// HALBORN

PancakeSwap - Aptos DEX

Move Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **November 14th, 2022 - December 2nd, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	11/14/2022	Luis Quispe Gonzales
0.2	Document Update	12/02/2022	Luis Quispe Gonzales
0.3	Draft Version	12/07/2022	Luis Quispe Gonzales
0.4	Draft Review	12/08/2022	Gabi Urrutia
1.0	Remediation Plan	01/13/2023	Luis Quispe Gonzales
1.1	Remediation Plan Review	01/13/2023	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

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

PancakeSwap engaged Halborn to conduct a security audit on their smart contracts beginning on November 14th, 2022 and ending on December 2nd, 2022. The security assessment was scoped to the smart contracts provided in the GitHub repository pancake-contracts-move, commit hashes and further details can be found in the Scope section of this report.

1.2 AUDIT SUMMARY

The team at Halborn assigned one 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:

- 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 partially addressed by PancakeSwap team. The main ones are the following:

- Increase the value of 'owners_seq_number' by 1 every time the amount of owners or the value of the threshold are modified.
- Make use of 'capabilities' in execution functions to avoid that someone invalidates approved transactions.
- Split admin transfer functionality to allow the recipient to complete the transfer.
- Use a predefined ratio when initializing / updating multisig wallets.
- Verify that 'eta' is greater than current timestamp at initiating multisig transactions.

1.3 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 the architecture, purpose, and use of the platform.
- Smart contract manual code review and walk-through to identify any logic issue.
- Thorough assessment of safety and usage of critical Move variables and functions in scope that could lead to arithmetic related vulnerabilities.
- Test coverage review (aptos move test).
- Localnet testing of core functions(aptos-cli)

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

1.4 SCOPE

1. Move Smart Contracts

   (a) Repository: pancake-contracts-move

   (b) Commit ID: 8392b1b

   (c) Modules in scope:

   - multisig_wallet
   - admin
   - router
   - swap_utils
   - swap

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	1	0	4	3

LIKELIHOOD

IMPACT

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

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) FUNCTION TO SET THRESHOLD CAN GET THE MULTISIG WALLETS TOTALLY STUCK	Critical	SOLVED - 11/24/2022
(HAL-02) APPROVED TRANSACTIONS CAN BE INVALIDATED	High	SOLVED - 12/01/2022
(HAL-03) PRIVILEGED ADDRESS TRANSFERRED WITHOUT CONFIRMATION	Low	RISK ACCEPTED
(HAL-04) INSECURE MINIMUM THRESHOLD WHEN INITIALIZING MULTISIG WALLETS	Low	RISK ACCEPTED
(HAL-05) MINIMUM THRESHOLD IN MULTISIG WALLETS IS NOT UPDATED SECURELY	Low	RISK ACCEPTED
(HAL-06) ETA IS NOT COMPLETELY VERIFIED WHEN INITIATING MULTISIG TRANSACTIONS	Low	RISK ACCEPTED
(HAL-07) MISLEADING ERROR MESSAGES	Informational	ACKNOWLEDGED
(HAL-08) UNNECESSARY USE OF MUTABLE REFERENCES	Informational	ACKNOWLEDGED
(HAL-09) UNUSED FUNCTIONS	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) FUNCTION TO SET THRESHOLD CAN GET THE MULTISIG WALLETS TOTALLY STUCK - CRITICAL

Description:

`init_set_threshold` and `init_remove_owner` can be called in parallel during usual operations in multisig wallets. However, this behavior can produce a situation where `threshold becomes greater than number of owners`. As a consequence, multisig wallets can get totally stuck, i.e.: no more transactions or withdrawals can be done on behalf of them.

It is important to note that the likelihood of this situation to happen drastically increases because **pancake-multisig-wallet** contract is planned to be used as a library for extended implementations of multisig wallets by the community. Here is a proof of concept showing how to exploit this security issue:

Proof of Concept:

Initial situation: 3 owners and threshold is 2

1. **Owner 1** calls `init_set_threshold` function to change the threshold to 3.
2. **Owner 2** calls `approve_set_threshold` function.
3. **Owner 1** calls `init_remove_owner` function.
4. **Owner 2** calls `approve_remove_owner` function.
5. **Owner 1** calls `execute_set_threshold` function and now the threshold is 3.
6. **Owner 3** calls `approve_remove_owner` function.
7. **Owner 2** calls `execute_remove_owner` function and now there are only 2 owners.
8. Finally, the `multisig wallet gets totally stuck` because there are only 2 owners, but the threshold is 3.

