BEOSIN
Blockchain Security

Smart contract security audit report

Audit Number：202101191405

Report Query Name: RAMP_VERSION_TEZOS

Project Name:

RAMP_VERSION_TEZOS

Project Link:

URL: https://github.com/RAMP-DEFI/RAMP_VERSION_TEZOS

Origin commit id: 7300d4a06864ad28f96cac9db8e4322535475155

Final commit id: 61aecb23d20480f56aa98e6e1023f874e7e1a516

Start Date：2020.12.10

Completion Date：2021.01.19

Overall Result：Pass

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	FA1.2 Token Standard Audit	Pass
		Operator Use Audit	Pass
		Redundant Code	Pass
		Slice Operation Security	Pass
		Module Import Audit	Pass
		Use non-int type as Index	Pass
		sp.verify Using Audit	Pass
		Division by zero Check	Pass
2	Function Call Audit	Visibility Decorator	Pass
		Authorization of Function Call	Pass
		Argument Setting Check	Pass
		sp.transfer Using Check	Pass
3	Common Vulnerability	Pseudo-random Number Generator (PRNG)	Pass

		Array Index Out of Bounds	Pass
		DoS (Denial of Service)	Pass
4	Business Security	Access Control of Owner	Pass
		Business Logic Audit	Pass
		Business Implementation Audit	Pass

Note: Audit results and suggestions in code comments

Disclaimer: This audit is only applied to the type of auditing specified in this report and the scope of given in the results table. Other unknown security vulnerabilities are beyond auditing responsibility. Beosin (Chengdu LianAn) Technology only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, Beosin (Chengdu LianAn) Technology lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Beosin (Chengdu LianAn) Technology before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted; if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Beosin (Chengdu LianAn) Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin (Chengdu LianAn) Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin (Chengdu LianAn). Due to the technical limitations of any organization, this report conducted by Beosin (Chengdu LianAn) still has the possibility that the entire risk cannot be completely detected. Beosin (Chengdu LianAn) disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin (Chengdu LianAn).

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of RAMP_VERSION_TEZOS project, including Coding Standards, Security, and Business Logic. **The RAMP_VERSION_TEZOS project passed all audit items. The overall result is Pass.** The detailed audit information of this project is shown following.

1. Coding Conventions

Check whether the code style conforms to Python and FA1.2 code style.

1.1 FA1.2 Token Standard Audit

- **Description:** Check whether the token module code used in the current contract conforms to the FA1.2 token standard.

- **Result:** Pass

1.2 Operator Use Audit

- **Description:** Check whether the operators used in this contract are reasonable, and avoid unexpected numerical results caused by incorrect use of operators.
- **Result:** Pass

1.3 Redundant Code

- **Description:** Check whether the contract code has redundant codes.
- **Result:** Pass

1.4 Slice Operation Security

- **Description:** Check whether the slice operations in contract are safe.
- **Result:** Pass

1.5 Module import audit

- **Description:** Check whether the imported modules are reasonable, avoid issues of invalid import, repeat import and wrong import.
- **Result:** Pass

1.6 Use non-int Type as Index

- **Description:** Check whether the contract code uses the non-int type data as an index, it will make the data cannot be normally got.
- **Result:** Pass

1.7 sp.verify Usage

- **Description:** Check the use reasonability of *sp.verify* in the contract.
- **Result:** Pass

1.9 Division by zero Check

- **Description:** Check whether the Division by zero vulnerability exists in the contract and effects the contract function.
- **Result:** Pass

2. Function Call Audit

Check whether the function implementation in the contract has security risks.

2.1 Visibility decorator

- **Description:** Check whether the visibility decorator *@sp.entry_point* is set correctly.
- **Result:** Pass

2.2 Authorization of Function Call

- **Description:** Check whether there is a caller authorization check for key functions in the contract.
- **Result:** Pass

2.3 Argument Setting Check

- **Description:** Check whether the parameters of the function are set correctly.

- **Result:** Pass

2.4 sp.transfer Using Check

- **Description:** Check whether the function *sp.transfer* used in this contract has security risks.
- **Result:** Pass

3. General Vulnerability

Check whether the general vulnerabilities exist in the contract.

3.1 Pseudo-random Number Generator (PRNG)

- **Description:** Whether the results of random numbers can be predicted.
- **Result:** Pass

3.2 Array Index Out of Bounds

- **Description:** Check whether the arrays in contract has possibility of existing array index out of bounds exception.
- **Result:** Pass

3.3 DoS (Denial of Service)

- **Description:** Check whether DoS attack exists in the contract.
- **Result:** Pass

4. Business Security

Project Description

The RAMP_VERSION_TEZOS project contains 4 contracts. The *w_tezos_token.py* contract creates a token wTezos, which is a standard FA1.2 token but cannot be freely traded by ordinary users. This token confirms delegation of XTZ for staking by some user; *deposit_contract.py* is the contract for performing deposit-related operations, used to create wTezos for each delegated XTZ and sends them to the user; *staking_manager.py* is the stake management contract, which implements the receiving user's stake and the distribution of stake rewards; *utils.py* contract is mainly used for testing.

