Dexter - Dexter Core

CosmWasm Financial Security Audit

Prepared by: **Halborn**

Date of Engagement: **December 5th, 2022 - February 10th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY 4
CONTACTS 5
1 EXECUTIVE OVERVIEW 6
1.1 INTRODUCTION 7
1.2 AUDIT SUMMARY 7
1.3 TEST APPROACH & METHODOLOGY 9
RISK METHODOLOGY 9
1.4 SCOPE 11
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 13
3 FINDINGS & TECH DETAILS 14
3.1 (HAL-01) SWAP FEE NOT VALIDATED FROM INPUT - HIGH 16
Description 16
Code Location 16
Risk Level 17
Recommendation 17
Remediation Plan 17
3.2 (HAL-02) THE MAXIMUM ALLOWED FEE COULD BE EQUAL THAN THE TOTAL SWAPPED AMOUNT - MEDIUM 18
Description 18
Code Location 18
Risk Level 19
Recommendation 19
Remediation Plan 19
3.3 (HAL-03) EMERGENCY UNSTAKE DO NOT ALLOW WITHDRAWAL OF FUNDS IN THE SHORT TERM - MEDIUM 20
Description 20

Code Location 20
Risk Level 21
Recommendation 21
Remediation Plan 21
3.4 (HAL-04) INITIAL LP TOKENS AMOUNT MAY VARY - LOW 22
Description 22
Code Location 22
Risk Level 23
Recommendation 23
Remediation Plan 23
3.5 (HAL-05) STAKE COULD BE STUCK IN THE CONTRACT - LOW 24
Description 24
Code Location 24
Risk Level 25
Recommendation 25
Remediation Plan 25
3.6 (HAL-06) REWARDS COULD BE STUCK IN THE CONTRACT - LOW 26
Description 26
Code Location 26
Risk Level 27
Recommendation 27
Remediation Plan 27
3.7 (HAL-07) ROUNDING ISSUES - LOW 28
Description 28
Code Location 28
Risk Level 29

Recommendation 29

Remediation Plan 29

3.8 (HAL-08) MATH MISCALCULATION - LOW 30

Description 30

Code Location 30

Risk Level 31

Recommendation 31

Remediation Plan 31

3.9 (HAL-09) NON-PROXY REWARDS ARE LOST DURING EMERGENCY UNSTAKE - LOW 32

Description 32

Code Location 32

Risk Level 33

Recommendation 33

Remediation Plan 33

3.10 (HAL-10) STABLE SWAP FORMULA DIFFERS FROM WHITEPAPER - INFORMATIONAL 34

Description 34

Code Location 34

Risk Level 35

Recommendation 35

Remediation Plan 35

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	12/05/2022	Elena Maranon
0.2	Document Update	02/06/2023	Elena Maranon
0.3	Document Update	02/08/2023	Elena Maranon
0.4	Draft Version	02/10/2023	Elena Maranon
0.5	Draft Review	02/15/2023	Luis Quispe Gonzales
0.6	Draft Review	02/15/2023	Gabi Urrutia
1.0	Remediation Plan	02/15/2023	Luis Quispe
1.1	Remediation Plan Review	02/16/2023	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
Elena Maranon	Halborn	Elena.Maranon@halborn.com
Gonzalo Junquera	Halborn	Gonzalo.Junquera@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Dexter engaged Halborn to conduct a financial audit of its smart contracts from December 5th, 2022 to February 10th, 2023. The security assessment was limited to the smart contracts provided in the GitHub repository Dexter-core, commit hashes and more details can be found in the Scope section of this report.

The audit focused on the financial part of the project, analyzing in depth all aspects related to the movement of funds, such as pools or the rewards' system.

1.2 AUDIT SUMMARY

The team at Halborn assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Thoroughly review the financial aspects to avoid any anomalous performance.
- Ensure that the mathematical mechanisms work as explained in the documentation.
- Ensure that smart contract functions operate 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 Dexter team. The main ones are the following:

- Validate the swap fee value provided by the user before saving it in the pool configuration.

- Verify that the maximum fee allowed for swapping is less than a predefined threshold, e.g.: 25%.
- In case of emergency, allow users to withdraw their funds immediately or within a short period of time.
- Separate the precision from the amount of tokens for the variable containing the initial amount of LP tokens to be minted in the weighted-pool contract.
- Add a function in the multi-staking contract to update the 'unlo ck_period' parameter in case of error.
- Validate the 'distribution_schedule' parameter in the ref_staking contract and add a function to update its value in case of error.
- Keep the 'distributed_amount' parameter as Decimal through all calculations.
- Handle appropriately the edge cases of 'calculate_pow' function in the weighted-pool contract.
- Save the rewards accumulated by the generator contract also as orphans, instead of reducing them to zero.

