Satay Finance - Satay Aptos

Move Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **December 2nd, 2022 - December 21st, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	12/02/2022	Lukasz Mikula
0.2	Document Update	12/20/2022	Lukasz Mikula
0.3	Draft Version	12/22/2022	Lukasz Mikula
0.4	Draft Review	12/23/2022	Gabi Urrutia
1.0	Remediation Plan	01/06/2023	Jakub Heba
1.1	Remediation Plan Review	01/06/2023	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
Lukasz Mikula	Halborn	Lukasz.Mikula@halborn.com
Jakub Heba	Halborn	Jakub.Heba@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Satay Finance engaged Halborn to conduct a security audit on their smart contracts beginning on December 2nd, 2022 and ending on December 21st, 2022. The security assessment was scoped to the smart contracts provided in the GitHub repository satay-aptos. 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 has been successfully addressed by Satay Finance. The main ones are the following:

- Update the code of withdrawal process in such a way that it unstakes instead of staking the Farming Coins.
- Hardcode maximal allowed slippage when interacting with liquidity pools.
- Separate the various roles, so they are controlled by different addresses and in turn, less centralized.
- Make sure that numeric variables size matches all potential data held in them.

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: satay-aptos

   (b) Commit ID: 7eb6efe

   (c) Modules in scope:

   - base_strategy
   - dao_storage
   - global_config
   - math
   - satay
   - vault
   - ditto_farming_strategy
   - ditto_farming

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	0	1	3	6

LIKELIHOOD

IMPACT

				(HAL-01)
(HAL-03)				
(HAL-04) (HAL-05)		(HAL-02)		
(HAL-06) (HAL-07)				
(HAL-08) (HAL-09) (HAL-10) (HAL-11)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) WITHDRAWAL FROM STRATEGY FAILS	Critical	SOLVED - 12/27/2022
(HAL-02) USERS ARE NOT PROTECTED AGAINST SLIPPAGE	Medium	SOLVED - 12/31/2022
(HAL-03) INSUFFICIENT ROLE SEPARATION	Low	SOLVED - 01/02/2023
(HAL-04) POTENTIAL INSECURE CALCULATIONS	Low	SOLVED - 12/30/2022
(HAL-05) POTENTIAL OVERFLOWS IN MATH MODULE	Low	SOLVED - 12/30/2022
(HAL-06) MISSING EVENT EMISSION	Informational	SOLVED - 12/29/2022
(HAL-07) VAULTS AND STRATEGIES CANNOT BE DELISTED	Informational	SOLVED - 12/29/2022
(HAL-08) MATH MODULE IS RARELY USED	Informational	SOLVED - 12/27/2022
(HAL-09) FEES VALUE INCONSISTENCY	Informational	SOLVED - 12/27/2022
(HAL-10) UPDATING STRATEGY DEBT RATIO RETURNS OLD VALUE	Informational	SOLVED - 12/27/2022
(HAL-11) CREDIT THRESHOLD IS NEVER USED	Informational	SOLVED - 12/29/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) WITHDRAWAL FROM STRATEGY FAILS - CRITICAL

Description:

The ditto_farming module provides deposit and withdraw possibilities for interacting users. The withdrawal process, however, contains a flaw. It should unstake DittoFarmingCoins to get LP<APT,stAPT> but instead it calls liquidity_mining::stake. This way, users are unable to get their deposit back.

Code Location:

The detailed explanation of the execution flow can be seen below.

1. The withdraw function calls liquidate_position using user supplied DittoFarmingCoins:

Listing 1: products/ditto-farming/sources/ditto_farming.move (Line 112)

```
107     public entry fun withdraw(
108         user: &signer,
109         amount: u64
110     ) acquires FarmingAccountCapability, DittoFarmingCoinCaps {
111         let ditto_farming_coin = coin::withdraw<DittoFarmingCoin>(
 ↳ user, amount);
112         let aptos_coin = liquidate_position(ditto_farming_coin);
113         coin::deposit<AptosCoin>(signer::address_of(user),
 ↳ aptos_coin);
114     }
```

2. Later, these coins are used as argument to function unstake_lp_and_burn, which normally should unstake them and receive LP<APT, stAPT>:

Listing 2: products/ditto-farming/sources/ditto_farming.move (Line 204)

```
198     public fun liquidate_position(
199         ditto_farming_coins: Coin<DittoFarmingCoin>,
200     ): Coin<AptosCoin> acquires FarmingAccountCapability,
 ↳ DittoFarmingCoinCaps {
201         let ditto_farming_cap = borrow_global<
 ↳ FarmingAccountCapability>(@satay_ditto_farming);
202         let ditto_farming_signer = account::
 ↳ create_signer_with_capability(&ditto_farming_cap.signer_cap);
203 
204         let lp_coins = unstake_lp_and_burn(ditto_farming_coins, &
 ↳ ditto_farming_signer);
205 
206         liquidate_lp_coins(lp_coins)
207     }
```

