Sienna.Network AMM Protocol

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **October 15th, 2021 - November November 12th, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	10/15/2021	Luis Quispe Gonzales
0.2	Document Updates	11/05/2021	Luis Quispe Gonzales
0.3	Draft Version	11/12/2021	Luis Quispe Gonzales
0.4	Draft Review	11/19/2021	Gabi Urrutia
1.0	Remediation Plan	12/29/2021	Luis Quispe Gonzales
1.1	Remediation Plan Review	12/30/2021	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Luis Quispe Gonzales	Halborn	Luis.QuispeGonzales@halborn.com

EXECUTIVE OVERVIEW

1.1 AUDIT SUMMARY

Sienna.Network engaged Halborn to conduct a security assessment on CosmWasm smart contracts beginning on October 15th, 2021 and ending November 12th, 2021.

The security engineers involved on the audit are blockchain and smart contract security experts with advanced penetration testing, smart contract hacking, and in-depth knowledge of multiple blockchain protocols.

The purpose of this audit is to achieve the following:

- Ensure that smart contract functions work as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were mostly addressed by Sienna.Network team. The main ones are the following:

- Update signature validation to handle blank signature cases.
- Harden factory and exchange contracts to restrict the creation of pools with the same pair.
- Include validation routines to limit the changes of fee rates in factory contract.
- Handle the case where a pool has no deposits and slippage is specified when users add liquidity.
- Enforce the use of a default maximum threshold when users add liquidity or swap.
- Split admin address transfer functionality to allow transfer to be completed by recipient.

External threats, such as financial related attacks, oracle attacks, and inter-contract functions and calls should be validated for expected logic and state.

1.2 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual review of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture, purpose, and use of the platform.
- Manual code read and walkthrough.
- Manual assessment of use and safety for the critical Rust variables and functions in scope to identify any contracts logic related vulnerability.
- Fuzz testing (Halborn custom fuzzing tool)
- Checking the test coverage (cargo tarpaulin)
- Scanning of Rust files for vulnerabilities (cargo audit)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.

3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.3 SCOPE

1. CosmWasm Smart Contracts
   (a) Repository: https://github.com/SiennaNetwork/sienna
   (b) Commit ID: 13ce1ac9728e16b3d79d64caca603fe029882371
   (c) Contracts in scope:
      i. amm-snip20
      ii. exchange
      iii. factory
      iv. lp-token

Out-of-scope: External libraries and financial related attacks

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	2	4	2	2

LIKELIHOOD

IMPACT

	(HAL-03)		(HAL-01)	
(HAL-07)	(HAL-05) (HAL-06)		(HAL-02)	
(HAL-09)				(HAL-04)
(HAL-10)		(HAL-08)		

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) SIGNATURE VALIDATION CAN BE BYPASSED	High	SOLVED - 11/01/2021
(HAL-02) POSSIBILITY TO CREATE POOLS WITH THE SAME PAIR	High	SOLVED - 11/16/2021
(HAL-03) UNRESTRICTED CHANGES IN FEE RATES LEAD TO TOKENS LOSS / DOS	Medium	RISK ACCEPTED
(HAL-04) ADDING LIQUIDITY TO NEW POOLS DOES NOT WORK PROPERLY	Medium	SOLVED - 12/23/2021
(HAL-05) MAXIMUM THRESHOLD FOR SLIPPAGE IS NOT ENFORCED WHEN ADDING LIQUIDITY OR SWAPPING	Medium	RISK ACCEPTED
(HAL-06) PRIVILEGED ADDRESS CAN BE TRANSFERRED WITHOUT CONFIRMATION	Medium	SOLVED - 12/13/2021
(HAL-07) EXCHANGES MIGRATION MECHANISM IS NOT COMPLETE	Low	SOLVED - 12/29/2021
(HAL-08) POSSIBILITY TO CREATE FAKE PAIRS WITH NATIVE COINS	Low	SOLVED - 12/29/2021
(HAL-09) PARTIALEQ FOR TOKENPAIR IS WRONGLY IMPLEMENTED	Informational	SOLVED - 10/26/2021
(HAL-10) SPREAD AMOUNT IS CALCULATED BUT NOT USED	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) SIGNATURE VALIDATION CAN BE BYPASSED - HIGH