4.1 w_tezos_token

(1) Mint

- **Description:** The *transferOwner* of wTezos token has the authority to mint wTezos. No cap is set for the circulation of wTezos.
- **Related functions:** *mint, is_transfer_owner*
- **Result:** Pass

(2) Transfer

- **Description:** Ordinary users cannot trade wTezos. Only addresses with *transferOwner* authority can trade wTezos. This authority can transfer any number of tokens from any address to the specified target address without pre-approve. The whitelist addresses, contract owner, and operator are granted *transferOwner* authority as default.

- **Related functions:** *transfer, is_transfer_owner*
- **Result:** Pass

(3) Burn

- **Description:** The *transferOwner* authority can destroy the wTezos of the specified user. This operation is designed to destroy the corresponding amount of wTezos when the user withdraw deposited XTZ. Note that because both *transfer* and *burn* can be called directly, if the user's wTezos balance is less than the number of XTZ in the deposit to be withdrawn, the withdrawal will fail.
- **Related functions:** *burn, is_transfer_owner*
- **Result:** Pass

(4) Authority management

- **Description:** The contract has three privileged roles: administrator, operator, and whitelist. The administrator can grant authority to the specified address. The operator is the address of the deposit contract. These three roles are included in the authority *transferOwner*, and the *transferOwner* authority has the right to call *mint, burn,* and *transfer* functions.
- **Related functions:** *setAdministrator, set_operator, add_to_whitelist, remove_from_whitelist*
- **Result:** Pass

(5) Unavailable function

- **Description:** Because ordinary users cannot trade wTezos normally, the *transfer* function does not perform approve detection, and the *approve* function is unavailable. The *pause* function is also unavailable, and it is recommended to delete the pause related code directly.
- **Related functions:** *approve, pause*
- **Result:** Pass

4.2 deposit_contract

(1) Deposit

- **Description:** The user transfers XTZ to this contract. This contract transfers the user's XTZ to the *staking_manager* contract and calls the *mint* function of the *w_tezos_token* contract to issue the same amount of wTezos tokens and transfer them to the deposit user. Finally, the deposit contract will record the user's deposit information.
- **Related functions:** *deposit*
- **Result:** Pass

(2) Withdraw

- **Description:** Deposit users can call the *withdraw* function to withdraw their deposited XTZ. The corresponding XTZ will be transferred to the user's address immediately without any delay. At the same time, this contract will call the *burn* function of the *w_tezos_token* contract to destroy the corresponding amount of wTezos tokens. Note that if the user's wTezos tokens are insufficient, the withdrawal will throw an exception.
- **Related functions:** *withdraw*
- **Result:** Pass

(3) Claim reward

- **Description:** Deposit users can claim their rewards through the *claim_reward* function of this contract. In fact, this contract will call the *claim_reward* function of the *staking_manager* contract, and all operations are completed in the *claim_reward* function of the *staking_manager* contract.
- **Related functions:** *claim_reward*
- **Result:** Pass

(4) Pause

- **Description:** The owner can call the *pause* function to set the pause state. When the pause state is True, the user cannot call *deposit*, *withdraw*, and *claim_reward* functions.
- **Related functions:** *pause, deposit, withdraw, claim_reward*
- **Result:** Pass

(5) Change eth address

- **Description:** The *change_eth_address* function in the *deposit_contract* contract is used to set the user's address on Ethereum. This contract will call the *change_eth_address* function of the *staking_manager* contract, and all operation are completed in the *change_eth_address* function of the *staking_manager* contract.
- **Related functions:** *change_eth_address*
- **Result:** Pass

4.3 staking_manager

(1) Create deposit

- **Description:** The operator (deposit contract) address will call the *create_deposit* function to add a deposit record of the user. This function will create a corresponding delegate contract for each new user. The XTZ sent to *staking_manager* contract when the user deposits will be transferred to the corresponding delegate contract, and the corresponding XTZ reward will also be transferred to the contract address.

- **Related functions:** *create_deposit*
- **Result:** Pass

(2) Process reward

- **Description:** The backend address can call the *process_reward* function, which will retrieve the rewarded XTZ from the user's delegate contract, and then update the user's reward information. Each staker's XTZ reward is divided into 3 parts, the first part is the team fee (the default is 10%, the percentage can be modified), the remaining XTZ is divided into two parts, the second part is the trade-in (the default is the remaining XTZ 30%, the percentage can be modified), the third part is the remaining rewards for the staker.
- **Related functions:** *process_reward*
- **Result:** Pass

(3) Claim reward

- **Description:** The staker can call the *claim_reward* function to withdraw the reward. Note that this function can be executed by other addresses, and the reward is still transferred to the original address.
- **Related functions:** *claim_reward*
- **Result:** Pass

(4) Withdraw tradein fee

- **Description:** The backend address can withdraw all trade-in fees through the *withdraw_tradein_fee* function and transferred them to the trade-in address.
- **Related functions:** *withdraw_tradein_fee*
- **Result:** Pass

