not include rebase rewards

Context

The protocol gives users the opportunity to gain rewards by staking their lydian tokens. The rewards are generated and claimed through the rebase mechanism. The user should mainly claim his rewards.

Concern

The rebase logic updates the number of fragments per token. A user would still have the same staked lydians balance after a rebase unless he manually claims the rewards. This impacts the protocol decisions that depend on a user’s staked lydian balance. Below is a list in the identified cases:

A user might not be able to submit a proposal even though he has 1% of the total token supply.

```
;; Governance-v1-1
;; -----------------------------------------
;; Core
;; -----------------------------------------

(define-public (propose-public

(title (string-utf8 256))

(url (string-utf8 256))

(contract principal)

(start-block-height uint))

(let (

(supply (unwrap-panic (contract-call? .lydian-token get-total-supply)))

(ldn-balance (unwrap-panic (contract-call? .lydian-token get-balance tx-sender)))

(sldn-balance (unwrap-panic (contract-call? .staked-lydian-token get-balance tx-sender)))

(total-balance (+ ldn-balance sldn-balance))

)

;; Requires 1% of the supply

(asserts! (>= (* total-balance u100) supply) (err ERR-INSUFFICIENT-BALANCE))

;; Update proposals

(propose title url contract start-block-height u1440)
```

```
)
)
```

1. **A debtor might not be able to incur treasury debt even though he might have enough staked lydians that are not yet reflected in his balance**

```
;; Treasury-v1-1
;; -----------------------------------------
;; Debt
;; -----------------------------------------

(define-public (incur-debt (token-trait <ft-trait>) (value-calculator
<value-calculator-trait>) (amount uint))
(let (
(debtor tx-sender)
(token-map (get-reserve-token (contract-of token-trait)))
(value (unwrap-panic (get-token-value token-trait value-calculator amount)))
(debtor-enabled (get enabled (get-debtor-info debtor (contract-of token-trait))))
(available-debt (unwrap-panic (contract-call? .staked-lydian-token get-available-debt
debtor)))
)
(asserts! (var-get contract-is-enabled) (err ERR-CONTRACT-DISABLED))
(asserts! debtor-enabled (err ERR-DEBTOR-NOT-AUTHORIZED))
(asserts! (>= available-debt value) (err ERR-EXCEED-DEBT-LIMIT))
(asserts! (is-eq (contract-of value-calculator) (get value-calculator token-map)) (err
ERR-WRONG-VALUE-CALCULATOR))
```

2. **Migrating funds in different contracts would not transfer staked lydians rewarded through a rebase**

```
;; Staking-v1-1, Bond-teller, Governance-v1-1, Treasury-v1-1
(define-public (migrate-funds (recipient principal))
(let (
(ldn-balance (unwrap-panic (contract-call? .lydian-token get-balance (as-contract
tx-sender))))
(sldn-balance (unwrap-panic (contract-call? .staked-lydian-token get-balance (as-contract
tx-sender))))
)
(asserts! (is-eq tx-sender .lydian-dao) (err ERR-NOT-AUTHORIZED))
;; Transfer LDN
(if (> ldn-balance u0)
  (try! (as-contract (contract-call? .lydian-token transfer ldn-balance (as-contract
tx-sender) recipient none)))
  true
)
```

Recommendation

One solution to fix this would be to check for the rebase whenever the get-balance method is used. Claim rebase before using the staked lydian token get-balance method then continue with the protocol logic.

```
;; ---------------------------------------
;; Core
;; ---------------------------------------

(define-public (propose-public
(title (string-utf8 256))
(url (string-utf8 256))
(contract principal)
(start-block-height uint)
)
(let (
(supply (unwrap-panic (contract-call? .lydian-token get-total-supply)))
(ldn-balance (unwrap-panic (contract-call? .lydian-token get-balance tx-sender)))
(rebase (unwrap-panic (contract-call? .staked-lydian-token claim-rebase )))
(sldn-balance (unwrap-panic (contract-call? .staked-lydian-token get-balance tx-sender)))
(total-balance (+ ldn-balance sldn-balance))
)
;; Requires 1% of the supply
(asserts! (>= (* total-balance u100) supply) (err ERR-INSUFFICIENT-BALANCE))
;; Update proposals
(propose title url contract start-block-height u1440)
```

0F-LDN-004 - A user can withdraw lydians from a canceled auction in case the auction minimum price was less than 1 dollar.