3. In that function, in line 217, there is a call to stake which should be unstake as its purpose is to reverse the deposit process and now get back LP<APT,stAPT> from DittoFarmingCoins:

Listing 3: products/ditto-farming/sources/ditto_farming.move (Lines 217,224)

```
211     fun unstake_lp_and_burn(
212         ditto_farming_coins: Coin<DittoFarmingCoin>,
213         ditto_farming_signer: &signer,
214     ) : Coin<LP<AptosCoin, StakedAptos, Stable>> acquires
 ↳ DittoFarmingCoinCaps {
215         // unstake amount of LP for given amount of
 ↳ DittoFarmingCoin
216         let farming_coin_amount = coin::value<DittoFarmingCoin>(&
 ↳ ditto_farming_coins);
217         liquidity_mining::stake<LP<AptosCoin, StakedAptos, Stable
 ↳ >>(
218             ditto_farming_signer,
219             farming_coin_amount,
220         );
221         // burn farming coin
222         let farming_account_addr = signer::address_of(
 ↳ ditto_farming_signer);
```

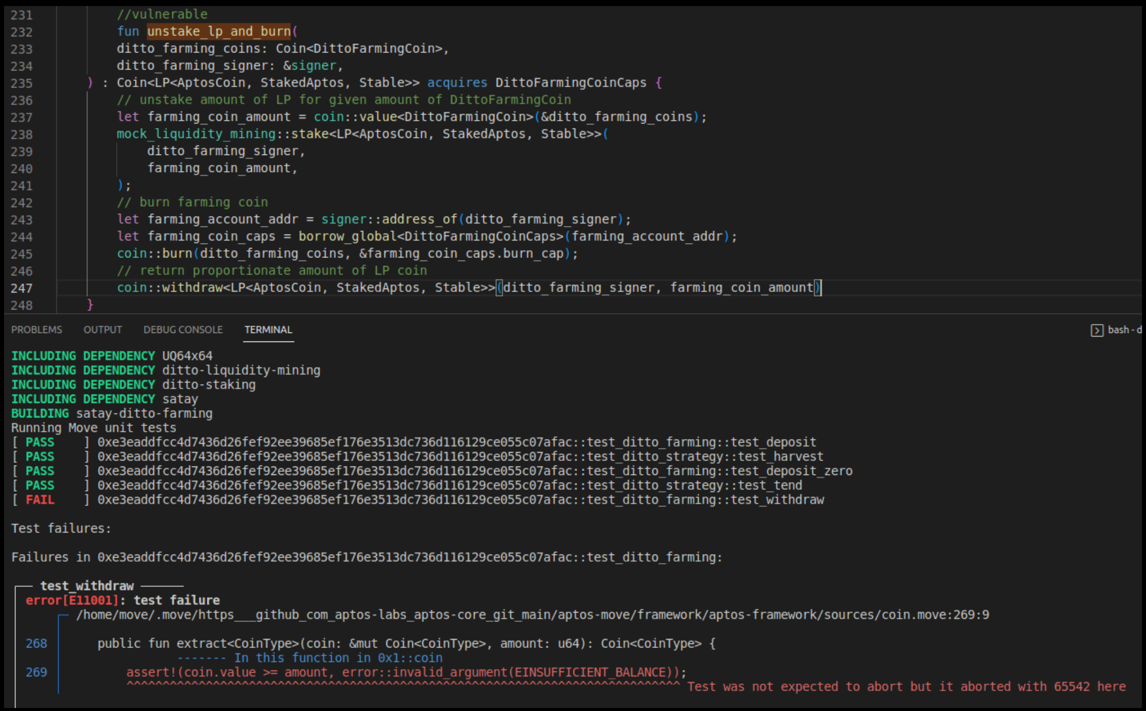
```
223         let farming_coin_caps = borrow_global<DittoFarmingCoinCaps
 ↳ >(farming_account_addr);
224         coin::burn(ditto_farming_coins, &farming_coin_caps.
 ↳ burn_cap);
225         // return proportionate amount of LP coin
226         coin::withdraw<LP<AptosCoin, StakedAptos, Stable>>(
 ↳ ditto_farming_signer, farming_coin_amount)
227     }
```

