t0ns dex
Audit Report

Tue Feb 03 2026

contact@bitslab.xyz https://twitter.com/tonbit_

TonBit

t0ns dex Audit Report

1 Executive Summary

1.1 Project Information

Description	This report has been commissioned by Tourbillon Labs GmbH (Zug, Switzerland) to identify any potential issues and vulnerabilities in the source code of the t0ns DEX smart contracts 1.0 version smart contract, as well as any contract dependencies that were not part of an officially recognized library. t0ns (https://t0ns.com) is a decentralized exchange (DEX) built on The Open Network (TON) blockchain. The protocol implements an automated market maker (AMM) model, enabling users to swap tokens in a fully on-chain, permissionless environment without relying on centralized intermediaries. Liquidity providers can supply assets to the pools and earn fees generated from trading activity, while traders benefit from continuous, algorithmic price discovery. The project introduces a unique NFT-based referral system. These NFTs represent and track referral relationships and rewards in a transparent, immutable, and verifiable manner.
Type	DeFi
Auditors	fishmen,N0n3
Timeline	Tue Oct 28 2025 - Tue Feb 03 2026

Languages	FunC
Platform	Ton
Methods	Architecture Review, Unit Testing, Manual Code Review
Source Code	https://github.com/tourbillonlabsio/smart-contracts
Commits	fd32c52e04dd4b08eb38f7fea4c9851b55a2012a c3def00fc44b5e6410adefd538ef47b15b067bcc

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
LIB1	contracts/libholder/libholder.fc	ef72c8f60cc56113690be6404a35dc7f0ffc8201
ENC	contracts/lp-wallet/encoder.fc	ab6b0991e27916fb7c2cf43af4d492ab5f300544
FCH	contracts/lp-wallet/fee-checks.fc	5596557b524cb95d9ddd303aa190937a1de836d3
LWA	contracts/lp-wallet/lp-wallet.fc	02e6fdb3b0d6fb3fb3ece4bd6f915aebde0cf995
ENC1	contracts/factory/encoder.fc	e915ae279eb26eb02738e81187988911a832ac75
FAC1	contracts/factory/factory.fc	c6b12b8c93820f6bc93da25b49623b5d982c1147
PRO1	contracts/referal-nft/proxy/proxy.fc	47f9842c66a00df43a9a1b36a74bf3a6ad7f25b4
PEN	contracts/referal-nft/proxy/proxy-encoder.fc	848ef440b251652a45845ed2932afe5960eea54a
NIE	contracts/referal-nft/nft-item/nft-item-encoder.fc	1d16f88d84ea4127ff88da006dc9c3cf4876c084
NIT1	contracts/referal-nft/nft-item/nft_item.fc	bad33bf8b3e2ea5a51f02cb10c89e5c0b66c14c0

NCV31	contracts/referal-nft/nft-collection/nft_collection_v3.fc	44bcd852a288d716676fb861f5495eb8fbf58c24
NCE	contracts/referal-nft/nft-collection/nft-collection-encoder.fc	a49bc4b1ba5f141ee26bc11626d5e8b3334dce5e
UTI	contracts/referal-nft/imports/utils.fc	e5ea38d98e4d38198649d498db4ad1c153234d11
ENC2	contracts/token-vault/encoder.fc	106d7e8304d472fc598902a6a916bcb81a736363
TVA1	contracts/token-vault/token-vault.fc	17b0d387a00901356ba6a1c6d9446f1029b3631e
ENC3	contracts/native-vault/encoder.fc	63dbc332638cbdbf6ee0c99107e092f8cdd018cc
NVA1	contracts/native-vault/native-vault.fc	01fd8276e14bfdc423d8806173f3071fb0092ff7
CON4	contracts/imports/context.fc	d8cea1556fce54baa36e692ec413a69b2801001e
ENC4	contracts/imports/encoder.fc	3eb29363b0bbc1c7d9f507ede22e766c43fa0a5b
JET	contracts/imports/jetton.fc	f48fff9149749a1707767717e0fbd2b4235d03b5
STD	contracts/imports/stdlib.fc	de650761d1fcef8b00a783a49a941dba53c4b196
HEL1	contracts/imports/helpers.fc	cc1b6780f015f54797c27f9d3e472def3c3fe388