1.3 TEST APPROACH & METHODOLOGY

This framework provides a risk-based approach to assess the likelihood of a financial security event based on auditing the interactions and inputs around environmental factors of a smart contract or DeFi protocol.

Given the dynamic nature of such an audit, several approaches are combined to perform a holistic assessment of which developers can make the best effort to protect themselves from a revenue impacting event through risk awareness and mitigating factors.

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the Rust code 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 and purpose.
- Smart contract manual code review and walkthrough.
- Manual testing by custom scripts and fuzzers.
- Scanning of Rust files for vulnerabilities, security hotspots or bugs.
- Static Analysis of security for scoped contract, and imported functions.
- Testnet deployment.

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.

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

1.4 SCOPE

First round of testing (Dec 5th - Jan 20th):

1. CosmWasm Smart Contracts
    (a) Repository: dexter-core
    (b) Commit ID: 6b49e0f
    (c) Contracts in scope:
        i. xyk_pool
        ii. stable_pool
        iii. stable_5pool
        iv. weighted_pool
        v. ref_staking
        vi. multi_staking
        vii. lp_token
        viii. generator
        ix. generator_proxy
        x. keeper
        xi. router
        xii. vesting
        xiii. vault

Second round of testing (Jan 20th - Feb 2nd): New commit with fixes and changes

1. CosmWasm Smart Contracts
    (a) Repository: dexter-core
    (b) Commit ID: c45b9dc
    (c) Contracts in scope:
        i. xyk_pool
        ii. stable_pool

iii. stable_5pool
iv. weighted_pool
v. ref_staking
vi. multi_staking
vii. lp_token
viii. generator
ix. generator_proxy
x. keeper
xi. router
xii. vesting
xiii. vault

Third round of testing (Feb 2nd - Feb 10th): New commit with fixes

1. CosmWasm Smart Contracts
   (a) Repository: dexter-core
   (b) Commit ID: 06dfa23
   (c) Contracts in scope:
      i. stable_5pool
      ii. weighted_pool
      iii. multi_staking
      iv. lp_Token
      v. keeper
      vi. router
      vii. vault

Out-of-scope: Not-specified contracts and external libraries.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	2	6	1

LIKELIHOOD

IMPACT

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

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) SWAP FEE NOT VALIDATED FROM INPUT	High	SOLVED - 01/19/2023
(HAL-02) THE MAXIMUM ALLOWED FEE COULD BE EQUAL THAN THE TOTAL SWAPPED AMOUNT	Medium	SOLVED - 01/23/2023
(HAL-03) EMERGENCY UNSTAKE DO NOT ALLOW WITHDRAWAL OF FUNDS IN THE SHORT TERM	Medium	SOLVED - 02/02/2023
(HAL-04) INITIAL LP TOKENS AMOUNT MAY VARY	Low	SOLVED - 01/31/2023
(HAL-05) STAKE COULD BE STUCK IN THE CONTRACT	Low	SOLVED - 01/24/2023
(HAL-06) REWARDS COULD BE STUCK IN THE CONTRACT	Low	SOLVED - 02/02/2023
(HAL-07) ROUNDING ISSUES	Low	RISK ACCEPTED
(HAL-08) MATH MISCALCULATION	Low	SOLVED - 12/14/2023
(HAL-09) NON-PROXY REWARDS ARE LOST DURING EMERGENCY UNSTAKE	Low	SOLVED - 02/02/2023
(HAL-10) STABLE SWAP FORMULA DIFFERS FROM WHITEPAPER	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) SWAP FEE NOT VALIDATED FROM INPUT - HIGH

Description:

The execute_create_pool_instance function from **vault** contract allows creating a new instantiation of those pools that have been previously registered.

One of the input parameters of this function is fee_info, which indicates the percentage of the swapped amount that will be collected as fee. This is an optional parameter and, in case of existence, it replaces the default_fee_info, being directly copied from the input of the ExecuteMsg ::CreatePoolInstance to the tmp_pool_info variable without any kind of validation of maximum or minimum and, consequently, used in the creation of the pool.