4. The liquidity_mining.move contains functions placeholders so far. This way, unit tests will not show unstaking error, as they normally would do in case of this type of coding error:

Listing 4: interfaces/ditto-lm-interface/sources/liquidity_mining.move

```
28      public entry fun stake<CoinType>(
29          _user: &signer,
30          _amount: u64,
31      ) {}
32
33      public entry fun unstake<CoinType>(
34          _user: &signer,
35          _amount: u64,
36      ) {}
```

Moreover, when examining the unit tests in **mock_ditto_farming.move**, it was noticed that they have correct implementation of the unstake_lp_and_burn function, so the issue was limited only to file **ditto_farming.move**. A proof of concept with aforementioned vulnerable implementation was created. It caused the withdraw function to fail, which is presented on the below screenshot. Note, that subroutine **mock_liquidity_mining::stake** was used instead of unimplemented **liquidity_mining::stake**.



```
//vulnerable
fun unstake_lp_and_burn(
ditto_farming_coins: Coin<DittoFarmingCoin>,
ditto_farming_signer: &signer,
) : Coin<LP<AptosCoin, StakedAptos, Stable>> acquires DittoFarmingCoinCaps {
    // unstake amount of LP for given amount of DittoFarmingCoin
    let farming_coin_amount = coin::value<DittoFarmingCoin>(&ditto_farming_coins);
    mock_liquidity_mining::stake<LP<AptosCoin, StakedAptos, Stable>>(
        ditto_farming_signer,
        farming_coin_amount,
    );
    // burn farming coin
    let farming_account_addr = signer::address_of(ditto_farming_signer);
    let farming_coin_caps = borrow_global<DittoFarmingCoinCaps>(farming_account_addr);
    coin::burn(ditto_farming_coins, &farming_coin_caps.burn_cap);
    // return proportionate amount of LP coin
    coin::withdraw<LP<AptosCoin, StakedAptos, Stable>>(ditto_farming_signer, farming_coin_amount)
}
```



```
INCLUDING DEPENDENCY UQ64x64
INCLUDING DEPENDENCY ditto-liquidity-mining
INCLUDING DEPENDENCY ditto-staking
INCLUDING DEPENDENCY satay
BUILDING satay-ditto-farming
Running Move unit tests
[ PASS    ] 0xe3eaddfcc4d7436d26fef92ee39685ef176e3513dc736d116129ce055c07afac::test_ditto_farming::test_deposit
[ PASS    ] 0xe3eaddfcc4d7436d26fef92ee39685ef176e3513dc736d116129ce055c07afac::test_ditto_strategy::test_harvest
[ PASS    ] 0xe3eaddfcc4d7436d26fef92ee39685ef176e3513dc736d116129ce055c07afac::test_ditto_farming::test_deposit_zero
[ PASS    ] 0xe3eaddfcc4d7436d26fef92ee39685ef176e3513dc736d116129ce055c07afac::test_ditto_strategy::test_tend
[ FAIL    ] 0xe3eaddfcc4d7436d26fef92ee39685ef176e3513dc736d116129ce055c07afac::test_ditto_farming::test_withdraw

Test failures:

Failures in 0xe3eaddfcc4d7436d26fef92ee39685ef176e3513dc736d116129ce055c07afac::test_ditto_farming:

┌── test_withdraw ──────
│ error[E11001]: test failure
│    ┌─ /home/move/.move/https___github_com_aptos-labs_aptos-core_git_main/aptos-move/framework/aptos-framework/sources/coin.move:269:9
│    │
│268 │     public fun extract<CoinType>(coin: &mut Coin<CoinType>, amount: u64): Coin<CoinType> {
│    │                ------- In this function in 0x1::coin
│269 │         assert!(coin.value >= amount, error::invalid_argument(EINSUFFICIENT_BALANCE));
│    │         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ Test was not expected to abort but it aborted with 65542 here
```

Figure 1: Withdraw failed.

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It is recommended to change stake function to unstake. The withdrawal process should be exactly opposite to the deposit process.

Remediation Plan:

SOLVED: The issue was fixed in commit 3fb4b8a.

3.2 (HAL-02) USERS ARE NOT PROTECTED AGAINST SLIPPAGE - MEDIUM

Description:

The strategy uses several operations which use **Liquidswap** to perform swap and adding or removing liquidity. All of these functions contains parameter which specify the potential slippage (e.g. minimal coins to be received after an operation). However, these values are usually hardcoded to be 1 or 0, very inefficient swaps or liquidity changes can be executed. In case of high volatility or low market depth, there will be nothing that will stop the swaps or liquidity additions/removal to be unfavorable for the users.