Context

The protocol enables users to get lydian tokens by participating in an auction. The DAO creates an auction setting the auction minimum price, maximum price, payment token, start block and end block. If the auction is successful, participants would be able to withdraw lydian tokens from the auction contract. Besides, the DAO has the ability to cancel an auction at any point in time.

Concern

If the auction minimum price was set to a value that is less than 1 dollar, the canceled auction status would be successful and participants would be able to withdraw lydians from the auction. Below are the steps leading to this issue:

1. **DAO creates an auction with minimum price less than or equal to 1 dollar**

```
(define-public (add-auction
(payment-token principal)
(start-block uint)
(end-block uint)
(total-tokens uint)
(start-price uint)
(min-price uint)
)
(let (
(auction-id (var-get auction-counter))
)
(asserts! (is-eq tx-sender .lydian-dao) (err  ERR-NOT-AUTHORIZED))
```

2. **Users participates in an auction by committing tokens (USDA)**

```
;; ---------------------------------------
;; Commit
;; ---------------------------------------
(define-public (commit-tokens (token <ft-trait>) (auction-id uint) (amount uint))
(let (
(auction (unwrap-panic (get-auction-info auction-id)))
(tokens-to-transfer (calculate-commitment auction-id amount))
)
(asserts! (auction-open auction-id) (err ERR-AUCTION-NOT-OPEN))
(asserts! (is-eq (contract-of token) (get payment-token auction)) (err ERR-WRONG-TOKEN))
```

3. **DAO cancels the auction**

```
(define-public (cancel-auction (auction-id uint))
  (let (
    (auction (unwrap-panic (get-auction-info auction-id)))
```

```
  )
    (asserts! (is-eq tx-sender .lydian-dao) (err ERR-NOT-AUTHORIZED))

    ;; Set total-tokens so auction is not successful
    (map-set auction-info
      { auction-id: auction-id }
      (merge auction { end-block: block-height, total-tokens: (get total-committed auction) })
    )

    (ok true)
  )
)
```

4. Users withdraw lydian tokens from the auction

```
(define-public (withdraw-tokens (auction-id uint))
  (let (
    (user tx-sender)
    (claimable (tokens-claimable auction-id user))

    (user-committed (get-commitments user auction-id))
    (current-claimed (get claimed user-committed))
  )
  (asserts! (> claimable u0) (err ERR-NO-CLAIMABLE-TOKENS))
```

Recommendation

Make sure to add validations to the parameters of the add-auction method in the auction-v1-1 contract. Make sure min-price > 1, start-block < end-block, start-block > block-height.

```
(define-public (add-auction
    (payment-token principal)
    (start-block uint)
    (end-block uint)
    (total-tokens uint)
    (start-price uint)
    (min-price uint)
  )
  (let (
    (auction-id (var-get auction-counter))
  )
    (asserts! (is-eq tx-sender .lydian-dao) (err  ERR-NOT-AUTHORIZED))
    (asserts! (> min-price u1000000) (err ERR-AUCTION-MIN-PRICE))
```

Or have a status for the auction reflecting if it is canceled, successful or failed.

Similar to the add-auction method, other methods in the protocol executed through the DAO do not include validations for the executed method parameters.

0F-LDN-005 - A debtor can burn staked lydians regardless of his current debt

Context

The protocol allows debtors to incur treasury debt as long as the debtor has enough staked lydians. The protocol keeps track of the debtor's debt balance and locks an equivalent amount of staked lydians that he is not allowed to transfer to other users.