Description:

ensure_correct_signature function in **contracts/factory/src/contract.rs** is the responsible to validate temporary signatures in **factory** contract and ensure the following scenarios:

1. **Exchanges** can be created by any user through create_exchange function in **factory** contract. Other methods like initializing **exchange** contract directly would not work because of signature validation.
2. **IDOs** can be created by admin or whitelisted users through create_ido function in **factory** contract. Other methods like initializing **ido** contract directly would not work because of signature validation.
3. **Launchpad** can be created once (if it does not exist yet) by admin through create_launchpad function in **factory** contract. Other methods like initializing **launchpad** contract directly would not work because of signature validation.

However, an attacker can completely bypass signature validation using '' (no space) as a signature because of incorrect use of **unwrap_or_default** in ensure_correct_signature function. This situation can produce the following consequences:

- (Malicious) exchanges can be created with valid pair of tokens using register_exchange function in **factory** contract, which could be harmful for users because the creator of a fake exchange could completely withdraw tokens from the contract and steal users deposits.
- Although these malicious exchanges would not appear directly in Sienna webpage because of filters used in their backend, they will appear as legitimate ones if other users query **factory** contract about existent exchanges using list_exchanges function.

- On the other hand, once a (malicious) exchange is created with a specific pair of tokens, no one can create a valid exchange with those tokens anymore because they remain as already registered in **factory** contract storage.
- Any user can create **(malicious) IDOs** using register_ido function in **factory** contract, completely bypassing admin / whithelisted user authorization checks. This newly created IDOs will appear as legitimate ones if other users query **factory** contract about existent IDOs using list_idos function.
- Any user can create a **(malicious) launchpad** using register_launchpad function in **factory** contract, completely bypassing admin authorization check. This newly created launchpad will appear as legitimate one if other users query **factory** contract about launchpad address using get_launchpad_address function. Besides, once the attacker creates the (malicious) launchpad, admin will not be able to create a valid one.

A proof of concept video showing how to exploit this security issue is included in the report.

Code Location:

Listing 1: contracts/factory/src/contract.rs (Lines 544)

```
543 fn ensure_correct_signature(storage: &mut impl Storage, signature:
        Binary) -> StdResult<()> {
544     let stored_signature: Binary = load(storage,
            EPHEMERAL_STORAGE_KEY)?.unwrap_or_default();
545
546     if stored_signature != signature {
547         return Err(StdError::unauthorized());
548     }
549
550     remove(storage, EPHEMERAL_STORAGE_KEY);
551
552     Ok(())
553 }
```

Risk Level:

Likelihood - 4
Impact - 5

Recommendations:

Update the logic of ensure_correct_signature function to use **ok_or_else** instead of **unwrap_or_default** to recover stored signature.

Remediation plan:

SOLVED: The issue was fixed in commit b6290c639ba4ef02fea97313d5154e800b809cb0.

3.2 (HAL-02) POSSIBILITY TO CREATE POOLS WITH THE SAME PAIR - HIGH

Description:

init function in **contracts/exchange/src/contract.rs** and create_exchange function in **contracts/factory/src/contract.rs** allow the possibility to create pools with the same pair, which generates unexpected situations, e.g.: a user could withdraw more tokens than his fair share and affect other users in the pool.

This issue happens because when a pair is compared with another one, if their contract_addr are the same but their token_code_hash differ just in their upper / lower cases, these pairs will appear as different values.

A proof of concept video showing how to exploit this security issue is included in the report.

