BEOSIN
Blockchain Security

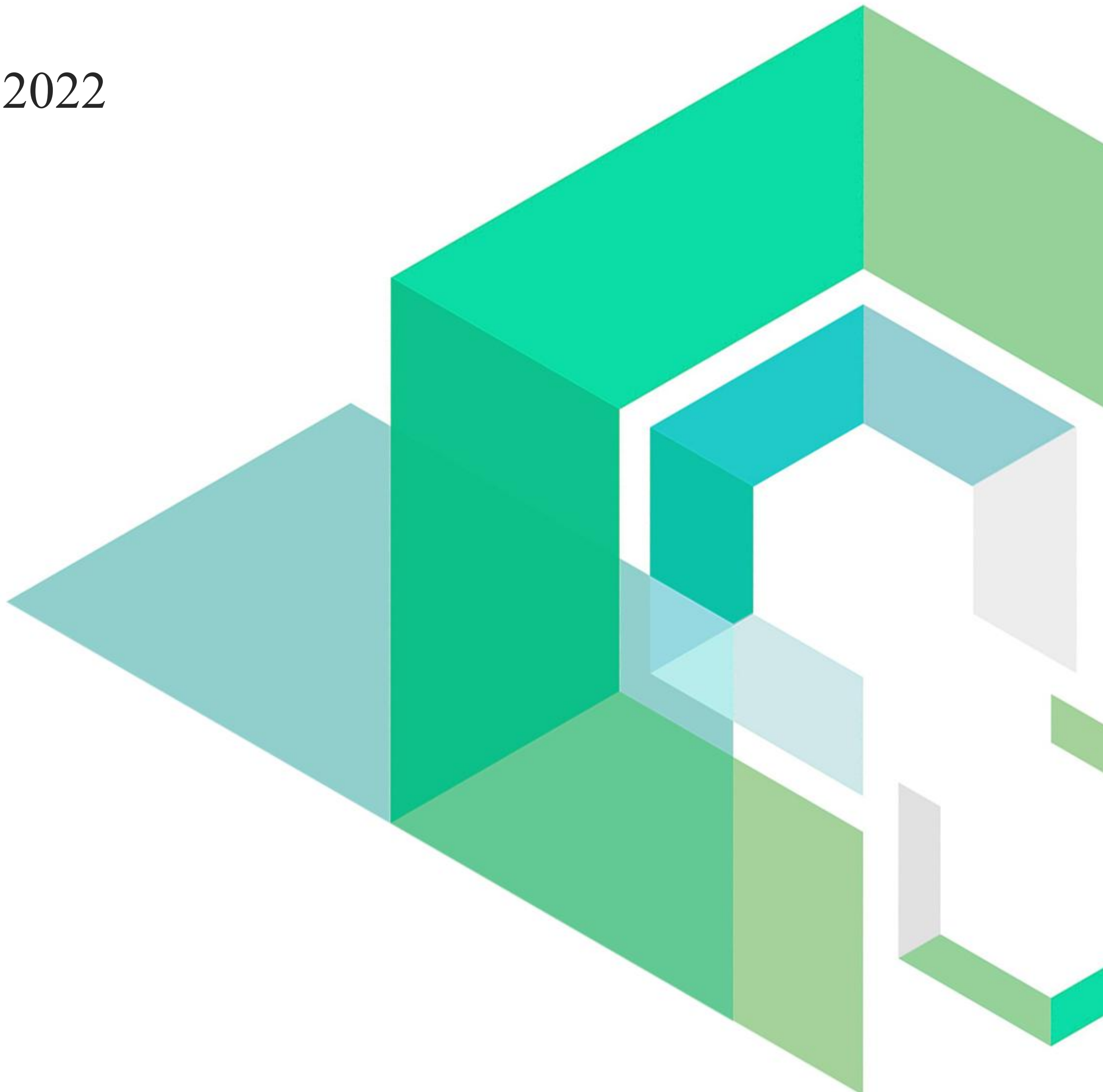
Versailles-heroes

Smart Contract Security Audit

V1.1

No. 202206131514

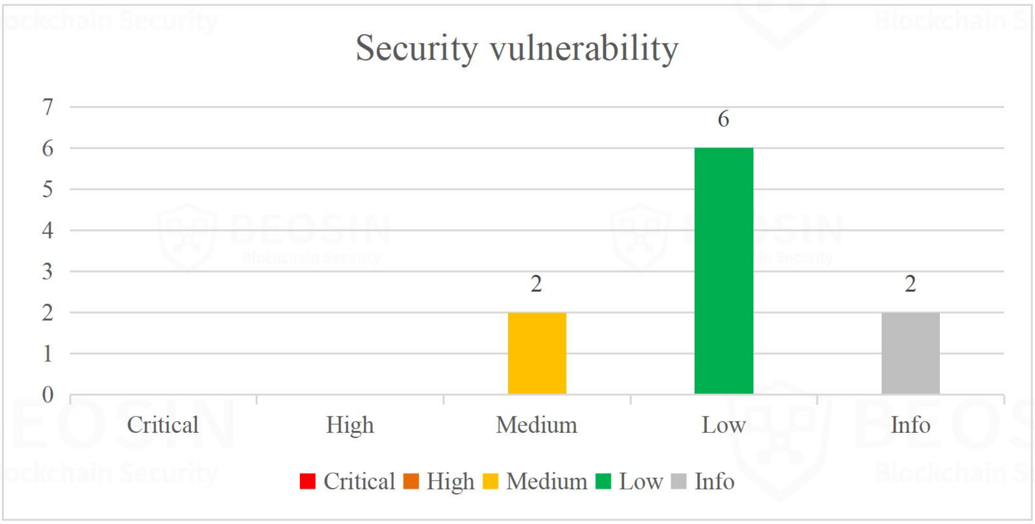
Jun 13th, 2022



Contents

Summary of audit results

After auditing, 2 Medium-risks, 6 Low-risks and 2 Info items were identified in the Versailles-heroes project. Specific audit details will be presented in the **Findings section**. Users should pay attention to the following aspects when interacting with this project:



Notes:

- **Risk Description:**

1. **Tokens required to create a guild are higher than expected**

When a user creates a guild, a stake of 100,000 VRH for 4 years or 400,000 VRH for one year cannot meet the minimum requirements for creating a guild. Users need to stake more VRH to do so.

2. **Token minimum lock time is lower than expected**

In the VotingEscrow contract, a WEEK is added when judging whether the minimum lock time is reached, so that the minimum VRH lock time can be less than 365 days. The project team replied that this is for front-end considerations.

3. **The owner's data is not updated when creating a guild**

When the administrator address in the GuildController contract calls *create_guild* function to create a guild, the relevant data of the owner is not updated. If the guild owner address forgets to update its own data, it may cause the guild's overall data to be abnormal.