MAT	contracts/imports/math/math.fc	ee0bf46986b92be496c1c5e433c83e39e4bde11a
FP	contracts/imports/math/fp.fc	0ad96b2330d3a67e8be60309fdf4bc84d3192f21
CON5	contracts/imports/constants.fc	6969df0b61c6ac1ceaf47863e5f1346486431947
ENC5	contracts/dex-pool/encoder.fc	b53b35a82dc7fb10dc49e7cdffea3b26600245fe
WMA	contracts/dex-pool/weighted-math.fc	3909cc4f2fa7fc499ed3a115f1f70d8317cdef6c
SMA	contracts/dex-pool/stable-math.fc	7c3c2a6fefb251c48d2e6dbb558c19a0a9a784df
DPO1	contracts/dex-pool/dex-pool.fc	4ee2c7beacf25bde47bc8deab4c47e6ec32154c7
LHO1	contracts/liq-hold/liq-hold.fc	f97e4622eb22bba4e2e44e3b02dbafc1f43b087d
ENC6	contracts/liq-hold/encoder.fc	9b269f38015f04fee9bd0ecb9d622ab4953e7e2b
NCV31	contracts/referal-nft/nft-collection/nft_collection_v3.fc	2216c66ff5c3a7e15e7f48a9daa93ab76abfe484
ENC1	contracts/factory/encoder.fc	40bbf94d2a551777029a1e92eb9d8cefd46a72fd
FAC1	contracts/factory/factory.fc	252b252e809eab80eccd7a1bee13585bd33ed935

CON5	contracts/imports/constants.fc	50384e93a467f2313f5dd08e4f56a6c41c85de89
ENC2	contracts/token-vault/encoder.fc	cad2c96b67805aa23b8b4f5a6f7fb990e76e8264
ENC5	contracts/dex-pool/encoder.fc	9a8c53d0fcc8427e075194d2fd2227c0300f5e59
TVA1	contracts/token-vault/token-vault.fc	fc082264d48052a6a0419beeacfa2fc2a55e0f3d
NVA1	contracts/native-vault/native-vault.fc	52685130f6d7f297f9081812d4767b8b1bc5df7a
CON4	contracts/imports/context.fc	01295f5ce7c230f20f5c9da9650431ab80d08097
HEL1	contracts/imports/helpers.fc	c629b4cbfbc2ef65772e0a6291fcc009c2c8e69b
DPO1	contracts/dex-pool/dex-pool.fc	aa9a02b364e994ad1069e400a5519baa5976180f

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	16	14	2
Critical	2	2	0
Major	7	7	0
Medium	3	2	1
Minor	3	2	1
Informational	1	1	0

1.4 TonBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Manual Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues. All issues identified during the audit have been confirmed or resolved, with the final secure state represented by Commit c3def00fc44b5e6410adefd538ef47b15b067bcc, which constitutes the **smart contract version 1.0**.

During the audit, we identified 16 issues of varying severity, listed below.

ID	Title	Severity	Status
DPO-1	Unlocked Initial Liquidity Enables Manipulation of Subsequent Liquidity Calculations	Major	Fixed
DPO-3	Insufficient Verification of `ctx::value` to Cover Gas Costs	Major	Fixed
DPO-4	Missing bounds check before subtraction	Critical	Fixed
DPO-5	Lack of Parameter Validation in the `constructor`	Major	Fixed
DPO-7	User-Controlled `is_ref?` Flag Allows Bypassing Recommendation Fee Payments	Minor	Acknowledged
DPO-8	`pool` Can Be Created Using the Same Token	Major	Fixed
DPO-12	Precision Loss Issues	Medium	Acknowledged
DPO-13	Missing Slippage Protection	Major	Fixed

DPO-14	Useless Parameter in `send_deposit_liquidity_single_after_swap` Function	Informational	Fixed
DPO-16	Parameter Mixing Issue in `swap_external` Function	Medium	Fixed
DPO-23	Bounce Message Truncation Causes on_bounce Handler Failure	Major	Fixed
FAC-10	No Validity Check for All Values in `factory`	Major	Fixed
LHO-17	Incorrect Handling of `_token_deposits` in `return_asset` Function	Critical	Fixed
NCV-18	Lack of Validity Check for `amount` Parameter in `deploy_new_nft` Function	Medium	Fixed
NCV-21	`get_royalty_params` Function Failure Caused by Missing Data Loading	Minor	Fixed
TVA-22	Cell Overflow Causing Multihop Swap Failure with Non-zero `minOut`	Minor	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the t0ns DEX smart contracts 1.0 version Smart Contract :
This is a DEX pool where users can add or remove liquidity by providing funds. Users can also swap assets between different pools to obtain the assets they desire. Additionally, the contract features NFT functionality, allowing users to purchase NFTs.