Code Location:

Listing 2: contracts/exchange/src/contract.rs (Lines 53)

```
48 pub fn init<S: Storage, A: Api, Q: Querier>(
49     deps: &mut Extern<S, A, Q>,
50     env: Env,
51     msg: InitMsg,
52 ) -> StdResult<InitResponse> {
53     if msg.pair.0 == msg.pair.1 {
54         return Err(StdError::generic_err(
55             "Trying to create an exchange with the same token.",
56         ));
57     }
```

Listing 3: contracts/factory/src/contract.rs (Lines 175)

```
169 pub fn create_exchange<S: Storage, A: Api, Q: Querier>(
170     deps: &mut Extern<S, A, Q>,
171     env: Env,
```

```
    pair: TokenPair<HumanAddr>,
    entropy: Binary,
) -> StdResult<HandleResponse> {
    if pair.0 == pair.1 {
        return Err(StdError::generic_err(
            "Cannot create an exchange with the same token.",
        ));
    }
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendations:

Update the logic of init and create_exchange functions to compare pairs only by their contract_addr value.

Remediation plan:

SOLVED: The issue was fixed in commit 57673cbe2aa9777b574095b2a68f8f7f4e792027. The Sienna.Network team updated the logic of PartialEq implementation for TokenType in **libraries/amm-shared/src/token_type.rs** to make comparisons of pairs only by contract_addr or denom values (depending on token type).

3.3 (HAL-03) UNRESTRICTED CHANGES IN FEE RATES LEAD TO TOKENS LOSS / DOS - MEDIUM

Description:

init and set_config functions in **contracts/factory/src/contract.rs** change the values of all fields in **ExchangeSettings** directly, so do not restrict that values of swap_fee and sienna_fee are greater or equal than a **maximum threshold**. This situation can produce the following consequences:

- A malicious (or compromised) admin can change temporarily sienna_fee to a very high value, e.g.: 99/100, and transfer all **high commissions** generated in swapping operations to his account (sienna_burner).
- If **total fee** (swap_fee + sienna_fee) exceeds 1, swapping operations will always panic, thus generating a denial of service (DoS) in Sienna.Network protocol.

Code Location:

ExchangeSettings struct contains swap_fee and sienna_fee fields, whose type is **Fee** struct:

Listing 4: libraries/amm-shared/src/exchange.rs (Lines 38,39)

```
pub struct ExchangeSettings<A> {
    pub swap_fee: Fee,
    pub sienna_fee: Fee,
    pub sienna_burner: Option<A>,
}
```

Fee struct represents a fraction with **nom** and **denom** fields:

Listing 5: libraries/amm-shared/src/exchange.rs (Lines 73,74)

```
72 pub struct Fee {
73     pub nom: u8,
74     pub denom: u16,
75 }
```

init function calls from_init_msg function, which does not restrict that initial values of swap_fee and sienna_fee are greater or equal than a **maximum threshold**:

Listing 6: contracts/factory/src/contract.rs (Lines 46)

```
37 pub fn init<S: Storage, A: Api, Q: Querier>(
38     deps: &mut Extern<S, A, Q>,
39     env: Env,
40     msg: InitMsg,
41 ) -> StdResult<InitResponse> {
42     let admin = msg.admin.clone().unwrap_or(env.message.sender);
43     save_admin(deps, &admin)?;
44 
45     save_prng_seed(&mut deps.storage, &msg.prng_seed)?;
46     save_config(deps, &Config::from_init_msg(msg))?;
47 
48     Ok(InitResponse::default())
49 }
```

Listing 7: contracts/factory/src/state.rs (Lines 48)

```
40 impl Config<HumanAddr> {
41     pub fn from_init_msg(msg: InitMsg) -> Self {
42         Self {
43             snip20_contract: msg.snip20_contract,
44             lp_token_contract: msg.lp_token_contract,
45             pair_contract: msg.pair_contract,
46             launchpad_contract: msg.launchpad_contract,
47             ido_contract: msg.ido_contract,
48             exchange_settings: msg.exchange_settings,
49         }
```

set_config function does not restrict that new values of swap_fee and sienna_fee are greater or equal than a **maximum threshold**:

Listing 8: contracts/factory/src/contract.rs (Lines 133)