Code Location:

There is a call to add_liquidity in add_apt_st_apt_lp, where there are no enforcement of how many coins should be, in fact, added to the pool. As a result, less LP than expected may be received:

Listing 5: products/ditto-farming/sources/ditto_farming.move (Line 162)

```
153     fun add_apt_st_apt_lp(
154         aptos_coins: Coin<AptosCoin>,
155         staptos_coins: Coin<StakedAptos>,
156         product_address: address
157     ) : (Coin<LP<AptosCoin, StakedAptos, Stable>>, Coin<AptosCoin
 ↳ >) {
158         let (
159             residual_aptos_coins,
160             residual_staptos_coins,
161             lp
162         ) = add_liquidity<AptosCoin, StakedAptos, Stable>(
 ↳ aptos_coins, 0, staptos_coins, 0);
163 
164         if(coin::value(&residual_staptos_coins) == 0){
165             coin::destroy_zero(residual_staptos_coins);
166         } else {
```

```
167             coin::merge(
168                 &mut residual_aptos_coins,
169                 ditto_staking::exchange_staptos(
 ↳ residual_staptos_coins, product_address)
170             );
171         };
172
173         (lp, residual_aptos_coins)
174     }
```

In the call to remove_liquidity in liquidate_lp_coins, minimal values of coins out is set to 1. This way, operation result may be significantly unfavorable:

Listing 6: products/ditto-farming/sources/ditto_farming.move (Line 234)

```
230     fun liquidate_lp_coins(
231         lp_coins: Coin<LP<AptosCoin, StakedAptos, Stable>>
232     ) : Coin<AptosCoin> {
233         // remove liquidity for lp_coins
234         let (aptos_coin, staked_aptos) = remove_liquidity<
 ↳ AptosCoin, StakedAptos, Stable>(
235             lp_coins,
236             1,
237             1
238         );
239         // swap returned stAPT for APT
240         coin::merge(&mut aptos_coin, swap_stapt_for_apt(
 ↳ staked_aptos));
241         // return APT
242         aptos_coin
243     }
```

Second argument of function swap_stapt_for_apt stands for coin_out_min_val - minimum amount of coin Y to get out. Here it means that minimal value will be 0:

Listing 7: products/ditto-farming/sources/ditto_farming.move (Line 308)

```
304     fun swap_stapt_for_apt(staptos_coins: Coin<StakedAptos>) :
 ↳ Coin<AptosCoin> {
305         // swap on liquidswap AMM
306         swap_exact_coin_for_coin<StakedAptos, AptosCoin, Stable>(
307             staptos_coins,
308             0
309         )
310     }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

It is recommended to calculate minimal values of coins to receive when performing underlying swaps. While it is not possible to eliminate slippage completely, a maximal acceptable level should be indicated (e.g. 50%), and the minimal amount of coins to receive can be then calculated based of swap/liquidity operation input amounts.

Remediation Plan:

SOLVED: The issue was fixed in commit 3b4db50.

3.3 (HAL-03) INSUFFICIENT ROLE SEPARATION - LOW

Description:

Satay protocol utilizes multiple roles to manage different utilities. However, these roles have overlapping privileges, which means that some of them can manage multiple role types, which increases centralization. In case of a compromise, some roles might be especially sensitive as they have capability to influence not only self privileges, but also privileges dedicated for another roles.

Code Location:

Example of roles that can be managed by others are: vault_manager, strategist and keeper. Each of them is checked by respective functions named assert_[rolename] and that check is present in every occurrence of an access control including ability to transfer the role to another address. Below snippets present examples of these assert functions showing that some roles can control other roles as well.

vault_manager is controlled by governance as well:

Listing 8: sources/global_config.move (Lines 188-189)

```
182     public fun assert_vault_manager<BaseCoin>(
183         vault_manager: &signer
184     ) acquires GlobalConfigResourceAccount, GlobalConfig,
 ↳ VaultConfig {
185         let addr = signer::address_of(vault_manager);
186 
187         assert!(
188             get_governance_address() == addr ||
189             get_vault_manager_address<BaseCoin>() == addr,
190             ERR_NOT_MANAGER
191         );
192     }
```

strategist is controlled by governance and vault_manager too:

Listing 9: sources/global_config.move (Lines 202-203)

```
195     public fun assert_strategist<StrategyType: drop, BaseCoin>(
196         strategist: &signer
197     ) acquires GlobalConfigResourceAccount, GlobalConfig,
 ↳ VaultConfig, StrategyConfig {
198         let addr = signer::address_of(strategist);
199 
200         assert!(
201             get_governance_address() == addr ||
202             get_vault_manager_address<BaseCoin>() == addr ||
203             get_strategist_address<StrategyType>() == addr,
204             ERR_NOT_STRATEGIST
205         );
206     }
```

keeper is controlled by governance, vault_manager and strategist:

Listing 10: sources/global_config.move (Lines 215-218)