Final situation: 2 owners and threshold is 3

Code Location:

Listing 1: pancake-multisig-wallet/sources/multisig_wallet.move (Lines 422-424)

```
412 let multisig_txs = borrow_global_mut<MultisigTxs<ParamsType>>(
 ↳ multisig_wallet_addr);
413 let tx = Table::borrow_mut(&mut multisig_txs.txs, seq_number);
414 assert!(Table::length(&tx.approvals) >= (multisig_wallet.threshold
 ↳  as u64), ERROR_LESS_THAN_THRESHOLD);
415 assert!(tx.owners_seq_number == multisig_wallet.owners_seq_number,
 ↳  ERROR_OWNERS_SEQ_NUMBER_NOT_MATCH);
416 tx.is_executed = true;
417 multisig_wallet.last_executed_seq_number = seq_number;
418
419 assert!(timestamp::now_seconds() >= tx.eta,
 ↳ ERROR_TIMELOCK_NOT_SURPASSED);
420 assert!(timestamp::now_seconds() < tx.expiration,
 ↳ ERROR_MULTISIG_TX_EXPIRED);
421
422 if (type_info::type_name<ParamsType>() == type_info::type_name<
 ↳ RemoveOwnerParams>()) {
423  multisig_wallet.owners_seq_number = multisig_wallet.
 ↳ owners_seq_number + 1;
424 };
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Update the logic of `execute_multisig_tx` function to increase the value of `owners_seq_number` by 1 every time the amount of owners or the value of the threshold are modified.

Remediation plan:

SOLVED: The issue was fixed in commit `0a4fde8`.

3.2 (HAL-02) APPROVED TRANSACTIONS CAN BE INVALIDATED - HIGH

Description:

`execute_multisig_tx` function allows that an owner maliciously (or mistakenly) invalidates an approved transaction. As a consequence, every transaction, like adding / removing an owner, setting a new threshold or withdrawing from the multisig wallet, can always be invalidated by just 1 malicious owner, even if the transaction goes to approval again and again.

It is important to note that the likelihood of this situation to happen drastically increases because **pancake-multisig-wallet** contract is planned to be used as a library for extended implementations of multisig wallets by the community. Here is a proof of concept showing how to exploit this security issue:

Proof of Concept:

Initial situation: 3 owners and threshold is 2

1. **Owner 1** calls `init_set_threshold` function to change the threshold to 1.

```
❯ aptos move run --profile owner1 --function-id '0x6d7b29d827a12fc676d8ecefa660163b812efaf544935582959b63d08494b191::example::init_set_threshold' --args 'u64:0' --args 'u8:1'
Do you want to submit a transaction for a range of [223300 - 334900] Octas at a gas unit price of 100 Octas? [yes/no] >
yes
{
  "Result": {
    "transaction_hash": "0xaadbaef78e393b24619412cd7d5bef5b918fe3813366e0691f73fb33a273b3d4",
    "gas_used": 2233,
    "gas_unit_price": 100,
    "sender": "4618af1588fdf1f94b39b0dbd3ceaa72aad03854e568fa357cdddc9cd32f91bc",
    "sequence_number": 8,
    "success": true,
    "timestamp_us": 1669768728502372,
    "version": 15203130,
    "vm_status": "Executed successfully"
  }
}
```

2. **Owner 2** calls approve_set_threshold function and now the transaction is ready to be executed.

```
❯ aptos move run --profile owner2 --function-id '0x6d7b29d827a12fc676d8ecefa660163b812efaf544935582959b63d08494b191::example::approve_set_threshold' --args 'u64:0'
Do you want to submit a transaction for a range of [115700 - 173500] Octas at a gas unit price of 100 Octas? [yes/no] >
yes
{
  "Result": {
    "transaction_hash": "0xa92c2000d2e16b09c015cf83642bdf68c6aceacb7435d43ccc335ed626583c9d",
    "gas_used": 1157,
    "gas_unit_price": 100,
    "sender": "196392c0333c38c9f964996075cb9039f755f118211e77d298743b83d1876832",
    "sequence_number": 5,
    "success": true,
    "timestamp_us": 1669768754733133,
    "version": 15203989,
    "vm_status": "Executed successfully"
  }
}
```

3. Any of the owners calls execute_multisig_tx function. The result is successful and an ExecuteMultisigTxEvent event is emitted. However, the threshold does not change.