- **Project Description:**

1. Basic Token Information

Token name	set when deploying
Token symbol	set when deploying
Decimals	set when deploying
Pre-mint	727.2 million
Total supply	Initial supply is 727.2 million (Mintable, burnable)
Token type	ERC20

Table 1 ERC20VRH Token Info

2. Business overview

The project mainly implements a blockchain game. Users gain veVRH tokens by locking VRH tokens (The minimum lock-up period is one year, and the maximum lock-up period is 4 years). After that, they can create or join guilds (After joining a guild, it takes a certain amount of time to exit) where VRH rewards will be generated, 30% of the rewards will be acquired immediately and the remaining 70% will be unlocked over time. And the rewards obtained can be increased by burning the GAS tokens (The operation is irreversible). The reward rate and GAS are not necessarily the same for different guilds.

1 Overview

1.1 Project Overview

Project Name	Versailles-heroes
Platform	ETH
Audit scope	https://github.com/Versailles-heroes-com/versailles-heroes-DAO
Commit Hash	d1b680295a6b3f41bd82056c68d7bd51cd2369b9

1.2 Audit Overview

Audit work duration：May 07, 2022 – June 13, 2022

Update Details: July 4, 2022. Update code.

Audit methods: Formal Verification, Static Analysis, Typical Case Testing and Manual Review.

Audit team: Beosin Technology Co. Ltd.