```
209     public fun assert_keeper<StrategyType: drop, BaseCoin>(
210         keeper: &signer
211     ) acquires GlobalConfigResourceAccount, GlobalConfig,
 ↳ VaultConfig, StrategyConfig {
212         let addr = signer::address_of(keeper);
213 
214         assert!(
215             get_governance_address() == addr ||
216             get_vault_manager_address<BaseCoin>() == addr ||
217             get_strategist_address<StrategyType>() == addr ||
218             get_keeper_address<StrategyType>() == addr,
219             ERR_NOT_KEEPER
220         );
221     }
```

Risk Level:

Likelihood - 1

Impact - 4

Recommendation:

It is recommended to remove multiple privileges from some roles and stick to a principle that one role is responsible only for their privilege types.

Remediation Plan:

SOLVED: The issue was fixed in commit 064080e.

3.4 (HAL-04) POTENTIAL INSECURE CALCULATIONS - LOW

Description:

The protocol's code contains calculations where a number is casted to a smaller type, which opens up the possibility for overflows. Below examples shows places in code, which in case of edge scenarios (e.g. extremely high values being placed as input), overflows may occur.

Code Location:

`calculate_share_amount_from_base_coin_amount` function returns `u64`, but in line 204 highlighted below it returns result of multiplication `u64 * u64` divided by another factor. Note, that since a `mul_div` function is already created in **math** module, it can be reused instead of coding the equation manually:

Listing 11: sources/vault.move (Line 204)

```
196     public fun calculate_share_amount_from_base_coin_amount<
 ↳ BaseCoin>(
197         vault_cap: &VaultCapability,
198         base_coin_amount: u64,
199     ): u64 acquires Vault, CoinStore {
200         let total_base_coin_amount = total_assets<BaseCoin>(
 ↳ vault_cap);
201         let total_supply = option::get_with_default<u128>(&coin::
 ↳ supply<VaultCoin<BaseCoin>>(), 0);
202 
203         if (total_supply != 0) {
204             (total_supply as u64) * base_coin_amount /
 ↳ total_base_coin_amount
205         } else {
206             base_coin_amount
207         }
208     }
```

`calculate_base_coin_amount_from_share` function casts value `share_total_supply` of type `u128` in line `192` to `u64` in line `193`. It is considered insecure to downcast variables as it is likely that an overflow can occur, if the first value is greater than `u64::MAX`:

Listing 12: sources/vault.move (Lines 192,193)

```
186     public fun calculate_base_coin_amount_from_share<BaseCoin>(
187         vault_cap: &VaultCapability,
188         share: u64
189     ): u64 acquires Vault, CoinStore {
190         let total_assets = total_assets<BaseCoin>(vault_cap);
191         let share_total_supply_option = coin::supply<VaultCoin<
 ↳ BaseCoin>>();
192         let share_total_supply = option::get_with_default<u128>(&
 ↳ share_total_supply_option, 0);
193         math::mul_div(total_assets, share, (share_total_supply as
 ↳ u64))
194     }
```

`get_user_amount` function contains both patterns described previously - it casts `u128` to `u64` (line 709) and, additionally, in the same line multiplies two `u64` to return an `u64`. Note, that `mul_div` could be used there instead, too:

Listing 13: sources/vault.move (Lines 709,710)

```
702     public fun get_user_amount<BaseCoin>(
703         vault_cap: &VaultCapability,
704         user_addr: address
705     ): u64 acquires Vault, CoinStore {
706         let total_assets = total_assets<BaseCoin>(vault_cap);
707         let user_share_amount = coin::balance<VaultCoin<BaseCoin
 ↳ >>(user_addr);
708         let share_total_supply_option = coin::supply<VaultCoin<
 ↳ BaseCoin>>();
709         let share_total_supply = option::get_with_default<u128>(&
 ↳ share_total_supply_option, 0);
710         total_assets * user_share_amount / (share_total_supply as
 ↳ u64)
711     }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended no to downcast numbers to smaller size variable to avoid overflows. Casting should be done to equal or larger size variables.

Remediation Plan:

SOLVED: The issue was fixed in commit a6dfc86.