```
❯ aptos move run --profile owner1 --function-id '0x6d7b29d827a12fc676d8ecefa660163b812efaf544935582959b63d08494b191::example::execute_multisig_tx' --type-args '0xff707422bd3f69c8b87368c64bba072ea8a67ac2f0d319fbbc402c658565c24d::multisig_wallet::SetThresholdParams' --args 'address:0x6d7b29d827a12fc676d8ecefa660163b812efaf544935582959b63d08494b191' --args 'u64:0'
Do you want to submit a transaction for a range of [64100 - 96100] Octas at a gas unit price of 100 Octas? [yes/no] >
yes
{
  "Result": {
    "transaction_hash": "0x6063c851af578a865d13b672eea8573ec5e23fda0aa1315c6064e53846064d21",
    "gas_used": 671,
    "gas_unit_price": 100,
    "sender": "4618af1588fdf1f94b39b0dbd3ceaa72aad03854e568fa357cdddc9cd32f91bc",
    "sequence_number": 9,
    "success": true,
    "timestamp_us": 1669773376369470,
    "version": 15275999,
    "vm_status": "Executed successfully"
  }
}
```

4. If any of the owners calls execute_set_threshold function, it will throw an error message because the approved transaction became invalidated.

```
❯ aptos move run --profile owner1 --function-id '0x6d7b29d827a12fc676d8ecefa660163b812efaf544935582959b63d08494b191::example::execute_set_threshold' --args 'u64:0'
{
  "Error": "Simulation failed with status: Move abort in 0xff707422bd3f69c8b87368c64bba072ea8a67ac2f0d319fbbc402c658565c24d::multisig_wallet: ERROR_MULTISIG_TX_INVALIDATED(0xd): "
}
```

Code Location:

Listing 2: pancake-multisig-wallet/sources/multisig_wallet.move (Lines 406,410,417)

```
406 public fun execute_multisig_tx<ParamsType: copy + store>(sender: &
 ↳ signer, multisig_wallet_addr: address, seq_number: u64) acquires
 ↳ MultisigWallet, MultisigTxs, MultisigWalletEvents {
407   let sender_addr = signer::address_of(sender);
408   let multisig_wallet = borrow_global_mut<MultisigWallet>(
 ↳ multisig_wallet_addr);
409   assert!(Table::contains(&multisig_wallet.owners, sender_addr),
 ↳ ERROR_NOT_OWNER);
410   assert!(multisig_wallet.last_executed_seq_number == MAX_U64 ||
 ↳ seq_number > multisig_wallet.last_executed_seq_number,
 ↳ ERROR_MULTISIG_TX_INVALIDATED);
411
412   let multisig_txs = borrow_global_mut<MultisigTxs<ParamsType>>(
 ↳ multisig_wallet_addr);
413   let tx = Table::borrow_mut(&mut multisig_txs.txs, seq_number);
414   assert!(Table::length(&tx.approvals) >= (multisig_wallet.
 ↳ threshold as u64), ERROR_LESS_THAN_THRESHOLD);
415   assert!(tx.owners_seq_number == multisig_wallet.
 ↳ owners_seq_number, ERROR_OWNERS_SEQ_NUMBER_NOT_MATCH);
416   tx.is_executed = true;
417   multisig_wallet.last_executed_seq_number = seq_number;
```

Risk Level:

Likelihood - 5
Impact - 3

Recommendation:

Make use of capabilities in execution functions to avoid that someone maliciously or mistakenly calls execute_multisig_tx function and invalidates approved transactions.

Remediation plan:

SOLVED: The issue was fixed in commit d09bfd6.

3.3 (HAL-03) PRIVILEGED ADDRESS TRANSFERRED WITHOUT CONFIRMATION - LOW

Description:

The set_admin function from **swap** module could set the admin to an invalid address mistakenly, unwillingly losing control of **pancake-swap** contract, which cannot be undone in any way. Currently, the admin of the contract can change its address using the aforementioned function in a single transaction and without confirmation from the new address.

Since the admin is planned to be a multisig contract, the likelihood of this scenario to happen is very low, but it is included in the report as a preventive measure, applying the security-in-depth approach.

Code Location:

Listing 3: pancake-swap/sources/swap/swap.move (Lines 701-702)

```
698 public entry fun set_admin(sender: &signer, new_admin: address)
 ↳ acquires SwapInfo {
699   let sender_addr = signer::address_of(sender);
700   let swap_info = borrow_global_mut<SwapInfo>(RESOURCE_ACCOUNT);
701   assert!(sender_addr == swap_info.admin, ERROR_NOT_ADMIN);
702   swap_info.admin = new_admin;
703 }
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

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:

RISK ACCEPTED: The PancakeSwap team accepted the risk of this finding and also stated that they will mitigate this issue from the Ops side.