Concern

A debtor can incur treasury debt and then burn his staked lydian tokens regardless of his current debt. Below are the steps leading to this issue:

1. **Debtor incur treasury debt**

```
;; ----------------------------------------
;; Debt
;; ----------------------------------------

(define-public (incur-debt (token-trait <ft-trait>) (value-calculator
<value-calculator-trait>) (amount uint))
  (let (
    (debtor tx-sender)
    (token-map (get-reserve-token (contract-of token-trait)))
    (value (unwrap-panic (get-token-value token-trait value-calculator amount)))
    (debtor-enabled (get enabled (get-debtor-info debtor (contract-of token-trait))))

    (available-debt (unwrap-panic (contract-call? .staked-lydian-token get-available-debt
debtor)))
  )
```

2. **Debtor burns staked lydian tokens regardless of his debt**

```
(define-public (burn (recipient principal) (amount uint))
  (begin
    (asserts!
      (or
        (is-eq contract-caller .lydian-dao)
        (is-eq contract-caller recipient)
      )
      (err ERR-NOT-AUTHORIZED)
    )
      (ft-burn? staked-lydian amount recipient)
  )
)
```

Recommendation

Make sure to lock debtor staked lydian token amounts equivalent to this debt in the burn method of the staked-lydian-token contract.

```
(define-public (burn (recipient principal) (amount uint))
  (let
  (
    (current-balance (unwrap-panic (get-balance recipient)))
    (new-balance (- current-balance amount))
    (debt (get debt (get-debt-balance recipient))))

    (asserts!
      (or
        (is-eq contract-caller .lydian-dao)
        (is-eq contract-caller recipient)
      )
      (err ERR-NOT-AUTHORIZED)
    )
    (asserts! (> new-balance debt) (err ERR-DEBT))
    (ft-burn? staked-lydian amount recipient))
)
```

0F-LDN-006 - Users would not be able to withdraw all lydian tokens they are eligible for from a successful auction

Context

The protocol gives users the opportunity to get lydian tokens through participating and committing USDA tokens into an auction. If the auction was successful, then the user would be able to withdraw his share of the auction lydian tokens.

Concern

In the auction-v1-1 contract, the method commit-tokens calculates the amount of tokens a user can commit taking into consideration the auction maximum commitment. However, what is being transferred to the auction is all the amount specified by the user. This would allow users to commit tokens more than the auction maximum commitment capacity which will cause the user claimable amount in case of a successful auction to be much lower than what he is eligible for.

This is besides the fact that what is being initially transferred from the user's account to the auction is more than what it should be.

```
;; ----------------------------------------
;; Commit
;; ----------------------------------------

(define-public (commit-tokens (token <ft-trait>) (auction-id uint) (amount uint))
  (let (
    (auction (unwrap-panic (get-auction-info auction-id)))
    (tokens-to-transfer (calculate-commitment auction-id amount))
  )
    (asserts! (auction-open auction-id) (err ERR-AUCTION-NOT-OPEN))
    (asserts! (is-eq (contract-of token) (get payment-token auction)) (err ERR-WRONG-TOKEN))

    (if (> tokens-to-transfer u0)
      (begin
        ;; Transfer from user
        (try! (contract-call? token transfer amount tx-sender (as-contract tx-sender) none))

        ;; Add commitment
        (add-commitment auction-id tx-sender amount)
      )
      (ok u0)
    )
  )
)
```

Recommendation

To fix this issue, the calculated tokens-to-transfer amount should be the amount transferred from the user to the auction instead of the amount initially provided by the user.

```
;; -----------------------------------------
;; Commit
;; -----------------------------------------

(define-public (commit-tokens (token <ft-trait>) (auction-id uint) (amount uint))
  (let (
    (auction (unwrap-panic (get-auction-info auction-id)))
    (tokens-to-transfer (calculate-commitment auction-id amount))
  )
    (asserts! (auction-open auction-id) (err ERR-AUCTION-NOT-OPEN))
    (asserts! (is-eq (contract-of token) (get payment-token auction)) (err ERR-WRONG-TOKEN))

    (if (> tokens-to-transfer u0)
      (begin
        ;; Transfer from user
        (try! (contract-call? token transfer tokens-to-transfer tx-sender (as-contract
tx-sender) none))

        ;; Add commitment
        (add-commitment auction-id tx-sender tokens-to-transfer)
      )
      (ok u0)
    )
  )
)
```

0F-LDN-007 - DAO can transfer treasury tokens without updating the treasury token reserves

Context

The protocol allows the DAO to transfer assets from the treasury. Total treasury reserves are updated whenever tokens are deposited or withdrawn or when a debtor incurs treasury debt or repays the debt. The treasury keeps track of its excess token reserves making sure there are enough assets backing the lydian token. If there are not enough reserves, the treasury stops minting lydians.