2 Findings

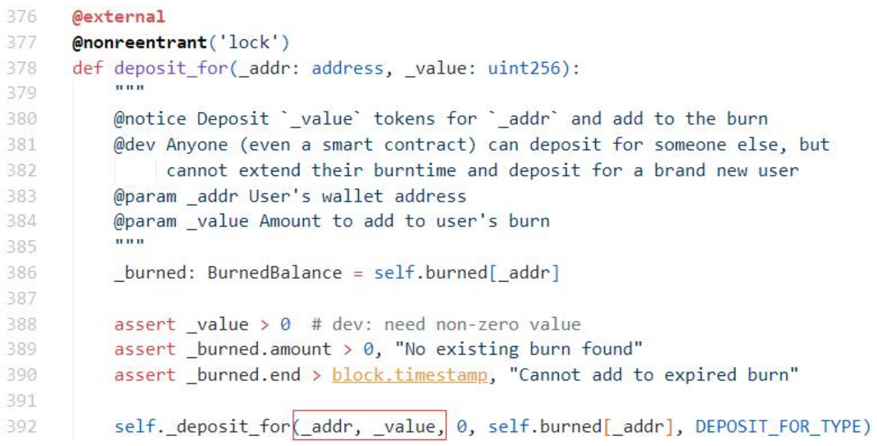
Index	Risk description	Severity level	Status
VH-1	*deposit_for* function without permission check	Medium	Fixed
VH-2	The amount of veVRH obtained by locking VRH is not as expected	Medium	Acknowledged
VH-3	Missing address check in *deposit_for* function	Low	Fixed
VH-4	No time limit for the initial owner of the guild to exit	Low	Fixed
VH-5	Guild rate modification limit error	Low	Fixed
VH-6	Incorrect minimum lock time judgment	Low	Acknowledged
VH-7	Risk of accidental token lockup	Low	Fixed
VH-8	The owner's data is not updated when creating a guild	Low	Acknowledged
VH-9	Abnormal increase in period	Info	Acknowledged
VH-10	*belongs_to_guild* function lacks view modifier	Info	Fixed

Risk Details Description：

1. VH-2 is not fixed and may cause the users have to stake more VRH to create a guild.
2. VH-6 is not fixed and may cause the user lockout time to be less than 365 days.
3. VH-8 is not fixed and may cause the guild data in the contract to be abnormal (if the guild owner does not manually update their own data).
4. VH-9 is not fixed and will not cause any security issue.

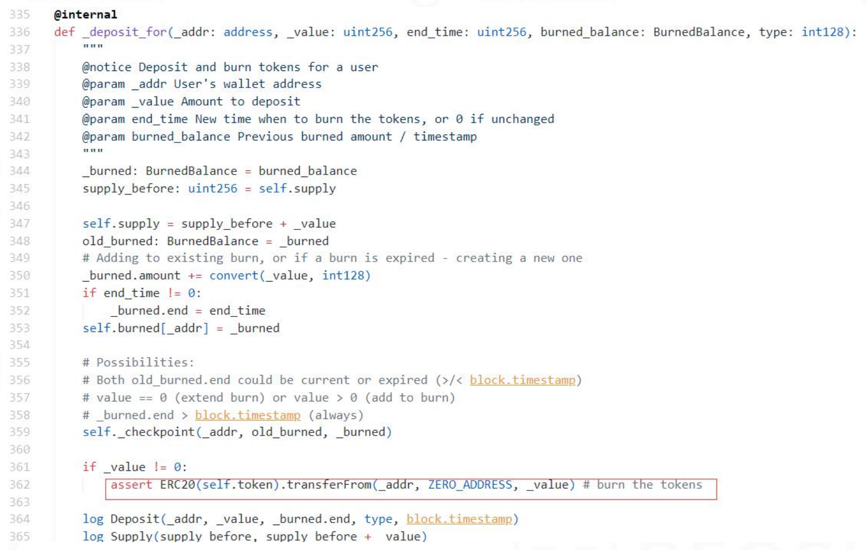
[VH-1] *deposit_for* function without permission check

Severity Level	Medium
Type	Business Security
Lines	GasEscrow.vy#L335-365, 376-392
Description	Any address can call the deposit_for function to maliciously stake the specific tokens of users who have excess authorization value in the contract into the contract, and the operation cannot be undone.



```
376 @external
377 @nonreentrant('lock')
378 def deposit_for(_addr: address, _value: uint256):
379     """
380     @notice Deposit `_value` tokens for `_addr` and add to the burn
381     @dev Anyone (even a smart contract) can deposit for someone else, but
382          cannot extend their burntime and deposit for a brand new user
383     @param _addr User's wallet address
384     @param _value Amount to add to user's burn
385     """
386     _burned: BurnedBalance = self.burned[_addr]
387 
388     assert _value > 0  # dev: need non-zero value
389     assert _burned.amount > 0, "No existing burn found"
390     assert _burned.end > block.timestamp, "Cannot add to expired burn"
391 
392     self._deposit_for(_addr, _value, 0, self.burned[_addr], DEPOSIT_FOR_TYPE)
```