3.4 (HAL-04) INSECURE MINIMUM THRESHOLD WHEN INITIALIZING MULTISIG WALLETS - LOW

Description:

Multisig wallets can be initialized as long as the threshold (i.e.: minimum amount of approvals for a transaction to be later executed) is greater or equal than 1. This default validation is not inherently secure and the good practices for handling multisig wallets recommend a reasonably secure setup for threshold (e.g.: 2 of 3, 3 of 5, etc.) in case 1 or more owners are malicious, or their keys get compromised.

Code Location:

Minimum threshold is set in 1:

Listing 4: pancake-multisig-wallet/sources/multisig_wallet.move (Line 36)

```
35 const MAX_U64: u64 = 18446744073709551615;
36 const MIN_THRESHOLD: u8 = 1;
```

Multisig wallets are initialized as long as threshold is greater or equal than minimum threshold (1), no matter how many owners are included in the wallet:

Listing 5: pancake-multisig-wallet/sources/multisig_wallet.move (Line 133)

```
131 public fun initialize(sender: &signer, owner_addresses: vector<
 ↳ address>, threshold: u8) {
132   let num_owners = vector::length(&owner_addresses);
133   assert!(threshold >= MIN_THRESHOLD,
 ↳ ERROR_LESS_THAN_MIN_THRESHOLD);
134   assert!((threshold as u64) <= num_owners,
 ↳ ERROR_MORE_THAN_NUM_OWNERS);
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended that **multisig_wallet** module allows creators to define a ratio when initializing wallets, so the threshold can be secure enough according to the amount of owners included in the wallets. For example, if the ratio is **33%** and the amount of owners is **6**, the threshold in this multisig wallet is required to be greater or equal than **2** (not just 1).

Remediation plan:

RISK ACCEPTED: The PancakeSwap team accepted the risk of this finding and also stated that they will mitigate this issue from the Ops side.

3.5 (HAL-05) MINIMUM THRESHOLD IN MULTISIG WALLETS IS NOT UPDATED SECURELY - LOW

Description:

The amount of owners or the threshold in multisig wallets can be updated as long as the threshold (i.e.: minimum amount of approvals for a transaction to be later executed) is greater or equal than 1. This verification does not guarantee that the new setup for the wallet after the update follows the good practices for handling multisig wallets, e.g.: 2 approvers out of 3 owners, 3 of 5, etc.

Code Location:

init_add_owner function only verifies that the new owner does not already exist, but does not check that the threshold remains reasonably secure after adding one more owner:

Listing 6: pancake-multisig-wallet/sources/multisig_wallet.move (Line 249)

```
247 public fun init_add_owner(sender: &signer, multisig_wallet_addr:
 ↳ address, eta: u64, expiration: u64, owner: address) acquires
 ↳ MultisigWallet, MultisigTxs, MultisigWalletEvents {
248   let multisig_wallet = borrow_global<MultisigWallet>(
 ↳ multisig_wallet_addr);
249   assert!(!Table::contains(&multisig_wallet.owners, owner),
 ↳ ERROR_OWNER_ALREADY_EXIST);
250   init_multisig_tx<AddOwnerParams>(sender, multisig_wallet_addr,
 ↳ eta, expiration, AddOwnerParams {
251     owner,
252   });
253 }
```

init_set_threshold function only verifies that the new threshold is greater or equal than minimum threshold (1), no matter how many owners are included in the wallet:

Listing 7: pancake-multisig-wallet/sources/multisig_wallet.move (Line 267)

```
264 public fun init_set_threshold(sender: &signer,
 ↳ multisig_wallet_addr: address, eta: u64, expiration: u64,
 ↳ threshold: u8) acquires MultisigWallet, MultisigTxs,
 ↳ MultisigWalletEvents {
265   let multisig_wallet = borrow_global<MultisigWallet>(
 ↳ multisig_wallet_addr);
266   assert!((threshold as u64) <= Table::length(&multisig_wallet.
 ↳ owners), ERROR_MORE_THAN_NUM_OWNERS);
267   assert!(threshold >= MIN_THRESHOLD,
 ↳ ERROR_LESS_THAN_MIN_THRESHOLD);
268   init_multisig_tx<SetThresholdParams>(sender,
 ↳ multisig_wallet_addr, eta, expiration, SetThresholdParams {
269     threshold,
270   });
271 }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to update the logic of init_add_owner and init_set_threshold functions to allow changes only if threshold value remains **reasonably secure**. The proposed ratio value can be helpful for this purpose, see the recommendation for the following issue for more details: (HAL-04) INSECURE MINIMUM THRESHOLD WHEN INITIALIZING MULTISIG WALLETS.