4 Findings

DPO-1 Unlocked Initial Liquidity Enables Manipulation of Subsequent Liquidity Calculations

Severity: Major

Status: Fixed

Code Location:

contracts/dex-pool/dex-pool.fc#225-249

Descriptions:

When a user first adds liquidity, the base 1000 liquidity should be locked in the pool rather than being deducted directly without being added to the pool. This deviates from Uniswap's design and increases the potential for subsequent liquidity manipulation, posing a risk of asset loss.

```
int deposit_liquidity(slice msg_body) impure inline {
    ...

    if (_pool_type == pool_type::stable) {
        int D_old = calculate_invariant(_amplification, r0, r1);
        int D_new = calculate_invariant(_amplification, r0 + used_td0, r1 + used_td1);

        if (_supply_lp == 0) {
            liquidity = D_new - LIQUIDITY_SUB;//@audit
        } else {
            int dD = D_new - D_old;
            liquidity = ((dD > 0) & (D_old > 0)) ? muldiv(_supply_lp, dD, D_old) : 0;
        }
    } elseif (_pool_type == pool_type::w_stable) {
        int p = _w0;
        int q = _w0.math::fp::complement();

        int K_old = _invariant(_rate, _amplification, r0, r1, p, q);
        int K_new = _invariant(_rate, _amplification, r0 + used_td0, r1 + used_td1, p, q);
```

```
    if (_supply_lp == 0) {
        liquidity = math::fp::to(K_new) - LIQUIDITY_SUB;//@audit
    } else {
        int dK = K_new - K_old;
        int dK_int  = math::fp::to(dK);
        int K_old_int = math::fp::to(K_old);

        liquidity = ((dK > 0) & (K_old > 0) & (dK_int > 0) & (K_old_int > 0))
            ? muldiv(_supply_lp, dK_int, K_old_int) : 0;
    }
  } else {
    if (_supply_lp == 0) {
        liquidity = h::sqrt(td'b.pair_first() * td'b.pair_second()) - LIQUIDITY_SUB; //@audit
    } else {
        int x = muldiv(td'b.pair_first(), _supply_lp, r0);
        int y = muldiv(td'b.pair_second(), _supply_lp, r1);

        liquidity = min(x, y);

        ...
  mint_lp(depositor'b, liquidity);
  return true;
}
```

here is the uniswapv2 version

```
108
109         // this low-level function should be called from a contract which performs important safety checks
110         function mint(address to) external lock returns (uint liquidity) {
111             (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
112             uint balance0 = IERC20(token0).balanceOf(address(this));
113             uint balance1 = IERC20(token1).balanceOf(address(this));
114             uint amount0 = balance0.sub(_reserve0);
115             uint amount1 = balance1.sub(_reserve1);
116
117             bool feeOn = _mintFee(_reserve0, _reserve1);
118             uint _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply can update in _mintFee
119             if (_totalSupply == 0) {
120                 liquidity = Math.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);
121                _mint(address(0), MINIMUM_LIQUIDITY); // permanently lock the first MINIMUM_LIQUIDITY tokens
122             } else {
123                 liquidity = Math.min(amount0.mul(_totalSupply) / _reserve0, amount1.mul(_totalSupply) / _reserve1);
124             }
125             require(liquidity > 0, 'UniswapV2: INSUFFICIENT_LIQUIDITY_MINTED');
126             _mint(to, liquidity);
127
128             _update(balance0, balance1, _reserve0, _reserve1);
129             if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0 and reserve1 are up-to-date
130             emit Mint(msg.sender, amount0, amount1);
131         }
132
```

Suggestion:

The initial 1,000 liquidity tokens remain in the pool as a donation rather than being directly burned.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

DPO-3 Insufficient Verification of `ctx::value` to Cover Gas Costs

Severity: Major

Status: Fixed

Code Location:

contracts/dex-pool/dex-pool.fc

Descriptions:

When executing the calculation `ctx::value - my_storage_due()`, failure to validate whether `ctx::value` is greater than or equal to `my_storage_due()` resulted in the computed `value` potentially being negative.