3.5 (HAL-05) POTENTIAL OVERFLOWS IN MATH MODULE - LOW

Description:

The **math** module contains implementation of mathematical functions. Most of them are simple calculations, e.g.: multiplication and division. However, using these implementations as intended does not protect against overflows, and even legitimate use of these functions may lead to their result being overflown. This in turn may make present and future utilities relying on **math** module not work. It should be known that there are some limitations that may affect result values of module's functions if supplied arguments are enormously large.

Code Location:

Below snippets shows examples of extreme value cases, where an overflow in the function is possible. Executing these functions (e.g. as a unit test) will cause the VM to throw a math error (overflow):

In mul_div function, overflow occurs, if input values are large enough. This is mainly due to the return value being expected to fit in u64, while it should be capable of holding up to u64::MAX * u64::MAX / 1:

Listing 14: sources/math.move (Lines 33-34)

```
31      public fun mul_div(x: u64, y: u64, z: u64): u64 {
32          assert!(z != 0, ERR_DIVIDE_BY_ZERO);
33          let r = (x as u128) * (y as u128) / (z as u128);
34          (r as u64)
```

In mul_div_u128 function, overflow has even more chance of occurring. The function performs similar calculation as above, with a difference that the entry arguments are of type u128 and the result is downcasted to u64. The return value is being expected to fit in u64, while it should be capable of holding up to u128::MAX * u128::MAX / 1:

Listing 15: sources/math.move (Lines 40-41)

```
38      public fun mul_div_u128(x: u128, y: u128, z: u128): u64 {
39          assert!(z != 0, ERR_DIVIDE_BY_ZERO);
40          let r = x * y / z;
41          (r as u64)
42      }
```

In pow_10 function, overflow occurs if input value is larger than 19. Exponential calculation can easily lead to large numbers in result; therefore, it is advised to set a maximum return value (e.g. u256::MAX) and disallow input that leads to larger results to prevent overflows:

Listing 16: sources/math.move (Lines 70,83)

```
70      public fun pow_10(degree: u8): u64 {
71          let res = 1;
72          let i = 0;
73          while ({
74              spec {
75                  invariant res == spec_pow(10, i);
76                  invariant 0 <= i && i <= degree;
77              };
78              i < degree
79          }) {
80              res = res * 10;
81              i = i + 1;
82          };
83          res
84      }
```

Risk Level:

Likelihood - 1

Impact - 3

Recommendation:

When using these functions, their edge cases should be always considered, preventing unexpected application behavior. For example, if a function will be used to process enormously large numbers, it might be worth modifying it by casting the variables used in calculation and return to larger ones.

Remediation Plan:

SOLVED: The issue was fixed in commits a6dfc86 and de2d2b6.

3.6 (HAL-06) MISSING EVENT EMISSION - INFORMATIONAL

Description:

The protocol does not implement event emission on key state-changing operations. Event assures better traceability of application state and in case a security incident occurs, it is easier to track the root cause by tracking the key operations. The best practice is to emit events on key operations that change the state of the protocol. For Satay protocol, this would be especially any withdraw and deposit operations from external accounts to the protocol, change of role ownership, and optionally other cashflows e.g. between vaults and strategies. For ditto_farming and ditto_farming_strategy, this could be deposits, withdrawals, tends and harvests. Currently, events are implemented only in satay::dao_storage module.

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to implement event emissions on operations such as role ownership changes, depositing and withdrawing funds by users, optionally any cashflows between strategies and vaults.

Remediation Plan:

SOLVED: The issue was fixed in commit d0eb577.

3.7 (HAL-07) VAULTS AND STRATEGIES CANNOT BE DELISTED - INFORMATIONAL

Description:

Once a strategy and a vault is initialized, there is no way to remove it. There are two scenarios where such utility may be useful:

1. When a certain strategy suffers a security incident and while it is available, it poses additional threat to users, who will join it after the incident occurs.
2. When over time the number of vaults and strategies grows, and it might be desired to delist unused or unprofitable products or vaults.

Currently, some workarounds might be applied, e.g. decreasing vault or strategy debt ratio to 0% should disallow its further use, while the graphical delisting can be done on front-end layer.

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It might be worth considering implementing delisting logic at some point. However, it should be implemented carefully having in mind, that anything prepared to be delisted should still allow users to release any funds that are still locked, and the delisting should occur in a Multisig or DAO decision when no more funds are pending release from the vault/strategy to be delisted.

Remediation Plan:

SOLVED: The issue was fixed in commit 099640b.