Figure 1 Source code of *deposit_for* function



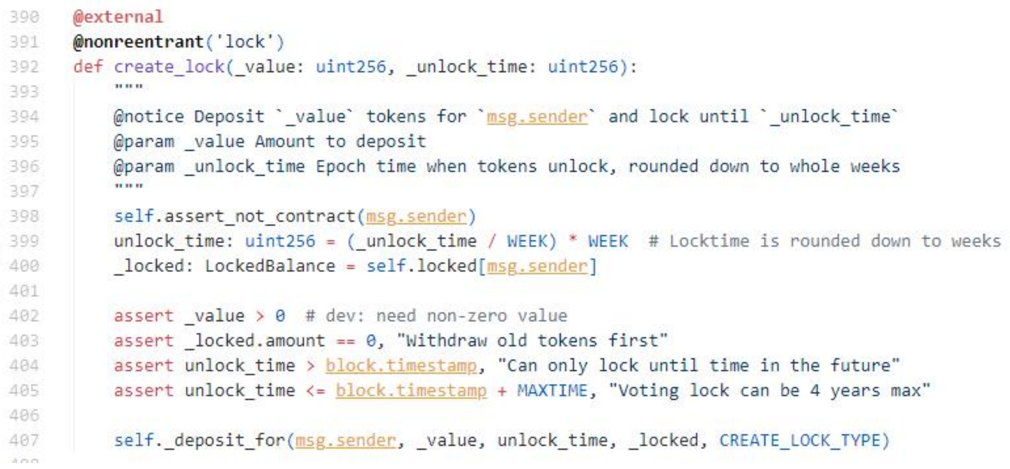
```
335 @internal
336 def _deposit_for(_addr: address, _value: uint256, end_time: uint256, burned_balance: BurnedBalance, type: int128):
337     """
338     @notice Deposit and burn tokens for a user
339     @param _addr User's wallet address
340     @param _value Amount to deposit
341     @param end_time New time when to burn the tokens, or 0 if unchanged
342     @param burned_balance Previous burned amount / timestamp
343     """
344     _burned: BurnedBalance = burned_balance
345     supply_before: uint256 = self.supply
346 
347     self.supply = supply_before + _value
348     old_burned: BurnedBalance = _burned
349     # Adding to existing burn, or if a burn is expired - creating a new one
350     _burned.amount += convert(_value, int128)
351     if end_time != 0:
352         _burned.end = end_time
353     self.burned[_addr] = _burned
354 
355     # Possibilities:
356     # Both old_burned.end could be current or expired (>/< block.timestamp)
357     # value == 0 (extend burn) or value > 0 (add to burn)
358     # _burned.end > block.timestamp (always)
359     self._checkpoint(_addr, old_burned, _burned)
360 
361     if _value != 0:
362         assert ERC20(self.token).transferFrom(_addr, ZERO_ADDRESS, _value) # burn the tokens
363 
364     log Deposit(_addr, _value, _burned.end, type, block.timestamp)
365     log Supply(supply_before, supply_before + _value)
```

Figure 2 Source code of *_deposit_for* function

Recommendations	It is recommended to remove the *deposit_for* function or add a permission check.
Status	Fixed. This function has been removed.

[VH-2] The amount of veVRH obtained by locking VRH is not as expected

Severity Level	Medium
Type	Business Security
Lines	VotingEscrow.vy#L390-407
Description	In the VotingEscrow contract, user cannot get 100,000 veVRH by locking 100,000 VRH for four years or 400,000 VRH for one year. This is inconsistent with the description in the white paper.