None of the instantiate functions of the pools performs additional checks on the fee_info value, so it could be any value from 0% to 100%. It could even be higher than 100%, but the swap execution would fail.

Since some pools can be instantiated by everyone, the risk of this issue becomes higher.

Code Location:

Fragment of execute_create_pool_instance function from **vault** contract:

Listing 1: contracts/vault/src/contract.rs (Lines 789,794)

```
786 // Pool Id for the new pool instance
787 let pool_id = config.next_pool_id;
788
789 let fee_info = fee_info.unwrap_or(pool_type_config.
 ↳ default_fee_info);
790 let tmp_pool_info = TmpPoolInfo {
791     code_id: pool_type_config.code_id,
792     pool_id,
```

```
793     lp_token_addr: None,
794     fee_info: fee_info.clone(),
795     assets,
796     pool_type: pool_type_config.pool_type.clone(),
797     init_params
798 };
799
800 // Store the temporary Pool Info
801 TMP_POOL_INFO.save(deps.storage, &tmp_pool_info)?;
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendation:

The `fee_info` input value must be validated before being assigned to the pool instantiation info. It is also good practice to include this validation in the instantiation function of each pool before saving the value in the configuration.

As a general recommendation, all percentage rates should be checked to ensure that they are within certain limits.

Remediation Plan:

SOLVED: The Dexter team has solved this issue in commit `214e122` by adding a call to `valid_fee_info` inside the `execute_create_pool_instance` function.

3.2 (HAL-02) THE MAXIMUM ALLOWED FEE COULD BE EQUAL THAN THE TOTAL SWAPPED AMOUNT - MEDIUM

Description:

The valid_fee_info function from **vault** package performs some checks on the fee_info parameter. This function is used during the initialization and update of the fee info in the Pool Type configuration, which is overridden if the optional parameter of **CreatePoolInstance** is set.

The maximum allowed value for fee.info.total_fee_bps is 10_000 with the same precision of 10_000. In the calculate_underlying_fees function it can be checked that, if fee_info.total_fee_bps reaches the maximum allowed, the amount will be multiplied by 1, which means that the fee percentage is the 100% of the amount. Even if fee_info is validated, a maximum fee value of 100% of the swapped amount does not make sense for the purpose of the project.

All functions that make use of valid_fee_info have access control for the owner, which reduces the likelihood of this issue.

Code Location:

The valid_fee_info function from **vault** package:

Listing 2: packages/dexter/src/vault.rs (Line 92)

```
91 pub fn valid_fee_info(&self) -> bool {
92     self.total_fee_bps <= MAX_TOTAL_FEE_BPS
93         && self.protocol_fee_percent <= MAX_PROTOCOL_FEE_PERCENT
94         && self.dev_fee_percent <= MAX_DEV_FEE_PERCENT
95 }
```

Constant values from **vault** package:

Listing 3: packages/dexter/src/vault.rs (Line 73)

```
69 // FEE PRECISION is 4 decimal places
70 pub const FEE_PRECISION: u16 = 10_000u16;
71 // Maximum total commission in bps that can be charged on any
 ↳ supported pool by Dexter
72 // If MAX_TOTAL_FEE_BPS / FEE_PRECISION is 1, then the maximum
 ↳ total commission that can be charged on any supported pool by
 ↳ Dexter is 1%
73 const MAX_TOTAL_FEE_BPS: u16 = 10_000u16;
```

The calculate_underlying_fees function from **helpers** package:

Listing 4: packages/dexter/src/helper.rs (Line 383)

```
381 // Returns the number of tokens charged as total fee
382 pub fn calculate_underlying_fees(amount: Uint128, total_fee_bps:
 ↳ u16) -> Uint128 {
383     amount * Decimal::from_ratio(total_fee_bps, FEE_PRECISION)
384 }
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

It is recommended to have a maximum swap fee allowed below 25%.

Remediation Plan:

SOLVED: The Dexter team has solved this issue in commit beafd41 by changing the MAX_TOTAL_FEE_BPS value from 10_000 to 1_000, which is equivalent to a fee of 10%.

3.3 (HAL-03) EMERGENCY UNSTAKE DO NOT ALLOW WITHDRAWAL OF FUNDS IN THE SHORT TERM - MEDIUM

Description:

The emergency_unstake function of the **generator** contract allows withdrawing the stake deposited in the contract if some kind of emergency occurs. However, the withdrawal cannot be done immediately, it is necessary to wait until the whole unlock_period has ended, like in normal unstake situations.