Suggestion:

Add corresponding checks for `ctx::value`

Resolution:

This issue has been fixed. The client has adopted our suggestions.

DPO-4 Missing bounds check before subtraction

Severity: Critical

Status: Fixed

Code Location:

contracts/dex-pool/dex-pool.fc

Descriptions:

In `compute_local_values`, the expression `tr1 -= (base_out + protocol_fee_out + ref_fee_out)` is executed without ensuring that `tr1` is at least the sum of the subtrahend. Several subtraction operations elsewhere in the project similarly lack bounds checks. If the subtrahend exceeds `tr1`, the arithmetic can produce negative values or wrap/underflow depending on the runtime semantics.

Suggestion:

We recommend adding checks to ensure that the minuend is greater than or equal to the subtrahend before performing subtraction operations.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

DPO-5 Lack of Parameter Validation in the `constructor`

Severity: Major

Status: Fixed

Code Location:

contracts/dex-pool/dex-pool.fc

Descriptions:

The `_lp_fee` constructor parameter is not validated. Because later calculations perform `factor - _lp_fee`, an out-of-range `_lp_fee` (e.g., > `factor` or > 10000 if `factor` is 10000) can produce negative or otherwise invalid results that corrupt fee computations. This issue also exists elsewhere.

Suggestion:

We recommend implementing validation checks for parameters within the `constructor` to ensure they fall within acceptable ranges

Resolution:

This issue has been fixed. The client has adopted our suggestions.

DPO-7 User-Controlled `is_ref?` Flag Allows Bypassing Recommendation Fee Payments

Severity: Minor

Status: Acknowledged

Code Location:

contracts/dex-pool/dex-pool.fc

Descriptions:

In the `compute_outputs` function, the `ref_fee` is deducted from the user's funds rather than allocated from `_protocol_fee`. Since `is_ref?` is controlled by the user, a malicious user could set `is_ref?` to false to refuse paying the referral fee, conflicting with the protocol's design logic.

```
 if (is_ref?) {
     throw_unless(error::ref_fee_out_of_ranges, (ref_fee >= ref_fee::min) & (ref_fee <=
ref_fee::max));
     ref_fee_out = h::ufits120(muldiv(base_out, ref_fee, factor));
     base_out -= ref_fee_out;
  }
```

Suggestion:

Recommendation to change the method of collecting referral fees

Resolution:

The team acknowledged this issue and stated that the `is_ref?` flag is intentionally user-controlled to allow explicit referral-channel selection, and bypassing it is equivalent to opting out of the referral program rather than introducing any pool risk, so no modifications will be made to the current version.

DPO-8 `pool` Can Be Created Using the Same Token

Severity: Major

Status: Fixed

Code Location:

contracts/dex-pool/dex-pool.fc

Descriptions:

In the `constructor` function of `dex-pool.fc`, there is no validation to ensure that the two tokens provided are distinct. This oversight allows for the creation of a `pool` using identical tokens, which may lead to unintended behaviors within the system.

Suggestion:

We recommend implementing a validation check in the `constructor` to ensure that the two tokens provided are different, thereby preventing the creation of a `pool` with the same token.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

DPO-12 Precision Loss Issues

Severity: Medium

Status: Acknowledged

Code Location:

contracts/dex-pool/dex-pool.fc;

contracts/dex-pool/weighted-math.fc

Descriptions:

1. In the `deposit_liquidity` function, when `_pool_type == pool_type::w_stable`, the liquidity calculation does not consider precision loss. The floating-point numbers are directly converted to integers, which can lead to precision loss and affect the user's liquidity share distribution.

```
int dK_int  = math::fp::to(dK);
int K_old_int = math::fp::to(K_old);

liquidity = ((dK > 0) & (K_old > 0) & (dK_int > 0) & (K_old_int > 0))
  ? muldiv(_supply_lp, dK_int, K_old_int) : 0;
```