Remediation plan:

RISK ACCEPTED: The PancakeSwap team accepted the risk of this finding and also stated that they will mitigate this issue from the Ops side.

3.6 (HAL-06) ETA IS NOT COMPLETELY VERIFIED WHEN INITIATING MULTISIG TRANSACTIONS - LOW

Description:

The following functions in **multisig_wallet** module do not verify that eta is greater than current timestamp at initiating multisig transactions:

- init_add_owner
- init_remove_owner
- init_set_threshold
- init_withdraw
- init_multisig_tx

As a consequence, an owner can mistakenly make a multisig transaction available to be executed before it is expected.

Code Location:

Listing 8: Affected resources

```
Module:
======
multisig_wallet

Functions:
=========
init_add_owner: L#247
init_remove_owner: L#255
init_set_threshold: L#264
init_withdraw: L#283
init_multisig_tx: L#295
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to update the logic of the functions mentioned above to verify that eta is greater than current timestamp for initiating multisig transactions.

Remediation plan:

RISK ACCEPTED: The PancakeSwap team accepted the risk of this finding and also stated that they will mitigate this issue from the Ops side.

3.7 (HAL-07) MISLEADING ERROR MESSAGES - INFORMATIONAL

Description:

Error messages shown in certain sections of code have inaccurate information, which could mislead legitimate users if these messages appear during a failed operation in **pancake-swap** contract while swapping.

Code Location:

The following functions used during swapping operations will throw the ERROR_INSUFFICIENT_OUTPUT_AMOUNT error message when the value of certain output coin is different from zero. However, this error message could mislead users and make them think that the operation failed because no enough output coins were generated during the swapping, which is not true and does not explain the root cause of the error:

Listing 9: pancake-swap/sources/swap/swap.move (Line 448)

```
440 public(friend) fun swap_exact_x_to_y_direct<X, Y>(
441   coins_in: coin::Coin<X>
442 ): (coin::Coin<X>, coin::Coin<Y>) acquires TokenPairReserve,
 ↳ TokenPairMetadata {
443   let amount_in = coin::value<X>(&coins_in);
444   deposit_x<X, Y>(coins_in);
445   let (rin, rout, _) = token_reserves<X, Y>();
446   let amount_out = swap_utils::get_amount_out(amount_in, rin, rout
 ↳ );
447   let (coins_x_out, coins_y_out) = swap<X, Y>(0, amount_out);
448   assert!(coin::value<X>(&coins_x_out) == 0,
 ↳ ERROR_INSUFFICIENT_OUTPUT_AMOUNT);
449   (coins_x_out, coins_y_out)
450 }
```

Listing 10: pancake-swap/sources/swap/swap.move (Line 471)

```
466 public(friend) fun swap_x_to_exact_y_direct<X, Y>(
467   coins_in: coin::Coin<X>, amount_out: u64
468 ): (coin::Coin<X>, coin::Coin<Y>) acquires TokenPairReserve,
 ↳ TokenPairMetadata {
469   deposit_x<X, Y>(coins_in);
470   let (coins_x_out, coins_y_out) = swap<X, Y>(0, amount_out);
471   assert!(coin::value<X>(&coins_x_out) == 0,
 ↳ ERROR_INSUFFICIENT_OUTPUT_AMOUNT);
472   (coins_x_out, coins_y_out)
473 }
```

Listing 11: pancake-swap/sources/swap/swap.move (Line 509)

```
504 public(friend) fun swap_y_to_exact_x_direct<X, Y>(
505   coins_in: coin::Coin<Y>, amount_out: u64
506 ): (coin::Coin<X>, coin::Coin<Y>) acquires TokenPairReserve,
 ↳ TokenPairMetadata {
507   deposit_y<X, Y>(coins_in);
508   let (coins_x_out, coins_y_out) = swap<X, Y>(amount_out, 0);
509   assert!(coin::value<Y>(&coins_y_out) == 0,
 ↳ ERROR_INSUFFICIENT_OUTPUT_AMOUNT);
510   (coins_x_out, coins_y_out)
511 }
```

Listing 12: pancake-swap/sources/swap/swap.move (Line 522)