Concern

The transfer-tokens method transfers tokens from the treasury without adjusting treasury reserves. This would let the treasury to mint lydians even if there is not enough excess reserves. Steps leading to this issue:

1. **DAO transfers treasury assets**

```
(define-public (transfer-tokens (token <ft-trait>) (amount uint) (recipient principal))
  (begin
    (asserts! (is-eq tx-sender .lydian-dao) (err ERR-NOT-AUTHORIZED))
    (try! (as-contract (contract-call? token transfer amount (as-contract tx-sender) recipient
none)))
    (ok true)
  )
)
```

2. **Treasury mints lydians due to a staking rebase**

```
(define-public (mint (recipient principal) (amount uint))
  (let (
    (minter-enabled (get enabled (get-minter-info contract-caller)))

    (excess-reserves (unwrap-panic (get-excess-reserves)))
  )
    (asserts! (var-get contract-is-enabled) (err ERR-CONTRACT-DISABLED))
    (asserts! minter-enabled (err ERR-NOT-AUTHORIZED))
    (asserts! (<= amount excess-reserves) (err ERR-INSUFFICIENT-RESERVES))

    (contract-call? .lydian-token mint recipient amount)
  )
)
```

Recommendation

To fix this issue, have the transfer-tokens method call the treasury audit-reserves-token method to adjust the treasury total reserves accordingly instead of depending on the proposal to do so.

```
(define-public (transfer-tokens (token <ft-trait>) (value-calculator <value-calculator-trait>)
(amount uint) (recipient principal))
  (begin
    (asserts! (is-eq tx-sender .lydian-dao) (err ERR-NOT-AUTHORIZED))
    (try! (as-contract (contract-call? token transfer amount (as-contract tx-sender) recipient
none)))
    (try! (audit-reserve-token token-trait value-calculator))
    (ok true)
  )
)
```

0F-LDN-008 - A user will not be able to transfer his minted staked lydian tokens

Context

Staked lydian token is a SIP-010 token that enables minting and transferring staked lydians. The transfer method depends on the amount of fragments the sender has.

The protocol keeps track of the number of fragments each user have and the rebase mechanism depends on the fragments concept to reward stakers through updating the number of fragments per token instead of updating the balances of all stakers.

Concern

The mint method in the staked-lydian-token contract does not associate the recipient with the minted tokens' fragments. This will prevent the recipient from transferring his staked lydians as the transfer method checks for the user token fragments. Steps leading to this issue:

1. **DAO mints staked lydians**

```
;; ---------------------------------------------------------
;; Mint / Burn
;; ---------------------------------------------------------

(define-public (mint (recipient principal) (amount uint))
  (begin
    (asserts! (is-eq contract-caller .lydian-dao) (err  ERR-NOT-AUTHORIZED))
    (ft-mint? staked-lydian amount recipient)
  )
)
```

2. **User transfer his staked lydian tokens**

```
(define-public (transfer (amount uint) (sender principal) (recipient principal) (memo
(optional (buff 34))))
  (let (
    (fragments-to-transfer (* amount (var-get fragments-per-token)))

    (sender-fragments (get fragments (get-account-fragments sender)))
    (recipient-fragments (get fragments (get-account-fragments recipient)))

    (new-sender-fragments (- sender-fragments fragments-to-transfer))
    (new-recipient-fragments (+ recipient-fragments fragments-to-transfer))

    (current-sender-balance (unwrap-panic (get-balance sender)))
    (new-sender-balance (- current-sender-balance amount))
    (sender-debt (get debt (get-debt-balance sender)))
  )
    (asserts! (var-get contract-is-enabled) (err ERR-CONTRACT-DISABLED))
```

```
    (asserts! (is-eq tx-sender sender) (err ERR-NOT-AUTHORIZED))
    (asserts! (>= new-sender-balance sender-debt) (err ERR-DEBT))
....
```

Transfer will not happen since the sender fragments is 0 as the mint method did not update the account-fragment map.