2. In the `deposit_liquidity function`, at lines 258 and 272, the calculation of `used_td*` should add 1 to prevent precision loss.

```
        if (x < y) {
            used_td0 = td'b.pair_first();
            used_td1 = muldiv(liquidity, r1, _supply_lp); ;;@audit need +1 to prevent
precision loss
            refund_td1 = td'b.pair_second() - used_td1;
            ...
        } else {
            used_td1 = td'b.pair_second();
            used_td0 = muldiv(liquidity, r0, _supply_lp); ;;@audit need +1 to prevent
```

```
precision loss
            refund_td0 = td'b.pair_first() - used_td0;
```

3. In the `deposit_liquidity_single` function, at lines 374-375, there is also a precision loss issue, especially when `_supply_lp` is small.

```
int used_td0 = muldiv(liquidity, _token_reserves.pair_first(), _supply_lp);
int used_td1 = muldiv(liquidity, _token_reserves.pair_second(), _supply_lp);
```

4. In the `weighted-math.fc` file, in the `solve_dy` function, the final return value does not account for precision loss.

```
int solve_dy(int w0, int rate, int amp, int x, int y, int dx) impure inline {
int p = w0;
int q = w0.math::fp::complement();

int start_y = y;
int K0 = _invariant(rate, amp, x, y, p, q);
int i = 0;
int delta_f = 0;

do {
    delta_f = _invariant(rate, amp, x + dx, start_y, p, q) - K0;
    int df_d = _dy_invariant(rate, amp, x + dx, start_y, p, q);

    throw_if(error::df_zero, df_d == 0);
    start_y -= math::fp::to(math::fp::div(delta_f, df_d));

    i += 1;
    throw_if(error::not_converge, i > __MAX_ITERATIONS);
} until (abs(delta_f) <= __EPSILON);

return (y - start_y);
}
```

Suggestion:

1. To avoid precision loss, consider preserving the floating-point precision of `dK` and `K_old_int` during liquidity calculations.
2. Add +1 to the result of `used_td*` in `deposit_liquidity`
3. Increment `used_td*` by one or adopt a more appropriate method to reduce precision loss
4. For the final return value, consider reducing the error through methods like decrementing to ensure consistency with `calc_out_given_in` when `pool_type::stable`

Resolution:

The team acknowledged this issue and stated that the precision loss only becomes significant for tiny liquidity amounts and is consistent with industry-standard downward-rounding behavior, so no modifications will be made to the current version.

DPO-13 Missing Slippage Protection

Severity: Major

Status: Fixed

Code Location:

contracts/dex-pool/dex-pool.fc

Descriptions:

In the `deposit_liquidity` and `deposit_liquidity_single` functions, there is a lack of slippage protection for the expected slippage from the user. When users add liquidity, the price fluctuations may result in the actual liquidity shares being different from the expected value, thus affecting the user's fund security and trading experience.

Suggestion:

It is recommended to implement a slippage protection mechanism in the `deposit_liquidity` and `deposit_liquidity_single` functions to ensure that users can add liquidity within their expected slippage range, thereby enhancing fund security and user experience.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

DPO-14 Useless Parameter in `send_deposit_liquidity_single_after_swap` Function

Severity: Informational

Status: Fixed

Code Location:

contracts/dex-pool/dex-pool.fc

Descriptions:

In the `send_deposit_liquidity_single_after_swap` function, the `salt` parameter is assigned a value from `now()`, but it is not used in the `deposit_liquidity_single` function. Therefore, the existence of this parameter is meaningless, and it increases the complexity of the code.

Suggestion:

It is recommended to remove the `salt` parameter from the `send_deposit_liquidity_single_after_swap` function to simplify the code logic and reduce gas consumption.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

DPO-16 Parameter Mixing Issue in `swap_external` Function

Severity: Medium

Status: Fixed

Code Location:

contracts/dex-pool/dex-pool.fc

Descriptions:

In the `swap_external` function, under the branch `final_action'b == final_action::withdraw_liquidity`, the `amount'b` parameter is used both as the input amount for the swap and passed to `send_withdraw_liquidity_after_swap` as the `lp_amount` parameter. This mixing of parameters may lead to logical confusion, and users may misunderstand the meaning of the parameter, which could affect the correctness of the transaction and fund security.