```
if let Some(new_value) = exchange_settings {
    config.exchange_settings = new_value;
}
```

Risk Level:

Likelihood - 2
Impact - 5

Recommendations:

Add a validation routine inside init and set_config functions to ensure that value of sienna_fee is lesser than a **maximum threshold** hardcoded in **factory contract** and **total fee** (swap_fee + sienna_fee) is lesser than 1.

Remediation plan::

RISK ACCEPTED: The Sienna.Network team accepted the risk for this finding, also stated that if for whatever reason a mistake is made, it can quickly be corrected.

3.4 (HAL-04) ADDING LIQUIDITY TO NEW POOLS DOES NOT WORK PROPERLY - MEDIUM

Description:

When users call add_liquidity function in **contracts/exchange/src/contract.rs** to add liquidity to new pools (i.e.: pools with no deposits), the assert_slippage_tolerance function is triggered and will always panic if **slippage** is specified at the beginning of the operation. This situation can produce the following consequences:

- When legitimate users try to add liquidity to new pools, operations will always panic and make users spend transactions fees needlessly.
- To force a new pool to work as expected, a user should transfer tokens directly to the pool without receiving LP tokens in return, and with the risk that another users benefit from his deposit.
- The issues explained above will arise **every time** a new pool is created (or when its deposits become 0) and legitimate users try to add liquidity.

A proof of concept video showing how to exploit this security issue is included in the report.

Code Location:

Listing 9: contracts/exchange/src/contract.rs (Lines 723,727)

```
720 if decimal_math::decimal_multiplication(
721     Decimal::from_ratio(deposits[0], deposits[1]),
722     one_minus_slippage_tolerance,
723 ) > Decimal::from_ratio(pools[0], pools[1])
724       || decimal_math::decimal_multiplication(
725           Decimal::from_ratio(deposits[1], deposits[0]),
726           one_minus_slippage_tolerance,
```

```
    ) > Decimal::from_ratio(pools[1], pools[0])
{
    return Err(StdError::generic_err(
        "Operation exceeds max slippage tolerance",
    ));
}
```

Risk Level:

Likelihood - 5
Impact - 2

Recommendations:

Update the logic of assert_slippage_tolerance function to handle correctly the case where a pool has no deposits and **slippage** is specified as an argument of the function.

Remediation plan:

SOLVED: The issue was fixed in commit 55e2f9770584cecf06ee37d15c253900de1a1d48.

3.5 (HAL-05) MAXIMUM THRESHOLD FOR SLIPPAGE IS NOT ENFORCED WHEN ADDING LIQUIDITY OR SWAPPING - MEDIUM

Description:

When users add liquidity / swap and do not specify slippage tolerance (or its equivalent) in the operation, Sienna.Network AMM protocol does not enforce a default maximum threshold, which could severely affect users' amount of tokens received in return. This issue can produce the following scenarios:

Scenario #1: Adding liquidity

- Someone creates a pool with **8000 token X** and **2000 token Y**, as a consequence, creator receives 4000 LP in return.
- User A sends a transaction to provide liquidity of **80 token X** and **20 token Y** to the pool, so he expects to receive 40 LP in return.
- However, some seconds before transaction of user A is processed, user B swaps **12000 token X** to **1200 token Y**. The final balance in the pool is: **20000 token X** and **800 token Y**.
- When transaction of user A is processed, he receives 16 LP in return, instead of 40 LP he was expecting, i.e.: **less than 50%**.

Scenario #2: Adding liquidity (imbalanced token pair)

If a user mistakenly (or fooled by an attacker) provides liquidity with an imbalanced token pair, he could lose all his surplus of tokens. See the following example:

- Someone creates a pool with **8000 token X** and **2000 token Y**, as a consequence, creator receives 4000 LP in return.

- User A provides liquidity of **80 token X** and **20 token Y** to the pool, so he receives 40 LP in return.
- User B provides liquidity of **80 token X** and **2000 token Y**, he also receives 40 LP in return, the same amount of LP tokens than previous transaction, but spending **100 times more** token B.

Scenario #3: Swapping

- Someone creates a pool with **8000 token X** and **2000 token Y**.
- User A sends a transaction to swap **100 token X** and expects to receive ~25 token Y in return.
- However, some seconds before transaction of user A is processed, user B swaps **12000 token X** to **1200 token Y**. The final balance in the pool is: **20000 token X** and **800 token Y**.
- When transaction of user A is processed, he receives ~4 token Y in return, instead of ~25 token Y he was expecting, i.e.: **less than 20%** of expected value.

Some recent DeFi attacks as occurred to BT.Finance or Saddle Finance show the importance to have a maximum predefined slippage to reduce the impact of tokens loss if unexpected situations appear or attackers compromise smart contracts in a platform.

Code Location:

When users add liquidity to a pool, assert_slippage_tolerance function will always return **Ok(())** if slippage is not specified:

Listing 10: contracts/exchange/src/contract.rs (Lines 712,713)