Recommendation

To fix this issue, the mint method should update the account-fragments map to include the recipient with the newly minted tokens fragments.

```
;; ---------------------------------------------------------
;; Mint / Burn
;; ---------------------------------------------------------

(define-public (mint (recipient principal) (amount uint))
  (let
    (
    (fragments (get fragments (get-account-fragments recipient)))
    (mint-fragments (/ amount (var-get fragments-per-token)))
    (new-fragment-balance (+ fragments mint-fragments))
    )
    (asserts! (is-eq contract-caller .lydian-dao) (err  ERR-NOT-AUTHORIZED))
    (map-set account-fragments { account: recipient } { fragments: new-fragment-balance })
    (ft-mint? staked-lydian amount recipient)
  )
)
```

0F-LDN-009 - Undocumented protocol administration entry point

Context

The protocol and its evolution is cleverly designed to be driven in a decentralized fashion by token holders, through a mechanism of proposals. Proposals are smart contracts that, when being approved by a majority of token holders, are being executed on the behalf of the community as a superuser.
The Lydian protocol includes a myriad of public functions and code paths that can only be executed if and only if it is executed by the DAO.

This design is providing a lot of flexibility to the protocol, and will require a certain due diligence by the token holders voting on proposals.

Concern

The governance contract includes a method that introduces a notion of ownership.

```
;; ----------------------------------------
;; Owner
;; ----------------------------------------

(define-public (execute-proposal (proposal-trait <lydian-dao-proposal-trait>))
  (begin
    (asserts! (is-eq tx-sender (var-get contract-owner)) (err  ERR-NOT-AUTHORIZED))
    (as-contract (contract-call? .lydian-dao execute-proposal proposal-trait))
  )
)
```

Per this method, the contract deployer would end up having the same rights and privileges on administering the protocol than a majority of voting community members, without going through the voting process.

Recommendation

We understand that in the early life of a protocol, granting special privileges to the protocol designer to perform certain actions in case of extraordinary events can be beneficial for every single participant involved in the protocol.

We would recommend constraining the access of this backdoor could be a good thing. We would recommend adding a comment to this method explaining how this endpoint will be used, constraining this access

with a time limit, emitting a dedicated event when this method is being used, and designating a multisig owner.

0F-LDN-010 - Token commitment when voting

Context

When protocol upgrades or enhancements are being emitted via proposals, token holders have the ability to vote for or against their execution. In order to participate to this voting events, users have to commit tokens, by sending them to the voting contract:

```
(define-public (vote (vote-for bool) (token <ft-trait>) (proposal-id uint) (amount uint))
...
    ;; Voter transfer tokens
    (try! (contract-call? token transfer amount tx-sender (as-contract tx-sender) none))
```

Once the proposal is expired or executed, voters can get back their tokens by invoking the following method:

```
(define-public (return-votes-to-member (token <ft-trait>) (proposal-id uint) (member
principal))
...
        ;; Return LDN or sLDN
        (as-contract (contract-call? token transfer token-count tx-sender member none))
```

Concern

This pattern is interesting and would be required if a punitive action (slashing mechanism or other) was at play.

This is not the case in this scenario. As such, we would recommend using a pattern that does not involve a transfer of tokens, which is an event that requires some trust.

Recommendation

Some other patterns could be explored. In this scenario, we want to avoid vote manipulation and tokens being spent twice for a given proposal.
We believe that this could be achieved by using the (at-block ...) construct. The contract would look at the balance of a voter at the block where the proposal was created, and use this amount to infer a voting power.

This could also be achieved with some tweaks at the token contract level, where users could be locking tokens for a vote. A quantity of tokens would be tainted and unusable, until the expiration of the lock.

0F-LDN-011 - General recommandations

1. When block-height is equal to the auction end-block, the protocol does not consider the auction neither open nor ended. Make sure the auction is open both on start and end blocks.
2. Remove warmup & claim methods from the staking-v1-1 contract as this functionality is not currently in use. Remove as well all the test methods from all the contracts.
3. The protocol traits do not include the main methods of the contracts implementing the traits. For example, the staking-trait-v1-1 trait does not include the unstake method of the staking-v1-1 contract. Same for the bond-teller-trait-v1-1 as it does not include the redeem and pending methods of the bond-teller-v1-1 contract.
4. Potential data discrepancies:
    a. The add-token method in the value-calculator contract adds tokens to the token-info map without checking if the token being added is a valid and enabled treasury reserve token
    b. The set-adjustment method in the staking-distributor-v1-1 adds recipient adjustments without checking if the recipient is a valid one. Make sure the recipient passed to the method is part of the recipient-info map.
5. Simplifying code:
    - In the tokens-claimable method in the auction-v1-1 contract, the get-commitments method is called twice:

```
(user-committed (get committed (get-commitments user auction-id)))
(user-claimed (get claimed (get-commitments user auction-id)))
Instead, call get-commitments once and then retrieve the committed and claimed properties
(user-commitments (get-commitments user auction-id))
(user-committed (get committed user-commitments))
(user-claimed (get claimed user-commitments))
```