This behavior turns the emergency_unstake into a normal unstake but losing all the accumulated rewards.

Code Location:

Fragment of emergency_unstake function from **generator** contract:

Listing 5: contracts/dexter_generator/generator/src/contract.rs (Lines 1078-1082,1085)

```
1069 // Update user's balance. All LP tokens are to be unbonded and the
 ↳  user's bonded amount is set to 0.
1070 let unbonded_amount = user.amount;
1071 let mut user = update_user_balance(user, &pool, Uint128::zero())?;
1072
1073 // Check that amount is non-zero
1074 if unbonded_amount == Uint128::zero() {
1075     return Err(ContractError::ZeroUnbondAmount {});
1076 }
1077
1078 // Create unbonding period
1079 let unbonding_period = UnbondingInfo {
1080     amount: unbonded_amount,
1081     unlock_timestamp: env.block.time.seconds() + cfg.
 ↳ unbonding_period,
1082 };
```

```
083
084 // Save the unbonding period
085 user.unbonding_periods.push(unbonding_period);
086
087 // Change the user's balance
088 USER_INFO.save(deps.storage, (&lp_token, &info.sender.clone()), &
  ↳ user)?;
089 POOL_INFO.save(deps.storage, &lp_token, &pool)?;
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

It is recommended to allow any user to withdraw their funds **immediately** in case of emergency or, at least, using a short unlocking period.

Remediation Plan:

SOLVED: The Dexter team has solved this issue since the **generator** contract has been removed from the final version of the code.

3.4 (HAL-04) INITIAL LP TOKENS AMOUNT MAY VARY - LOW

Description:

The query_on_join_pool function of the **weighted_pool** contract uses the constant variable INIT_LP_TOKENS to mint the initial LP tokens when an empty pool receives the first deposit. This variable has a value of 100_000000, which means 100 tokens with a precision of 6 decimal places.

Since the precision of the LP tokens is configured during the instantiation of the LP contract, it is not known beforehand how many decimal places it will have, so it can be different from 6, reducing or increasing this initial amount of tokens.

Code Location:

Initialization of the constant INIT_LP_TOKENS in the **weighted_pool** contract:

Listing 6: contracts/pools/weighted_pool/src/contract.rs (Line 39)

```
38 // Number of LP tokens to mint when liquidiity is provided for the
 ↳  first time to the pool
39 const INIT_LP_TOKENS: u128 = 100_000000;
```

Fragment of query_on_join_pool function from **weighted_pool** contract:

Listing 7: contracts/pools/weighted_pool/src/contract.rs (Line 547)

```
545 let (mut num_shares, remaining_tokens_in, err): (Uint128, Vec<
 ↳ Asset>, ResponseType) =
546     if total_share.is_zero() {
547         let num_shares = Uint128::from(INIT_LP_TOKENS);
548         (num_shares, vec![], ResponseType::Success {})
549     } else {
550         maximal_exact_ratio_join(act_assets_in.clone(), &
 ↳ pool_assets_weighted, total_share)?
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

It is recommended to separate the precision from the number of tokens, since the decimal precision is not known until the moment of the LP contract instantiation.

Remediation Plan:

SOLVED: The Dexter team has solved this issue in commit 830cbce.

3.5 (HAL-05) STAKE COULD BE STUCK IN THE CONTRACT - LOW

Description:

The instantiate function of the **multi_staking** contract accepts as input the parameters owner and unlock_period. Both are saved directly from the **Instantiate** message without any validation.

In addition, **this contract has no function to update any of the configuration values**, which means that an error in the value of unlock_period would leave the stake locked in the contract for longer than desired.

Code Location:

The instantiate function form **multi_staking** contract:

Listing 8: contracts/multi_staking/src/contract.rs (Lines 42,43)

```
33 pub fn instantiate(
34     deps: DepsMut,
35     _env: Env,
36     _info: MessageInfo,
37     msg: InstantiateMsg,
38 ) -> StdResult<Response> {
39     CONFIG.save(
40         deps.storage,
41         &Config {
42             unlock_period: msg.unlock_period,
43             owner: msg.owner,
44             allowed_lp_tokens: vec![],
45         },
46     )?;
47
48     Ok(Response::default())
49 }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to add a function to update the unlock_period in case of error.

Remediation Plan:

SOLVED: The Dexter team has solved this issue in commit d017094 by adding the update_config function.

3.6 (HAL-06) REWARDS COULD BE STUCK IN THE CONTRACT - LOW

Description:

The instantiate function of the **ref_staking** contract does not validate the distribution_schedule input data before saving it in the configuration. There is not a function to update its value. This also applies to the token addresses in case they need to be updated in the future.

The variable distribution_schedule is a tuple of three values representing the starting and ending point in time of the reward distribution and the amount of rewards to be shared. There is no validation in case the ending point is lesser than starting point, **which would cause an underflow error preventing any kind of reward distribution**, or if the amount to be distributed is zero (useless operations).

Code Location:

Fragment of instantiate function from **ref_staking** contract:

Listing 9: contracts/dexter_generator/ref_staking/src/contract.rs (Line 30)

```
25     CONFIG.save(
26         deps.storage,
27         &Config {
28             anchor_token: deps.api.addr_canonicalize(&msg.
↳ anchor_token)?,
29             staking_token: deps.api.addr_canonicalize(&msg.
↳ staking_token)?,
30             distribution_schedule: msg.distribution_schedule,
31         },
32     )?;
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to validate the distribution_schedule input from InstantiateMsg before any assignment, or to include an update configuration function in the code to update its value in case of error.

Remediation Plan:

SOLVED: The Dexter team has solved this issue since the **ref_staking** contract has been removed from the final version of the code.

3.7 (HAL-07) ROUNDING ISSUES - LOW

Description:

The compute_reward function of the **multi_staking** contract calculates the reward to be distributed over a period of time.

The variable distributed_amount increases its value over a few iterations, and it starts as **Decimal**, but at the end of each iteration it is converted from **Decimal** to **Uint128**, losing some precision.

Since this value is subsequently divided by a **Decimal** number and then added to another **Decimal**, it should be kept as **Decimal** since the beginning without losing precision.

Code Location:

Fragment of compute_reward function from **multi_staking** contract:

Listing 10: contracts/multi_staking/src/contract.rs (Lines 306,310,311)

```
292 for s in reward_schedules.iter() {
293     let start_time = s.start_block_time;
294     let end_time = s.end_block_time;
295 
296     if start_time > current_block_time || end_time < state.
 ↳ last_distributed {
297         continue;
298     }
299 
300     // min(s.1, block_time) - max(s.0, last_distributed)
301     let passed_time = std::cmp::min(end_time, current_block_time)
302         - std::cmp::max(start_time, state.last_distributed);
303 
304     let time = end_time - start_time;
305     let distribution_amount_per_second: Decimal = Decimal::
 ↳ from_ratio(s.amount, time);
306     distributed_amount += distribution_amount_per_second * Uint128
 ↳ ::from(passed_time as u128);
```

```
307 }
308
309 state.last_distributed = current_block_time;
310 state.reward_index =
311     state.reward_index + Decimal::from_ratio(distributed_amount,
 ↳ total_bond_amount);
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to keep the distributed_amount parameter as **Decimal** through all calculations.

Remediation Plan:

RISK ACCEPTED: The Dexter team has accepted the risk of this finding.

3.8 (HAL-08) MATH MISCALCULATION - LOW

Description:

There is a calculation error in the calculate_pow function of the **weighted_pool** contract for edge cases exponent = 0 or base = 0. The correct approach for those cases should be:

- If base = 0 -> return 0.
- If exponent = 0 -> return 1.

Code Location:

Fragment of calculate_pow function from **weighted_pool** contract:

Listing 11: contracts/pools/weighted_pool/src/approx_pow.rs (Lines 25,26)

```
19 pub fn calculate_pow(
20     base: Decimal,
21     exp: Decimal,
22     precision: Option<Decimal>,
23 ) -> StdResult<Decimal> {
24     let precision = precision.unwrap_or(Decimal::from_str("
↳ 0.00000001").unwrap());
25     if exp.is_zero() || base.is_zero() {
26         return Ok(base);
27     };
28
29     // we can adjust the algorithm in this setting.
30     if base > Decimal::from_str("2").unwrap() {
31         return Err(StdError::generic_err(
32             "calculate_pow : base must be less than 2",
33         ));
34     }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to apply the correct results for the edge cases.

Remediation Plan:

SOLVED: The Dexter team solved this issue in commit d2db2ab.