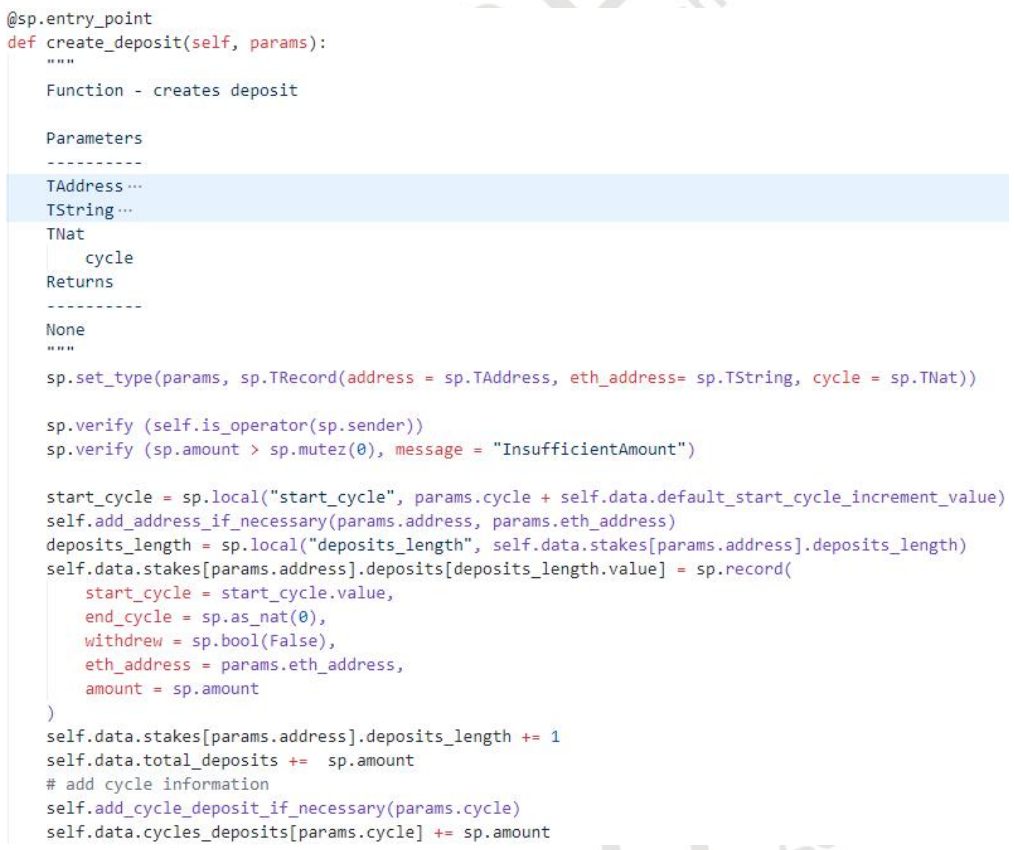
(5) Withdraw team fee

- **Description:** The backend address can withdraw all team fees through the *withdraw_team_fee* function and transferred them to the team fee address.
- **Related functions:** *withdraw_team_fee*
- **Result:** Pass

5. Details of audit results

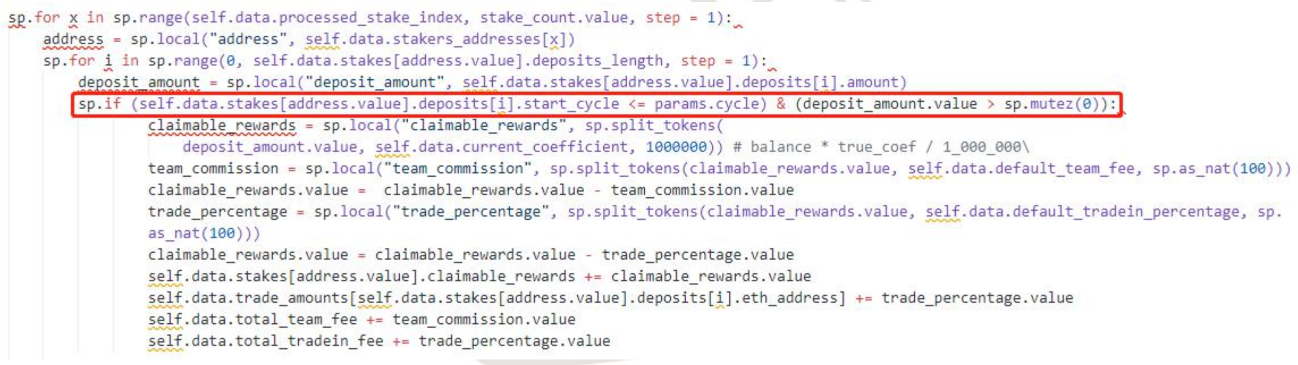
5.1 *create_deposit* function can specify start cycle (commit: f08cb6c)

- **Description:** In the deposit function *create_deposit*, the start cycle can be specified by the user. There is such a scenario: User executes the deposit during the *process_reward* process, and the start cycle specified when calling does not exceed *params.cycle-default_start_cycle_increment_value*, it can cause the *current_coefficient* and the current *full_active_deposits* correspond to the coefficient inconsistency (a little larger than the actual coefficient), causing the *process_reward* exception.



```
@sp.entry_point
def create_deposit(self, params):
    """
    Function - creates deposit

    Parameters
    ----------
    TAddress ···
    TString ···
    TNat
        cycle
    Returns
    ----------
    None
    """
    sp.set_type(params, sp.TRecord(address = sp.TAddress, eth_address= sp.TString, cycle = sp.TNat))

    sp.verify (self.is_operator(sp.sender))
    sp.verify (sp.amount > sp.mutez(0), message = "InsufficientAmount")

    start_cycle = sp.local("start_cycle", params.cycle + self.data.default_start_cycle_increment_value)
    self.add_address_if_necessary(params.address, params.eth_address)
    deposits_length = sp.local("deposits_length", self.data.stakes[params.address].deposits_length)
    self.data.stakes[params.address].deposits[deposits_length.value] = sp.record(
        start_cycle = start_cycle.value,
        end_cycle = sp.as_nat(0),
        withdrew = sp.bool(False),
        eth_address = params.eth_address,
        amount = sp.amount
    )
    self.data.stakes[params.address].deposits_length += 1
    self.data.total_deposits +=  sp.amount
    # add cycle information
    self.add_cycle_deposit_if_necessary(params.cycle)
    self.data.cycles_deposits[params.cycle] += sp.amount
```