Suggestion:

It is recommended to review the parameter design in this function to ensure that each parameter has a clear purpose and avoid logical confusion caused by mixing parameters.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

DPO-23 Bounce Message Truncation Causes on_bounce Handler Failure

Severity: Major

Status: Fixed

Code Location:

contracts/dex-pool/dex-pool.fc

Descriptions:

The `on_bounce` function in `dex-pool.fc` is responsible for handling bounced messages during multi-hop swap operations. When a `swap_internal` message is rejected by the next pool in a multi-hop swap chain, this handler attempts to recover the user's funds by parsing the bounced message and redirecting the tokens back to the user.

However, the implementation attempts to parse `amount`, `destination`, and `token_vault_prev` from the bounced message body. According to TON's default bounce mechanism (https://docs.ton.org/foundations/messages/internal#default-bounce), only the first 256 bits of the original message body are returned in a bounce message. After subtracting 64 bits for `query_id`, only 192 bits remain available for parsing. Since parsing a single TON address requires 267 bits, the handler fails with a cell underflow error when attempting to parse the `destination` address.

This issue results in the `on_bounce` handler failing to execute properly during multi-hop swap failures, causing user funds to become stuck in an intermediate pool without automatic recovery.

Suggestion:

It is recommended to consider utilizing the new bounce format with `extra_flags` to return the full message body to resolve this issue.

Resolution:

The team adopted our advice and fixed this issue by adding support for the new bounce format. Additionally, corresponding test cases were added to verify the bounce handling behavior. The fix can be found at commit `c3def00fc44b5e6410adefd538ef47b15b067bcc`.

FAC-10 No Validity Check for All Values in `factory`

Severity: Major

Status: Fixed

Code Location:

contracts/factory/factory.fc

Descriptions:

The `factory` does not perform validity checks on the parameters passed in, allowing negative or unreasonable values to be set, which can affect subsequent calculation logic.

- Although the factory includes some checks, it is not necessary to create a pool through the factory.The constructor of the pool also does not check the validity of parameters, which allows for the deployment of pool contracts with unreasonable parameters.
- `deploy_dex_pool` does not sort `token_vaults'b` using `h::sort_addrs_by_hash`, and the calculation logic for `dex-pool` assumes that `token_vaults'b` is ordered, leading to potential consistency issues.

Suggestion:

- It is recommended to add validity checks in both the `factory` and the pool constructor to ensure that the parameters are within a reasonable range and prevent negative or unreasonable values.
- Ensure that the constructor can only be called by the factory.
- Sort `token_vaults'b` to maintain consistency.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LHO-17 Incorrect Handling of `_token_deposits` in `return_asset` Function

Severity: Critical

Status: Fixed

Code Location:

contracts/liq-hold/liq-hold.fc

Descriptions:

In the `liq-hold` contract's `return_asset` function, the `_token_deposits` are not properly deducted for the refunded amount, which could allow users to withdraw funds multiple times, affecting the security of the funds.

Suggestion:

It is recommended to correctly deduct the refunded amount from `_token_deposits` in the `return_asset` function, ensuring that users can only withdraw the funds they have actually deposited, preventing security issues related to multiple withdrawals.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

NCV-18 Lack of Validity Check for `amount` Parameter in `deploy_new_nft` Function

Severity: Medium

Status: Fixed

Code Location:

contracts/referal-nft/nft-collection/nft_collection_v3.fc

Descriptions:

In the `nft_collection_v3.fc` file, the `deploy_new_nft` function lacks a validity check for the `amount` parameter, which may allow negative or unreasonable values to be passed in, affecting subsequent calculation logic and the normal operation of the contract.

Suggestion:

It is recommended to add a validity check for the `amount` parameter in the `deploy_new_nft` function to ensure that the parameter is within a reasonable range and to prevent negative or unreasonable values from being passed in.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

NCV-21 `get_royalty_params` Function Failure Caused by Missing Data Loading

Severity: Minor

Status: Fixed

Code Location:

contracts/referal-nft/nft-collection/nft_collection_v3.fc#226-29

Descriptions:

The `get_royalty_params` function in the `nft_collection_v3.fc` contract is responsible for retrieving and sending royalty parameters to the caller. This function relies on the global variable `_royalty_params` to access the stored royalty configuration data.