```
707 fn assert_slippage_tolerance(
708     slippage: Option<Decimal>,
709     deposits: &[Uint128; 2],
710     pools: &[Uint128; 2],
711 ) -> StdResult<()> {
```

```
    if slippage.is_none() {
        return Ok(());
    }
```

When users try to swap, swap function does not verify if difference between expected value and return value is within a default maximum threshold when expected_return is not specified in the operation:

Listing 11: contracts/exchange/src/contract.rs (Lines 461)

```
458 let settings = query_exchange_settings(querier, config.
        factory_info.clone())?;
459 let swap = do_swap(querier, &config, &settings, &offer, false)?;
460
461 if let Some(expected_return) = expected_return {
462     if swap.result.return_amount.lt(&expected_return) {
463         return Err(StdError::generic_err(
464             "Operation fell short of expected_return",
465         ));
466     }
467 }
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendations:

Enforce the use of a default maximum threshold when users add liquidity or swap, but do not specify slippage tolerance or slippage value is greater than the threshold. As a reference, **max slippage** for Uniswap Pool and Uniswap Swap is 50%.

Remediation plan::

RISK ACCEPTED: The Sienna.Network team accepted the risk for this finding, also stated that the front-end already pre-calculates the expected_return parameter for swaps and they will also make changes for it to set the slippage tolerance for providing liquidity.

3.6 (HAL-06) PRIVILEGED ADDRESS CAN BE TRANSFERRED WITHOUT CONFIRMATION - MEDIUM

Description:

An incorrect use of HandleMsg::Admin message in **contracts/factory/src/-contract.rs** can set admin of **factory** contract to an invalid address and inadvertently lose total control of this contract, which cannot be undone in any way.

Currently, the admin of **factory** contract can change the **admin address** using the aforementioned message in a single transaction and without confirmation from the new address.

Code Location:

HandleMsg::Admin message in **factory** is routed to admin_handle function:

Listing 12: contracts/factory/src/contract.rs (Lines 78)

```
75      HandleMsg::AddExchanges { exchanges } => add_exchanges(deps,
            env, exchanges),
76      HandleMsg::AddIdos { idos } => add_idos(deps, env, idos),
77      HandleMsg::AddLaunchpad { launchpad } => add_launchpad(deps,
            env, launchpad),
78      HandleMsg::Admin(msg) => admin_handle(deps, env, msg,
            AdminHandle),
```

admin_handle function calls change_admin function:

Listing 13: libraries/fadroma-21.07/scrt-admin/composable-admin/src/admin.rs (Lines 18)

```
11  pub fn admin_handle<S: Storage, A: Api, Q: Querier>(
12      deps: &mut Extern<S, A, Q>,
13      env: Env,
```

```
    msg: AdminHandleMsg,
    handle: impl AdminHandle,
) -> StdResult<HandleResponse> {
    match msg {
        AdminHandleMsg::ChangeAdmin { address } => handle.
            change_admin(deps, env, address)
    }
}
```

change_admin function saves the new admin address in a single transaction:

Listing 14: libraries/fadroma-21.07/scrt-admin/composable-admin/src/admin.rs (Lines 40)

```
32 pub trait AdminHandle {
33     fn change_admin<S: Storage, A: Api, Q: Querier>(
34         &self,
35         deps: &mut Extern<S, A, Q>,
36         env: Env,
37         address: HumanAddr,
38     ) -> StdResult<HandleResponse> {
39         assert_admin(deps, &env)?;
40         save_admin(deps, &address)?;
41 
42         Ok(HandleResponse::default())
43     }
44 }
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendations:

It is recommended to split **admin transfer** functionality into set_admin and accept_admin functions. The latter function allows the transfer to be completed by the recipient.

Remediation plan:

SOLVED: The issue was fixed in commit d92bf78c98f29c9eab73cf32a135ebb0600ffec4.

3.7 (HAL-07) EXCHANGES MIGRATION MECHANISM IS NOT COMPLETE - LOW

Description:

When a new factory is deployed to replace an old one, it is mandatory to consider the following steps:

1. Exchanges must be registered in the new **factory** contract. add_exchanges function in **contracts/factory/src/contract.rs** can be used by admin to complete this step.
2. Factory address in all exchanges deployed must be updated; otherwise, changes in new factory (e.g.: fee rates) won't affect any exchange. Currently, there are no mechanisms in **factory** or **exchange** contracts to complete this step.

Code Location:

Listing 15: contracts/factory/src/contract.rs

```
#[require_admin]
fn add_exchanges<S: Storage, A: Api, Q: Querier>(
    deps: &mut Extern<S, A, Q>,
    env: Env,
    exchanges: Vec<Exchange<HumanAddr>>,
) -> StdResult<HandleResponse> {
    store_exchanges(deps, exchanges)?;

    Ok(HandleResponse {
        messages: vec![],
        log: vec![log("action", "add_exchanges")],
        data: None,
    })
}
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendations:

Update the exchanges' migration mechanism to allow a mass migration process with security considerations, e.g.: restrict address that participate in the migration, use temporary password, etc. It is also important that this mechanism updates factory address in all exchanges deployed.

Remediation plan:

SOLVED: The following commits fixed the security issue:

- 1eda85cae42125cc327691cc31e59728ecb4cfe0
- 43ead12b75aa74bf6cecd342d390206f63eea86f
- 6d08465d71364877e892a383a7d5f9bb51c2c272
- 835aed6b9c2ef76dc4abd90c2c7a1cbbe96fc5a8

3.8 (HAL-08) POSSIBILITY TO CREATE FAKE PAIRS WITH NATIVE COINS - LOW

Description:

An attacker can create a pool with a pair that contains a fake native token trying to imitate a real one, e.g.: denom in native token uses 'USCRT' / 'Uscrt' / 'uSCRT' / . . . as value instead of 'uscrt', as shown in the following example:

- **Token 1:** `{ custom_token: {...} }`
- **Token 2:** `{ native_token: { denom: 'USCRT'} }`

This pool is created and will appear as a legitimate one in **factory** contract, and when users try to add liquidity to the pool, operations will always fail and make users spend transactions fees needlessly.

This issue happens because `generate_pair_key` function in **contracts/factory/src/state.rs** does not restrict that denom in native tokens use upper case letters.

Code Location:

Listing 16: contracts/factory/src/state.rs (Lines 290,295)

```
286 pub(crate) fn generate_pair_key(pair: &TokenPair<CanonicalAddr>)
        -> Vec<u8> {
287     let mut bytes: Vec<&[u8]> = Vec::new();
288
289     match &pair.0 {
290         TokenType::NativeToken { denom } => bytes.push(denom.
                as_bytes()),
291         TokenType::CustomToken { contract_addr, .. } => bytes.push
                (contract_addr.as_slice()),
292     }
293
294     match &pair.1 {
295         TokenType::NativeToken { denom } => bytes.push(denom.
```

```
                as_bytes()),
            TokenType::CustomToken { contract_addr, .. } => bytes.push
                (contract_addr.as_slice()),
    }
```

Risk Level:

Likelihood - 3
Impact - 1

Recommendations:

Update the logic of generate_pair_key function to throw an error message when denom in native tokens use upper case letters.

Remediation plan:

SOLVED: The issue was fixed in commit 0255154665c3b034bab328efb0abdb142adb4376.