3.9 (HAL-09) NON-PROXY REWARDS ARE LOST DURING EMERGENCY UNSTAKE - LOW

Description:

The emergency_unstake function of the **generator** contract allows withdrawing the stake if some kind of emergency occurs and saving the accumulated proxy rewards in the pool.orphan_proxy_rewards parameter. However, rewards accumulated by the generator contract using the pool.accumulated_rewards_per_share parameter are lost.

Code Location:

Fragment of emergency_unstake function from **generator** contract:

Listing 12: contracts/dexter_generator/generator/src/contract.rs (Lines 1055-1057,1071)

```
048 if let Some(proxy) = &pool.reward_proxy {
049         let accumulated_proxy_rewards = pool
050             .accumulated_proxy_rewards_per_share
051             .checked_mul_uint128(user.amount)?
052             .checked_sub(user.reward_debt_proxy)?;
053
054         // All users' proxy rewards become orphaned
055         pool.orphan_proxy_rewards = pool
056             .orphan_proxy_rewards
057             .checked_add(accumulated_proxy_rewards)?;
058
059         transfer_msgs.push(WasmMsg::Execute {
060             contract_addr: proxy.to_string(),
061             msg: to_binary(&ProxyExecuteMsg::EmergencyWithdraw {
062                 account: env.contract.address.clone(),
063                 amount: user.amount,
064             })?,
065             funds: vec![],
066         });
067     }
```

```
1068
1069     // Update user's balance. All LP tokens are to be unbonded and
 ↳  the user's bonded amount is set to 0.
1070     let unbonded_amount = user.amount;
1071     let mut user = update_user_balance(user, &pool, Uint128::zero
 ↳ ())?;
1072
1073     // Check that amount is non-zero
1074     if unbonded_amount == Uint128::zero() {
1075         return Err(ContractError::ZeroUnbondAmount {});
1076     }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to save the rewards accumulated by the generator also as orphans instead of reducing them to zero.

Remediation Plan:

SOLVED: The Dexter team has solved this issue since the **generator** contract has been removed from the final version of the code.

3.10 (HAL-10) STABLE SWAP FORMULA DIFFERS FROM WHITEPAPER - INFORMATIONAL

Description:

The formula used for **stable_pool** and **stable_5pool** contracts differs from the original formula from Curve whitepaper, which is the one referenced in the documentation.

$$An^n \sum x_i + D = DAn^n + \frac{D^{n+1}}{n^n \prod x_i}$$

Parameters in the formula for the stable swap invariant are the following:

- **n:** Number of different tokens in pool
- **A:** Amplification coefficient (defined by pool creator)
- **x, y:** Pooled tokens
- D: Stable swap invariant

In the **stable_pool** and **stable_5pool** contracts, the formula replaces the An^n by simply An. This difference is noticeable when the number of tokens in the pool is the maximum (5) and the amounts exchanged are high, resulting in fewer tokens given in exchange than the original formula indicates.

Code Location:

Fragment of calc_y function from **stable_5pool** contract:

Listing 13: contracts/pools/stable_5pool/src/math.rs (Line 112)

```
111     let n_coins = Uint64::from(pools.len() as u8);
112     let ann = Uint256::from(amp.checked_mul(n_coins)?.u64() /
 ↳ AMP_PRECISION);
113     let mut sum = Decimal256::zero();
```

```
114     let pool_values = pools.iter().map(|asset| asset.amount).
↳ collect_vec();
```

Fragment from compute_d function from **stable_5pool** contract:

Listing 14: contracts/pools/stable_5pool/src/math.rs (Line 49)

```
43     // Sum of all the pools liquidity,  Eq - xp: [1242000000,
↳ 1542000000, 1456000000] = 4240000000
44     let sum_x = pools.iter().fold(Decimal256::zero(), |acc, x| acc
↳  + (*x));
45
46     let n_coins = pools.len() as u8;
47
48     // ann = amp * n                Eq - 100 * 3 = 300
49     let ann = Decimal256::from_integer(amp.checked_mul(n_coins.
↳ into())?.u64() / AMP_PRECISION);
50     let n_coins = Uint64::from(n_coins);
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to adjust the applied formula to the whitepaper equation, or change the documentation and the comments with the modified formula.

Remediation Plan:

ACKNOWLEDGED: The Dexter team acknowledged this finding. They also stated that in the case of a 5-token pool (where there may be a maximum deviation) they do not consider these differences to have an impact.

THANK YOU FOR CHOOSING

// HALBORN