However, in the original implementation, when `get_royalty_params` was invoked through the message handler, the global variable `_royalty_params` was not initialized before use. The function attempted to access `_royalty_params` directly without first calling the `load_data()` function, which is responsible for loading all global variables from the contract storage.

This missing initialization step would cause the `get_royalty_params` function to fail immediately upon invocation, as it would attempt to access uninitialized data. This prevents callers from successfully retrieving the royalty parameters, effectively breaking this functionality of the NFT collection contract.

Suggestion:

It is recommended to ensure that `load_data()` is called before any function that relies on global variables. This can be achieved by placing the `load_data()` call at the beginning of the message routing logic, ensuring all global variables are properly initialized before any operation handlers are invoked.

Resolution:

The team adopted our advice and fixed this issue by adding the `load_data()` function call at the beginning of the `selector` function, which ensures all global variables are properly loaded before any operation handler, including `get_royalty_params`, is called. This can be found at the latest commit(d8840c14a1917865224fb6c46bf01d91d5ce007d).

TVA-22 Cell Overflow Causing Multihop Swap Failure with Non-zero `minOut`

Severity: Minor

Status: Fixed

Code Location:

wrappers/interfaces/TokenVault.ts

Descriptions:

The DEX protocol supports multihop swap functionality, which allows users to execute a series of token swaps across multiple liquidity pools in a single transaction. Each swap step is represented by a `SwapStep` structure containing swap parameters, including pool type, token vault addresses, destination, amount, minimum output (`minOut`), deadline, and reference to the next step.

During testing, the team discovered that multihop swaps failed with a "Cell overflow" error (exit code 8) when using non-zero `minOut` values (such as `minOut = 1n`), while the same operations succeeded with `minOut = 0n` . This issue was particularly critical for multihop swaps involving 2 or more steps.

The root cause of this issue lies in the encoding method used in `TokenVault.createSwapPayload` . The function used `storeSlice()` to embed the entire `SwapStep` cell contents directly into the message body, rather than passing it as a reference:

```
static createSwapPayload(swapStep: SwapStep): Cell {
    return beginCell()
        .storeUint(0x02, 8)
        .storeSlice(swapStep.toCell().beginParse()) // Embeds contents directly
    .endCell();
}
```

When using `storeSlice()`, all swap step data (including nested references to subsequent steps) must fit within a single cell. In TON blockchain, each cell has a maximum capacity of 1023 bits (approximately 128 bytes). The difference in VarUInteger16 encoding between `minOut = 0n` (4 bits) and `minOut = 1n` (12 bits) adds 8 bits per step. While this increase appears small, when combined with the overhead of embedding all step data directly, it causes the cell to exceed the 1023-bit limit, resulting in overflow errors.
This issue prevents users from executing multihop swaps with any slippage protection (`minOut > 0`), forcing them to either use single-step swaps or accept unlimited slippage risk by setting `minOut = 0n`. This significantly impacts the protocol's usability and exposes users to potential value loss from slippage.

Suggestion:

It is recommended to modify the encoding scheme to use cell references instead of direct embedding. Specifically, the `TokenVault.createSwapPayload` function should use `storeRef()` to pass the `SwapStep` as a reference, which only consumes 1 bit for the reference flag rather than embedding the entire cell contents. Correspondingly, the receiving functions in the smart contracts (`init_swap` in `token-vault.fc` and `swap_internal` in `dex-pool.fc`) should be updated to load the swap step from the reference using `load_ref()`. Additionally, implementing cell depth checks is recommended to prevent potential depth-based denial of service attacks.

Resolution:

The team identified this issue during testing and has fixed it by refactoring the encoding scheme across three contract files. The changes include:

1. Modified `dp::e_swap_internal_with_builder` in `encoder.fc` to use `store_ref(swap_step_cell)` instead of embedding contents directly
2. Updated `swap_internal` function in `dex-pool.fc` to load swap step from reference via `msg_body~load_ref().begin_parse()`.

3. Updated `init_swap` function in `token-vault.fc` to load swap step from reference and added cell depth check (`cell_depth() > tv::max_payload_depth`) to prevent depth attacks These changes allow the protocol to support multihop swaps of any reasonable size with non-zero `minOut` values without encountering cell overflow errors, while maintaining security through proper depth validation. This can be found at 68c4a5388298259ddcbfb9dbe2f2b5ce46b50cda

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.