3.8 (HAL-08) MATH MODULE IS RARELY USED - INFORMATIONAL

Description:

The protocol's codebase includes a Math module. However, the module is used only in two calculations in vault.move, while it could have more use cases throughout the codebase. Additional codebase costs additional gas to deploy, as well as standardizing code in form of reusable libraries/functions increases clarity as well as security - as long as a reused function is tested for vulnerabilities, it is better to reuse it in code for specific tasks it is designed to do, instead of implementing that task using different code a time. In this way, it is easier to predict potential behavior of the code and manage potential vulnerabilities by auditing and patching just one function instead of numerous implementations of the same task.

Code Location:

The following code snippets from Satay protocol include calculations that could utilize math functions instead of implementing them from scratch:

In calculate_share_amount_from_base_coin_amount function from **vault.move**, mul_div could have been used:

Listing 17: sources/vault.move (Line 204)

```
196     public fun calculate_share_amount_from_base_coin_amount<
 ↳ BaseCoin>(
197         vault_cap: &VaultCapability,
198         base_coin_amount: u64,
199     ): u64 acquires Vault, CoinStore {
200         let total_base_coin_amount = total_assets<BaseCoin>(
 ↳ vault_cap);
201         let total_supply = option::get_with_default<u128>(&coin::
 ↳ supply<VaultCoin<BaseCoin>>(), 0);
202 
203         if (total_supply != 0) {
```

```
204             (total_supply as u64) * base_coin_amount /
 ↳ total_base_coin_amount
205         } else {
206             base_coin_amount
207         }
208     }
```

In assess_fees function from **vault.move**, mul_div could have been used:

Listing 18: sources/vault.move (Line 405)

```
385     fun assess_fees<StrategyType: drop, BaseCoin>(
386         profit: &Coin<BaseCoin>,
387         vault_cap: &VaultCapability,
388         _witness: &StrategyType
389     ) acquires VaultStrategy, Vault, CoinStore, VaultCoinCaps {
390         let vault = borrow_global<Vault>(vault_cap.vault_addr);
391         let strategy = borrow_global_mut<VaultStrategy<
 ↳ StrategyType>>(vault_cap.vault_addr);
392 
393         let duration = timestamp::now_seconds() - strategy.
 ↳ last_report;
394         let gain = coin::value(profit);
395 
396         if (duration == 0 || gain == 0) {
397             return
398         };
399 
400         let management_fee_amount = strategy.total_debt
401             * duration
402             * vault.management_fee
403             / MAX_DEBT_RATIO_BPS
404             / SECS_PER_YEAR;
405         let performance_fee_amount = gain * vault.performance_fee
 ↳ / MAX_DEBT_RATIO_BPS;
406 
407         let total_fee_amount = management_fee_amount +
 ↳ performance_fee_amount;
408         if (total_fee_amount > gain) {
409             total_fee_amount = gain;
410         };
```

In report_loss function from **vault.move**, mul_div could have been used:

Listing 19: sources/vault.move (Line 454)

```
445     public(friend) fun report_loss<StrategyType: drop>(
446         vault_cap: &VaultCapability,
447         loss: u64,
448         _witness: &StrategyType
449     ) acquires Vault, VaultStrategy {
450         let vault = borrow_global_mut<Vault>(vault_cap.vault_addr)
 ↳ ;
451         let strategy = borrow_global_mut<VaultStrategy<
 ↳ StrategyType>>(vault_cap.vault_addr);
452 
453         if (vault.debt_ratio != 0) {
454             let ratio_change = loss * vault.debt_ratio / vault.
 ↳ total_debt;
455             if (ratio_change > strategy.debt_ratio) {
456                 ratio_change = strategy.debt_ratio;
457             };
458             strategy.debt_ratio = strategy.debt_ratio -
 ↳ ratio_change;
459             vault.debt_ratio = vault.debt_ratio - ratio_change;
460         };
```

In credit_available function from **vault.move**, mul_div could have been used:

Listing 20: sources/vault.move (Line 571)

```
558     public fun credit_available<StrategyType: drop, BaseCoin>(
559         vault_cap: &VaultCapability
560     ): u64 acquires Vault, VaultStrategy, CoinStore {
561         let vault = borrow_global<Vault>(vault_cap.vault_addr);
562         assert!(vault.base_coin_type == type_info::type_of<
 ↳ BaseCoin>(), ERR_COIN);
563 
564         let vault_debt_ratio = vault.debt_ratio;
565         let vault_total_debt = vault.total_debt;
566         let vault_total_assets = total_assets<BaseCoin>(vault_cap)
 ↳ ;
567         let vault_debt_limit = vault_debt_ratio *
 ↳ vault_total_assets / MAX_DEBT_RATIO_BPS;
568 
```

```
569         let strategy = borrow_global<VaultStrategy<StrategyType>>(
 ↳ vault_cap.vault_addr);
570
571         let strategy_debt_limit = strategy.debt_ratio *
 ↳ vault_total_assets / MAX_DEBT_RATIO_BPS;
```