3.9 (HAL-09) PARTIALEQ FOR TOKENPAIR IS WRONGLY IMPLEMENTED - INFORMATIONAL

Description:

The implementation of PartialEq for TokenPair in **libraries/amm-shared/src/token_pair.rs** makes wrong comparisons between two **TokenPair**, as shown in the following example:

Example of comparison:

1. **pair = (A, B)**

```rust
        let pair: TokenPair<HumanAddr> = TokenPair(
            TokenType::CustomToken {
                contract_addr: "address".into(),
                token_code_hash: "hash".into(),
            },
            TokenType::NativeToken {
                denom: "denom".into(),
            },
        );
```

2. **pair2 = (A, A)**

```rust
        let pair2: TokenPair<HumanAddr> = TokenPair(pair.0.clone(), pair.0.clone());
```

3. When **pair** is compared against **pair2**, the test fails because the comparison concludes that values are different:

```rust
        assert_eq!(pair, pair2);
    }
}
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL        test-mod token_pair::tests - Task

failures:
    token_pair::tests::token_pair_equality

test result: FAILED. 0 passed; 1 failed; 0 ignored; 0 measured; 1 filtered out; finished in 0

error: test failed, to rerun pass '-p amm-shared --lib'
The terminal process "C:\Users\OwlAtNite\.cargo\bin\cargo.exe 'test', '--package', 'amm-share
 exit code: 101.
```

4. However, when **pair2** is compared against **pair**, the test succeeds because the comparison wrongly concludes that values are equal:

```
159             assert_eq!(pair2, pair);
160         }
161     }
162
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

    Finished test [unoptimized + debuginfo] target(s) in 0.63s
     Running unittests (target\debug\deps\amm_shared-645fed86e3c876d5.exe)

running 1 test
test token_pair::tests::token_pair_equality ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 1 filtered out; finished in
0.00s
```

Although the vulnerability is not exploitable in current codebase, it is important to solve it because it is inside a library, and as such, it could be wrongly used in later iterations of the codebase and generates unexpected situations.

Code Location:

Listing 17: libraries/amm-shared/src/token_pair.rs (Lines 80)

```
78 impl<A: PartialEq> PartialEq for TokenPair<A> {
79     fn eq(&self, other: &TokenPair<A>) -> bool {
80         (self.0 == other.0 || self.0 == other.1) && (self.1 ==
               other.0 || self.1 == other.1)
81     }
82 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendations:

Update the implementation of PartialEq to make adequate comparisons between TokenPair values. Below is a proposed sample code:

Listing 18: Sample code for TokenPair comparison

```
impl<A: PartialEq> PartialEq for TokenPair<A> {
    fn eq(&self, other: &TokenPair<A>) -> bool {
        (self.0 == other.0 && self.1 == other.1) || (self.0 ==
            other.1 && self.1 == other.0)
    }
}
```

Remediation plan:

SOLVED: The issue was fixed in commit 9826e90b55961f18c47ed355d7b2fe9c07190739.

3.10 (HAL-10) SPREAD AMOUNT IS CALCULATED BUT NOT USED - INFORMATIONAL

Description:

The spread_amount value is calculated in compute_swap function from **contracts/exchange/src/contract.rs**, but is not used as part of the internal logic of this function, nor in do_swap (precedent function). So, its presence is not necessary for the correct working of swapping operations and only incurs in additional gas fees consumption.

Code Location:

Listing 19: contracts/exchange/src/contract.rs (Lines 687,688,692)

```
686 // spread = offer_amount * ask_pool / offer_pool - return_amount
687 let spread_amount = ((offer_amount * ask_pool)? / offer_pool)?;
688 let spread_amount = (spread_amount - return_amount).unwrap_or(
        Uint256::zero());
689
690 Ok(SwapResult {
691     return_amount: return_amount.clamp_u128()?.into(),
692     spread_amount: spread_amount.clamp_u128()?.into(),
693 })
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

If not used, it is recommended to remove spread_amount calculus in compute_swap function to reduce gas fees consumption.

Remediation plan::

ACKNOWLEDGED: The Sienna.Network team acknowledged this finding.

THANK YOU FOR CHOOSING

HALBORN