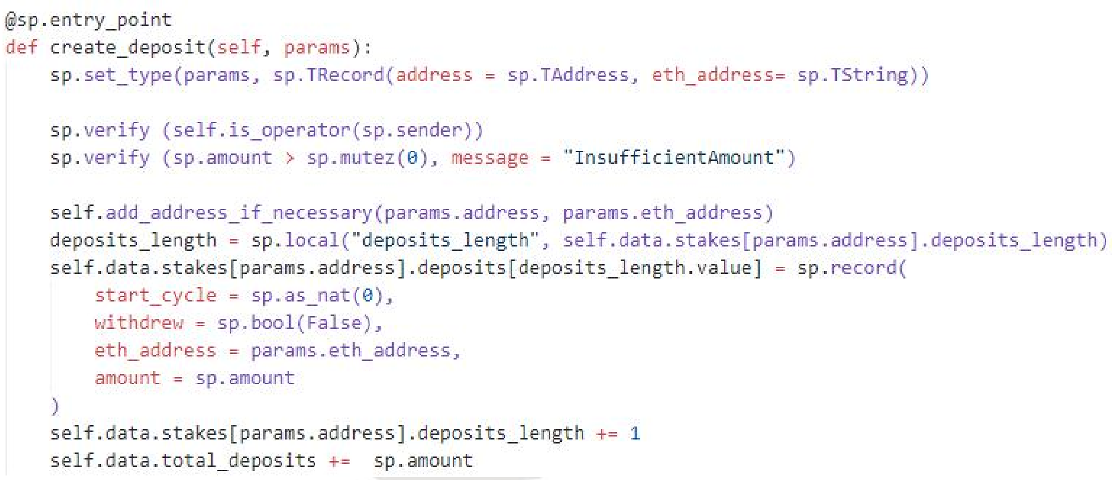
Figure 1 The origin source code of function *create_deposit*



```
sp.for x in sp.range(self.data.processed_stake_index, stake_count.value, step = 1):
    address = sp.local("address", self.data.stakers_addresses[x])
    sp.for i in sp.range(0, self.data.stakes[address.value].deposits_length, step = 1):
        deposit_amount = sp.local("deposit_amount", self.data.stakes[address.value].deposits[i].amount)
        sp.if (self.data.stakes[address.value].deposits[i].start_cycle <= params.cycle) & (deposit_amount.value > sp.mutez(0)):
            claimable_rewards = sp.local("claimable_rewards", sp.split_tokens(
                deposit_amount.value, self.data.current_coefficient, 1000000)) # balance * true_coef / 1_000_000\
            team_commission = sp.local("team_commission", sp.split_tokens(claimable_rewards.value, self.data.default_team_fee, sp.as_nat(100)))
            claimable_rewards.value =  claimable_rewards.value - team_commission.value
            trade_percentage = sp.local("trade_percentage", sp.split_tokens(claimable_rewards.value, self.data.default_tradein_percentage, sp.
            as_nat(100)))
            claimable_rewards.value = claimable_rewards.value - trade_percentage.value
            self.data.stakes[address.value].claimable_rewards += claimable_rewards.value
            self.data.trade_amounts[self.data.stakes[address.value].deposits[i].eth_address] += trade_percentage.value
            self.data.total_team_fee += team_commission.value
            self.data.total_tradein_fee += trade_percentage.value
```

Figure 2 The origin source code of function *process_reward*

- **Fix Result:** Fixed. The final code is shown below.