    - In an auction, a user will always claim or withdraw all tokens at once. So (+ current-claimed claimable) could indeed be replaced by (claimable) in the both withdraw-tokens & withdraw-committed in the auction-v1-1 contract.
    - In the audit-reserve-token in the treasury-v1-1 contract, use the token variable instead of retrieving it again. Instead of the below:

```
(token (contract-of token-trait))
```

```
(token-map (get-reserve-token (contract-of token-trait)))
Use the following:
(token (contract-of token-trait))
(token-map (get-reserve-token token))
```

- In the redeem method in the bond-teller-v1-1 contract, bond-info is being deleted as per the below code:

```
(map-delete bond-info { id: bond-id })
```

It would be better to call instead the below existing method used in the redeem-all method:

```
(map-delete bond-info { id: bond-id })
```

- In the next-adjust-rate method in the staking-distributor-v1-1 contract, the get-recipient-adjust method is called several times.

```
(adjust-add (get add (get-recipient-adjust recipient)))
(adjust-rate (get rate (get-recipient-adjust recipient)))
(adjust-target (get target (get-recipient-adjust recipient)))
```

Instead, call the get-recipient-adjust method once and then retrieve the add, rate and target properties.

```
(recipient-adjust (get-recipient-adjust recipient))
(adjust-add (get add recipient-adjust))
(adjust-rate (get rate recipient-adjust))
(adjust-target (get target recipient-adjust))
```

- as-contract is being used redundantly in different places in the protocol. For example, in the unstake method of the staking-v1-1 contract the following code is used to transfer lydians from the contract to the staker:

```
(try! (as-contract (contract-call? .lydian-token transfer amount (as-contract tx-sender)
staker none)))
```

The inner as-contract is redundant as the outer one sets the tx-sender to the address of the calling contract. Instead, we can use the below:

```
(try! (as-contract (contract-call? .lydian-token transfer amount tx-sender staker none)))
```

- The claim method in the wrapped-lydian-creator-v1-1 contract is getting claim-amount first and if it is >u0, the claim-rebase is being called. No need for this step as the check for the claim amount is already done in the claim-rebase method itself.
- In the deposit method of the bond-depository-v1-1 contract, define a variable bond-capacity and set it to either payout or amount depending on the bond capacity being payout or not.
  Then replace the below code:

```
(if (get capacity-is-payout bond-type)
(begin
(asserts! (>= capacity payout) (err ERR-BOND-CAPACITY-REACHED))
               ;; set capacity and increase total debt
    (map-set bond-types { bond-id: bond-id } (merge bond-type {
      capacity: (- capacity payout),
      total-debt: (+ (get total-debt bond-type) token-value)
    }))
  )
  (begin
    (asserts! (>= capacity amount) (err ERR-BOND-CAPACITY-REACHED))

    ;; set capacity and increase total debt
    (map-set bond-types { bond-id: bond-id } (merge bond-type {
      capacity: (- capacity amount),
      total-debt: (+ (get total-debt bond-type) token-value)
    }))
  )
)
```

with the following:

```
(asserts! (>= capacity bond-capacity) (err ERR-BOND-CAPACITY-REACHED))
;; set capacity and increase total debt
(map-set
   bond-types
   { bond-id: bond-id }
   (merge bond-type {
     capacity: (- capacity bond-capacity),
     total-debt: (+ (get total-debt bond-type) token-value)
 }))
```

Conclusion

The findings explained in this report were discussed in detail with the Lydian team that was providing, in time, all the needed information that helped us assess the protocol smart contracts. The Lydian team has already fixed 5 major findings (0F-LDN-002, 0F-LDN-005, 0F-LDN-006, 0F-LDN-007, 0F-LDN-008) by the time we were writing the report as part of their efforts and commitment to properly secure the protocol.