In get_user_amount function from **vault.move**, mul_div could have been used:

Listing 21: sources/vault.move (Line 710)

```
702     public fun get_user_amount<BaseCoin>(
703         vault_cap: &VaultCapability,
704         user_addr: address
705     ): u64 acquires Vault, CoinStore {
706         let total_assets = total_assets<BaseCoin>(vault_cap);
707         let user_share_amount = coin::balance<VaultCoin<BaseCoin
 ↳ >>(user_addr);
708         let share_total_supply_option = coin::supply<VaultCoin<
 ↳ BaseCoin>>();
709         let share_total_supply = option::get_with_default<u128>(&
 ↳ share_total_supply_option, 0);
710         total_assets * user_share_amount / (share_total_supply as
 ↳ u64)
711     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to reuse the functions from math library instead of implementing the calculations from scratch each time.

Remediation Plan:

SOLVED: The issue was fixed in commit b587d56.

3.9 (HAL-09) FEES VALUE INCONSISTENCY - INFORMATIONAL

Description:

The vault module implements a constant for maximal management and performance fees. The number states 50%, but the comment states 30%, which means that the number might be incorrect.

Code Location:

The fee value declaration can be found in the first lines of vault.move as per below snippet:

Listing 22: sources/vault.move

```
20      const MAX_MANAGEMENT_FEE: u64 = 5000; // 30%
21      const MAX_PERFORMANCE_FEE: u64 = 5000; // 30%
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to verify the proper value and fix either that value or the comment.

Remediation Plan:

SOLVED: The issue was fixed in commit 6c830d6.

3.10 (HAL-10) UPDATING STRATEGY DEBT RATIO RETURNS OLD VALUE - INFORMATIONAL

Description:

It was noticed that update_strategy_debt_ratio function returns the old value (the one that was valid before update) instead of the updated one. Currently, the return value of that function is not used anywhere, but it might not be intuitive to expect the update function to returns old values (which are supposed to be abandoned during update). Thus, it might introduce a confusion or a vulnerability in the future.

Code Location:

Listing 23: sources/vault.move (Line 276)

```
261     public(friend) fun update_strategy_debt_ratio<StrategyType:
 ↳ drop>(
262         vault_cap: &VaultCapability,
263         debt_ratio: u64,
264         _witness: &StrategyType
265     ): u64 acquires Vault, VaultStrategy {
266         let vault = borrow_global_mut<Vault>(vault_cap.vault_addr)
 ↳ ;
267         let strategy = borrow_global_mut<VaultStrategy<
 ↳ StrategyType>>(vault_cap.vault_addr);
268         let old_debt_ratio = strategy.debt_ratio;
269 
270         vault.debt_ratio = vault.debt_ratio - old_debt_ratio +
 ↳ debt_ratio;
271         strategy.debt_ratio = debt_ratio;
272 
273         // check if the strategy's updated debt ratio is valid
274         assert!(vault.debt_ratio <= MAX_DEBT_RATIO_BPS,
 ↳ ERR_INVALID_DEBT_RATIO);
275 
276         old_debt_ratio
277     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to consider returning the new, updated value instead of the old one.

Remediation Plan:

SOLVED: The issue was fixed in commit f9ce4bd.

3.11 (HAL-11) CREDIT THRESHOLD IS NEVER USED - INFORMATIONAL

Description:

Multiple modules contain references to a variable named `credit_threshold`, which includes extensive logic related to getters and setters of that value. However, this variable is not used anywhere in the code in any other logic. Consequently, it either causes the module codebase to be unnecessarily larger or there is some missing logic related to it that should be yet implemented.

The below listing contains modules in which reference to that variable exists along with a function name and a line number, at which the variable is accessed.

- satay.move:
  - update_strategy_credit_threshold, satay.move#L228
- vault.move
  - struct VaultStrategy, vault.move#L72
  - approve_strategy, vault.move#L249
  - update_strategy_credit_threshold, vault.move#L296
  - credit_threshold, vault.move#L675
- base_strategy.move
  - update_credit_threshold, base_strategy.move#L307
- ditto_farming_strategy.move
  - update_credit_threshold, ditto_farming_strategy.move#L288

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to either implement missing logic or remove any reference to the `credit_threshold` variable.

Remediation Plan:

SOLVED: The issue was fixed in commit de2d2b6.

THANK YOU FOR CHOOSING

// HALBORN