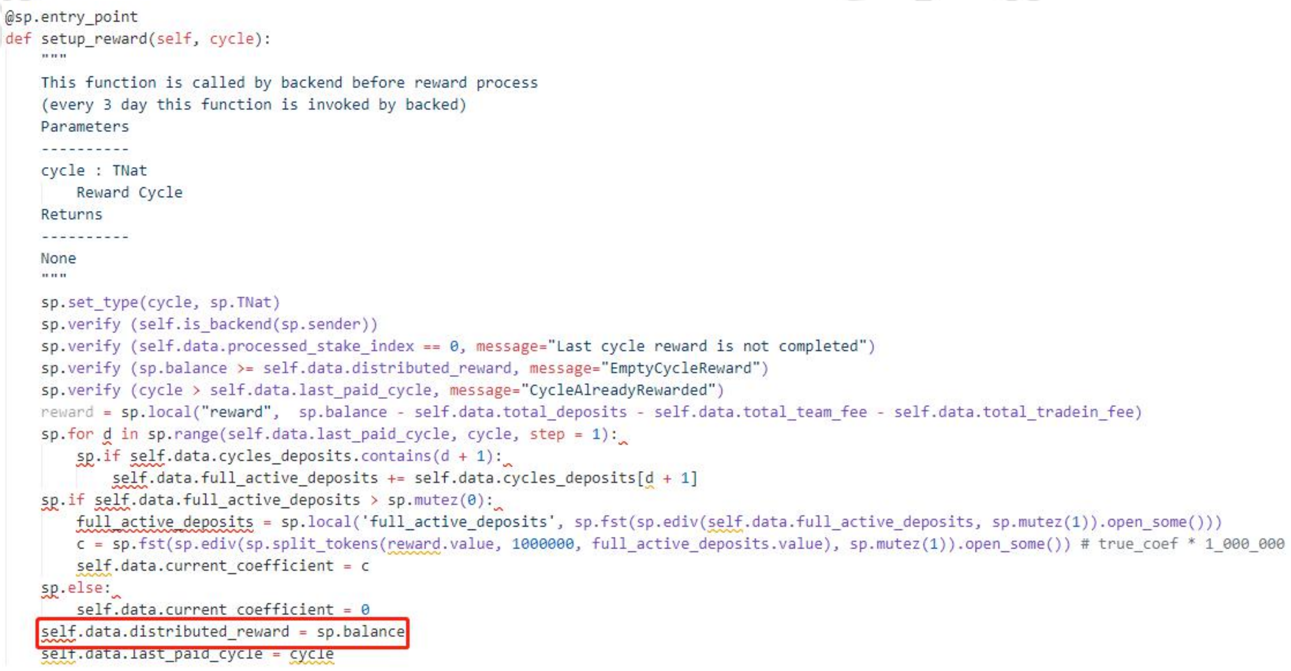


```
@sp.entry_point
def create_deposit(self, params):
    sp.set_type(params, sp.TRecord(address = sp.TAddress, eth_address= sp.TString))

    sp.verify (self.is_operator(sp.sender))
    sp.verify (sp.amount > sp.mutez(0), message = "InsufficientAmount")

    self.add_address_if_necessary(params.address, params.eth_address)
    deposits_length = sp.local("deposits_length", self.data.stakes[params.address].deposits_length)
    self.data.stakes[params.address].deposits[deposits_length.value] = sp.record(
        start_cycle = sp.as_nat(0),
        withdrew = sp.bool(False),
        eth_address = params.eth_address,
        amount = sp.amount
    )
    self.data.stakes[params.address].deposits_length += 1
    self.data.total_deposits +=  sp.amount
```

Figure 3 The final source code of function *create_deposit*

5.2 Wrong check about *distributed_reward* (commit: f08cb6c)

- **Description:** After the *setup_reward* function is executed successfully (*self.data.distributed_reward* has been set to *sp.balance*), the staker calls the *withdraw* function to directly withdraw the corresponding deposit. When *process_reward* function is called later, an exception will be thrown because 'sp.verify (sp.balance >= self.data.distributed_reward, message="EmptyCycleReward")' fails the verification. In the same way, calling *withdraw* after *process_reward* will also affect the next *setup_reward*.