```
514 public(friend) fun swap_exact_y_to_x_direct<X, Y>(
515   coins_in: coin::Coin<Y>
516 ): (coin::Coin<X>, coin::Coin<Y>) acquires TokenPairReserve,
 ↳ TokenPairMetadata {
517   let amount_in = coin::value<Y>(&coins_in);
518   deposit_y<X, Y>(coins_in);
519   let (rout, rin, _) = token_reserves<X, Y>();
520   let amount_out = swap_utils::get_amount_out(amount_in, rin, rout
 ↳ );
521   let (coins_x_out, coins_y_out) = swap<X, Y>(amount_out, 0);
522   assert!(coin::value<Y>(&coins_y_out) == 0,
 ↳ ERROR_INSUFFICIENT_OUTPUT_AMOUNT);
523   (coins_x_out, coins_y_out)
524 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Correct error messages to show more accurate information and to avoid confusing users if these messages appear.

Remediation plan:

ACKNOWLEDGED: The PancakeSwap team acknowledged this finding.

3.8 (HAL-08) UNNECESSARY USE OF MUTABLE REFERENCES - INFORMATIONAL

Description:

admin and fee_to functions in **swap** module use mutable references whose values are not later changed. This issue does not trigger an exploitable scenario, but is included in the report as a good practice based on the principle of least privilege.

Code Location:

Functions that make unnecessary use of mutable references:

Listing 13: pancake-swap/sources/swap/swap.move (Line 273)

```
272 public fun admin(): address acquires SwapInfo {
273   let swap_info = borrow_global_mut<SwapInfo>(RESOURCE_ACCOUNT);
274   swap_info.admin
275 }
```

Listing 14: pancake-swap/sources/swap/swap.move (Line 278)

```
277 public fun fee_to(): address acquires SwapInfo {
278   let swap_info = borrow_global_mut<SwapInfo>(RESOURCE_ACCOUNT);
279   swap_info.fee_to
280 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to utilize immutable references by using `borrow_global` operation instead of `borrow_global_mut` in the functions mentioned above.

Remediation plan:

ACKNOWLEDGED: The PancakeSwap team acknowledged this finding.

3.9 (HAL-09) UNUSED FUNCTIONS - INFORMATIONAL

Description:

Some functions in the **multisig_wallet** module are declared in the code, but not used appropriately. This is not a security issue itself, but a good practice recommendation to improve code hygiene.

Code Location:

next_seq_number function is declared in the code, but never used by any other function, except during tests.

Listing 15: pancake-multisig-wallet/sources/multisig_wallet.move

```
201 public fun next_seq_number(multisig_wallet_addr: address): u64
 ↳ acquires MultisigWallet {
202   let multisig_wallet = borrow_global<MultisigWallet>(
 ↳ multisig_wallet_addr);
203   Table::length(&multisig_wallet.seq_number_to_params_type_name)
204 }
```

is_withdraw_multisig_txs_registered function is declared in the code, but not used in the assert of the init_withdraw function:

Listing 16: pancake-multisig-wallet/sources/multisig_wallet.move

```
273 public fun is_withdraw_multisig_txs_registered<CoinType>(addr:
 ↳ address): bool {
274   exists<MultisigTxs<WithdrawParams<CoinType>>>(addr)
275 }
```

Listing 17: pancake-multisig-wallet/sources/multisig_wallet.move (Line 284)

```
283 public fun init_withdraw<CoinType>(sender: &signer,
 ↳ multisig_wallet_addr: address, eta: u64, expiration: u64, amount:
```

```
 ↳ u64) acquires MultisigWallet, MultisigTxs, MultisigWalletEvents {
284   assert!(exists<MultisigTxs<WithdrawParams<CoinType>>>(
 ↳ multisig_wallet_addr), ERROR_MULTISIG_TXS_NOT_EXIST);
285   let sender_addr = signer::address_of(sender);
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to include the **#[test]** annotation in next_seq_number function and use is_withdraw_multisig_txs_registered in the assert of the init_withdraw function.

Remediation plan:

ACKNOWLEDGED: The PancakeSwap team acknowledged this finding.

THANK YOU FOR CHOOSING

// HALBORN