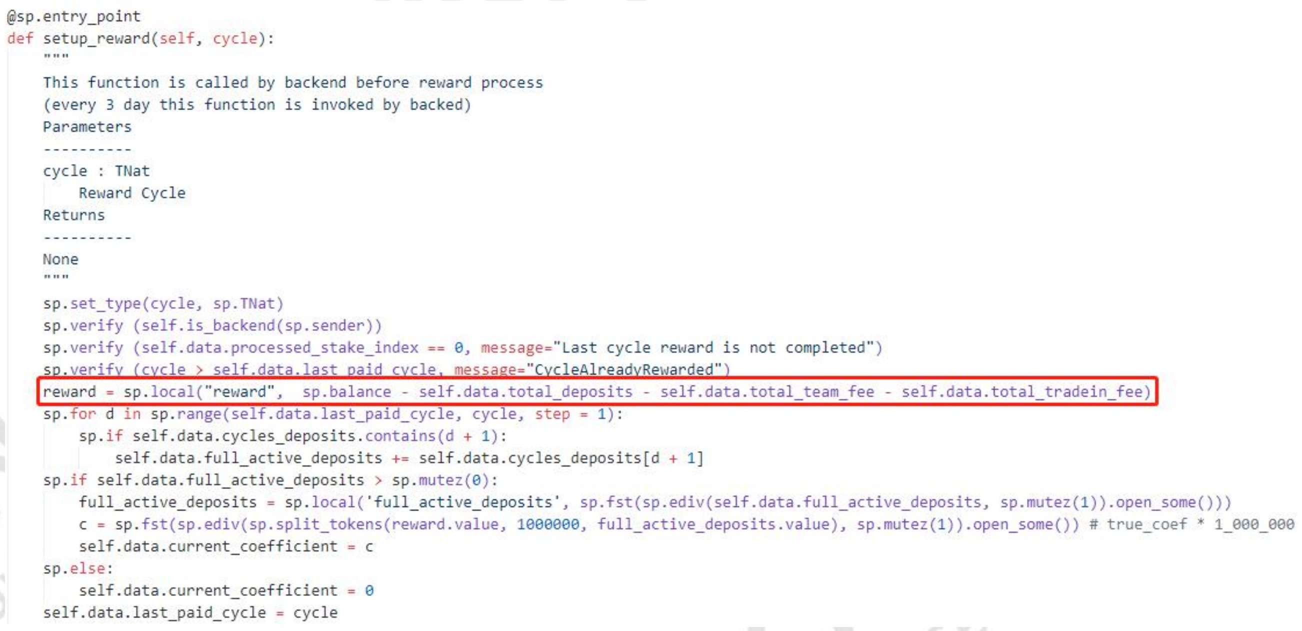
```
@sp.entry_point
def setup_reward(self, cycle):
    """
    This function is called by backend before reward process
    (every 3 day this function is invoked by backed)
    Parameters
    ----------
    cycle : TNat
        Reward Cycle
    Returns
    ----------
    None
    """
    sp.set_type(cycle, sp.TNat)
    sp.verify (self.is_backend(sp.sender))
    sp.verify (self.data.processed_stake_index == 0, message="Last cycle reward is not completed")
    sp.verify (sp.balance >= self.data.distributed_reward, message="EmptyCycleReward")
    sp.verify (cycle > self.data.last_paid_cycle, message="CycleAlreadyRewarded")
    reward = sp.local("reward",  sp.balance - self.data.total_deposits - self.data.total_team_fee - self.data.total_tradein_fee)
    sp.for d in sp.range(self.data.last_paid_cycle, cycle, step = 1):
        sp.if self.data.cycles_deposits.contains(d + 1):
            self.data.full_active_deposits += self.data.cycles_deposits[d + 1]
    sp.if self.data.full_active_deposits > sp.mutez(0):
        full_active_deposits = sp.local('full_active_deposits', sp.fst(sp.ediv(self.data.full_active_deposits, sp.mutez(1)).open_some()))
        c = sp.fst(sp.ediv(sp.split_tokens(reward.value, 1000000, full_active_deposits.value), sp.mutez(1)).open_some()) # true_coef * 1_000_000
        self.data.current_coefficient = c
    sp.else:
        self.data.current_coefficient = 0
    self.data.distributed_reward = sp.balance
    self.data.last_paid_cycle = cycle
```

Figure 4 The origin source code of function *setup_reward*

- **Fix Result:** Fixed. The *distributed_reward* variable and related checks have been removed.

5.3 Incorrect calculation of reward (commit: 9d2b5c0)

- **Description:** If some stakers do not claim reward in a certain cycle, their *claimable_rewards* will be put into the reward in the next *setup_reward* and distributed to all stakers. Because this part of *claimable_rewards* is calculated repeatedly, finally all *claimable_rewards+deposit+trade_in+team_fee* exceeds *sp.balance*. The script *ForgottenClaimReward.py* is used to reproduce this problem. The modify recommendation is to add a variable to record the sum of all *claimable_rewards* and subtract this part from *setup_reward.*



```
@sp.entry_point
def setup_reward(self, cycle):
    """
    This function is called by backend before reward process
    (every 3 day this function is invoked by backed)
    Parameters
    ----------
    cycle : TNat
        Reward Cycle
    Returns
    ----------
    None
    """
    sp.set_type(cycle, sp.TNat)
    sp.verify (self.is_backend(sp.sender))
    sp.verify (self.data.processed_stake_index == 0, message="Last cycle reward is not completed")
    sp.verify (cycle > self.data.last_paid_cycle, message="CycleAlreadyRewarded")
    reward = sp.local("reward",  sp.balance - self.data.total_deposits - self.data.total_team_fee - self.data.total_tradein_fee)
    sp.for d in sp.range(self.data.last_paid_cycle, cycle, step = 1):
        sp.if self.data.cycles_deposits.contains(d + 1):
            self.data.full_active_deposits += self.data.cycles_deposits[d + 1]
    sp.if self.data.full_active_deposits > sp.mutez(0):
        full_active_deposits = sp.local('full_active_deposits', sp.fst(sp.ediv(self.data.full_active_deposits, sp.mutez(1)).open_some()))
        c = sp.fst(sp.ediv(sp.split_tokens(reward.value, 1000000, full_active_deposits.value), sp.mutez(1)).open_some()) # true_coef * 1_000_000
        self.data.current_coefficient = c
    sp.else:
        self.data.current_coefficient = 0
    self.data.last_paid_cycle = cycle
```

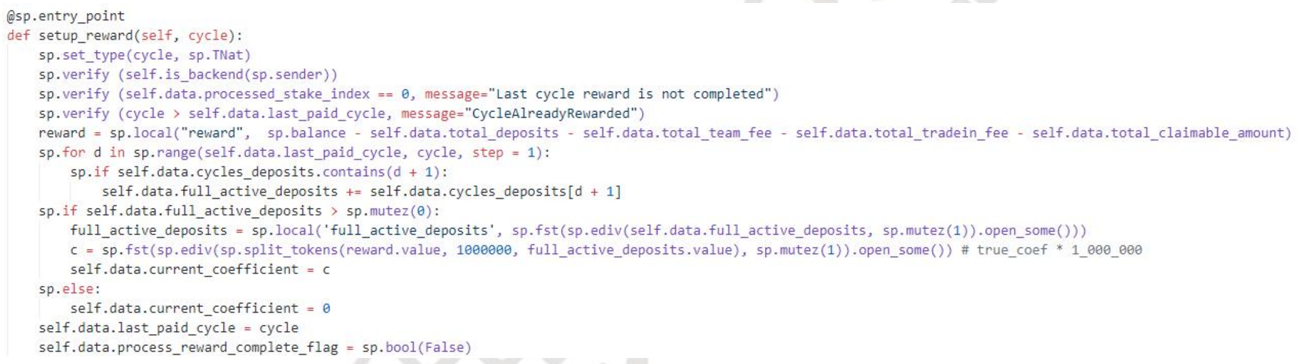
Figure 5 The origin source code of function *setup_reward*

The following is the source code of *ForgottenClaimReward.py*.

```
if "templates" not in __name__:
    @sp.add_test(name = "wXTZ_SM_DC")
    def test():

        scenario = sp.test_scenario()
        scenario.h1("wXTZ")

        scenario.table_of_contents()

        # sp.test_account generates ED25519 key-pairs deterministically:
        admin = sp.test_account("Administrator")
        alice = sp.test_account("Alice")
        team = sp.test_account("Team")
        tradein = sp.test_account("Tradein")
        backend = sp.test_account("Backend")
        node = sp.test_account("Node")

        # Let's display the accounts:
        scenario.h1("Accounts")
        # scenario.show([admin, alice, team, backend, node])

        scenario.h1("Contract")
        t = WTezosToken(admin.address)
```

```
sm = StakingManager(admin.address, team.address, tradein.address)
dc = DepositContract(admin.address)

scenario += t
scenario += sm
scenario += dc

scenario.h1("Entry points")
scenario.h2("Set parameters for deposit contract")
scenario += dc.set_token_address(t.address).run(sender = admin)
scenario += dc.set_staking_manager_address(sm.address).run(sender = admin)
scenario += dc.set_node_key_hash(sp.hash_key(node.public_key)).run(sender = admin)

scenario.h2("Set DepositContract as token contract operator")
scenario += t.set_operator(sp.some(dc.address)).run(sender = admin)

scenario.h2("Set DepositContract as staking manager contract set_owner")
scenario += sm.set_operator(sp.some(dc.address)).run(sender = admin)
scenario += sm.set_backend(sp.some(backend.address)).run(sender = admin)

scenario.h2("Alice deposit")
scenario += dc.deposit(
        user = alice.address,
        eth_address = "0x0000000000000000000000000000000000000001",
).run(sender = alice, amount = sp.tez(100))

scenario.verify_equal(dc.balance, sp.tez(0))
scenario.verify_equal(sm.balance, sp.tez(100))
scenario.verify(t.data.balances[alice.address] == 100*10**6)
scenario.verify(sm.data.stakes[alice.address].deposits[0]
                        == sp.record(start_cycle = 0,
                                     end_cycle = 0,
                                     withdrew = sp.bool(False),
                                     eth_address = '0x0000000000000000000000000000000000000001',
                                     amount = sp.tez(100),
                                     ))


scenario.h1("backend can process reward for users")
scenario.h2("Call process_reward for epoch")
# Just for set start_cycle
scenario += sm.setup_reward(113).run(sender = backend)
scenario += sm.process_reward(sp.record(max_count = 10)).run(sender = backend)

scenario.h2("Call process_reward for epoch #150")
scenario += sm.setup_reward(150).run(sender = backend, amount = sp.tez(60))
scenario += sm.process_reward(sp.record(max_count = 10)).run(sender = backend)
scenario.h2("Call process_reward for epoch #180")
scenario += sm.setup_reward(180).run(sender = backend, amount = sp.tez(60))
scenario += sm.process_reward(sp.record(max_count = 10)).run(sender = backend)

scenario.h2("Claim reward and withdraw")
scenario += dc.withdraw(sp.as_nat(0)).run(sender = alice)
scenario += dc.claim_reward(alice.address).run(sender = alice)
scenario += sm.withdraw_team_fee(sp.record(address = backend.address)).run(sender = backend)
scenario += sm.withdraw_tradein_fee(sp.record(address = backend.address)).run(sender = backend)
```

- **Fix Result:** Fixed. The final code is shown below.



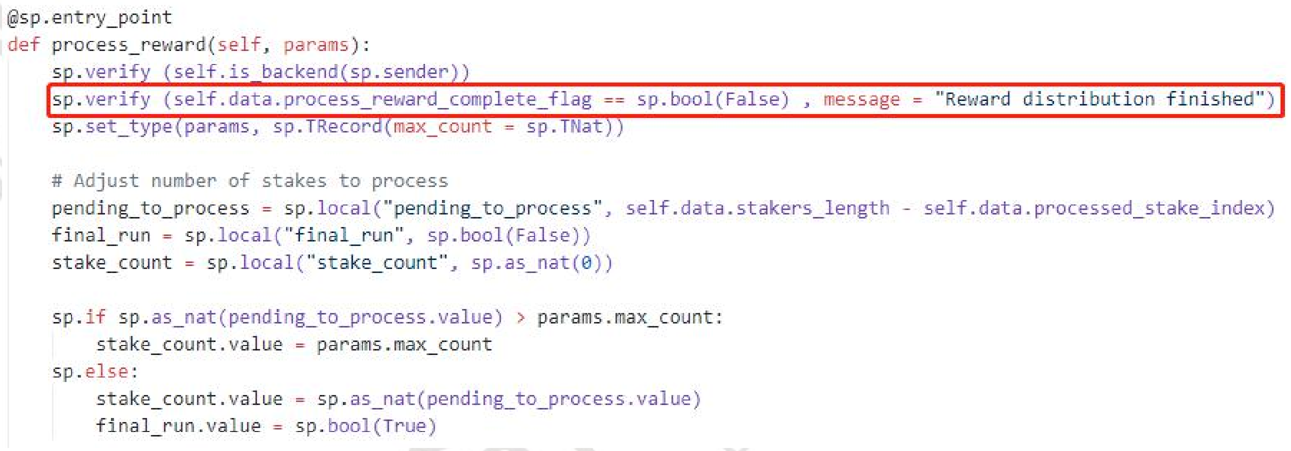
```
@sp.entry_point
def setup_reward(self, cycle):
    sp.set_type(cycle, sp.TNat)
    sp.verify (self.is_backend(sp.sender))
    sp.verify (self.data.processed_stake_index == 0, message="Last cycle reward is not completed")
    sp.verify (cycle > self.data.last_paid_cycle, message="CycleAlreadyRewarded")
    reward = sp.local("reward",  sp.balance - self.data.total_deposits - self.data.total_team_fee - self.data.total_tradein_fee - self.data.total_claimable_amount)
    sp.for d in sp.range(self.data.last_paid_cycle, cycle, step = 1):
        sp.if self.data.cycles_deposits.contains(d + 1):
            self.data.full_active_deposits += self.data.cycles_deposits[d + 1]
    sp.if self.data.full_active_deposits > sp.mutez(0):
        full_active_deposits = sp.local('full_active_deposits', sp.fst(sp.ediv(self.data.full_active_deposits, sp.mutez(1)).open_some()))
        c = sp.fst(sp.ediv(sp.split_tokens(reward.value, 1000000, full_active_deposits.value), sp.mutez(1)).open_some()) # true_coef * 1_000_000
        self.data.current_coefficient = c
    sp.else:
        self.data.current_coefficient = 0
    self.data.last_paid_cycle = cycle
    self.data.process_reward_complete_flag = sp.bool(False)
```

Figure 6 The final source code of function *setup_reward*

5.4 Multi-time *process_reward* (commit: 9d2b5c0)

- **Description:** If backend address calls *process_reward* multiple times after *setup_reward*, and the parameter *max_count* of *process_reward* exceeds *stakers_length*, it will cause *claimable_rewards* of stakers to be calculated multiple times.

- **Fix Result:** Fixed. The final code is shown below.



```
@sp.entry_point
def process_reward(self, params):
    sp.verify (self.is_backend(sp.sender))
    sp.verify (self.data.process_reward_complete_flag == sp.bool(False) , message = "Reward distribution finished")
    sp.set_type(params, sp.TRecord(max_count = sp.TNat))

    # Adjust number of stakes to process
    pending_to_process = sp.local("pending_to_process", self.data.stakers_length - self.data.processed_stake_index)
    final_run = sp.local("final_run", sp.bool(False))
    stake_count = sp.local("stake_count", sp.as_nat(0))

    sp.if sp.as_nat(pending_to_process.value) > params.max_count:
        stake_count.value = params.max_count
    sp.else:
        stake_count.value = sp.as_nat(pending_to_process.value)
        final_run.value = sp.bool(True)
```

Figure 7 The final source code of function *process_reward*

5.5 Repeat deposit attack

- **Description:** The user deposits 1000 XTZ when the cycle is 0, and then when the cycle is 1, after the backend executes the *process_reward*, his *start_cycle* is set to 14, then withdraw 1000 XTZ, and then immediately deposit 1000 XTZ again, after the next round of *process_reward*, The *start_cycle* of the new deposit is 15. At this time, in the range of 15-21 cycle, he is equivalent to deposit 2000 XTZ and obtain corresponding dividends. Of course, he can do this all the time, and use this 1000 XTZ at most (cycle >= 20), he can obtain dividends equivalent to 7000 XTZ at the same time.

- **Fix Result: Fixed.** The user's deposit will be put into the corresponding delegate contract, and the corresponding XTZ rewards will also be sent to the respective delegate contract first, and the deposit of each user will not affect each other.

5.6 Repeat deposit attack v2

- **Description:** Furthermore, even users don't need to wait to confirm until the next cycle, they can deposit & withdraw repeatedly in the same cycle. It is recommended to check them, if *start_cycle* is 0 when withdrawing, delete this stake record directly.

- **Fix Result: Fixed**.

6 Conclusion

Beosin (Chengdu LianAn) conducted a detailed audit on the design and code implementation of the RAMP_VERSION_TEZOS project. All the problems found in the audit process were notified to the project party, and got quick feedback and repair from the project party. Beosin (Chengdu LianAn) confirms that all the problems found have been properly fixed or have reached an agreement with the project party on how to deal with it. The overall result of this RAMP_VERSION_TEZOS audit is Pass.

BEOSIN

Blockchain Security

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com