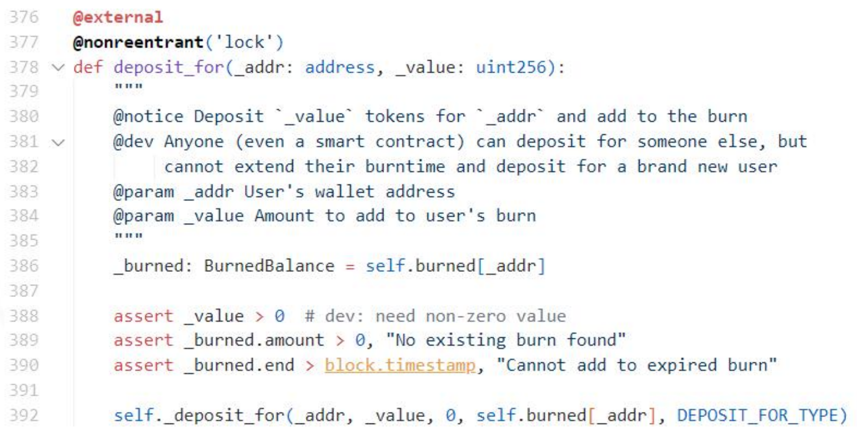


```
@external
@nonreentrant('lock')
def create_lock(_value: uint256, _unlock_time: uint256):
    """
    @notice Deposit `_value` tokens for `msg.sender` and lock until `_unlock_time`
    @param _value Amount to deposit
    @param _unlock_time Epoch time when tokens unlock, rounded down to whole weeks
    """
    self.assert_not_contract(msg.sender)
    unlock_time: uint256 = (_unlock_time / WEEK) * WEEK  # Locktime is rounded down to weeks
    _locked: LockedBalance = self.locked[msg.sender]

    assert _value > 0  # dev: need non-zero value
    assert _locked.amount == 0, "Withdraw old tokens first"
    assert unlock_time > block.timestamp, "Can only lock until time in the future"
    assert unlock_time <= block.timestamp + MAXTIME, "Voting lock can be 4 years max"

    self._deposit_for(msg.sender, _value, unlock_time, _locked, CREATE_LOCK_TYPE)
```

Figure 3 Source code of *create_lock* function

Recommendations	It is recommended to allow a certain error when judging the conditions for creating a guild.
Status	Acknowledged. The project team has changed the description in the white paper and recommends that users stake more tokens to meet the requirements.

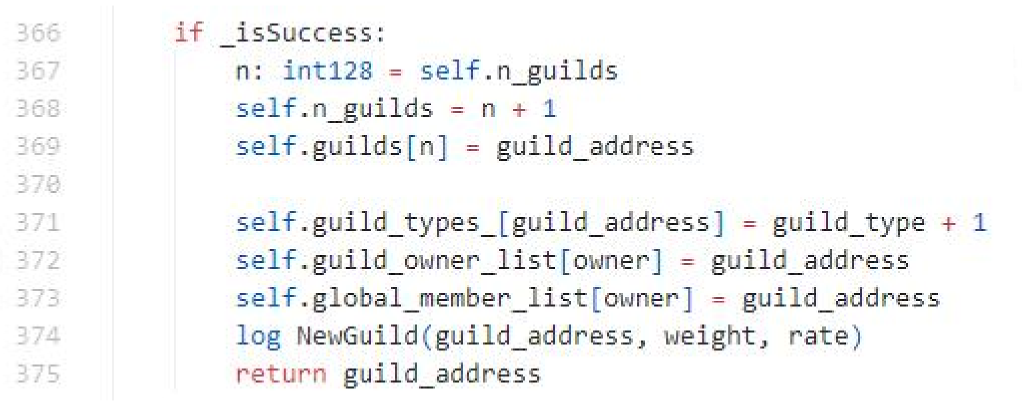
[VH-3] Missing address check in *deposit_for* function

Severity Level	Low
Type	Business Security
Lines	GasEscrow.vy#L376-392
Description	The *deposit_for* function in the GasEscrow contract does not check whether the _addr address is the contract address.  Figure 4 Source code of *deposit_for* function
Recommendations	It is recommended to add contract address judgment to the *deposit_for* function.
Status	Fixed. This function has been removed.

```
@external
@nonreentrant('lock')
def deposit_for(_addr: address, _value: uint256):
    """
    @notice Deposit `_value` tokens for `_addr` and add to the burn
    @dev Anyone (even a smart contract) can deposit for someone else, but
         cannot extend their burntime and deposit for a brand new user
    @param _addr User's wallet address
    @param _value Amount to add to user's burn
    """
    _burned: BurnedBalance = self.burned[_addr]

    assert _value > 0  # dev: need non-zero value
    assert _burned.amount > 0, "No existing burn found"
    assert _burned.end > block.timestamp, "Cannot add to expired burn"

    self._deposit_for(_addr, _value, 0, self.burned[_addr], DEPOSIT_FOR_TYPE)
```

[VH-4] No time limit for the initial owner of the guild to exit

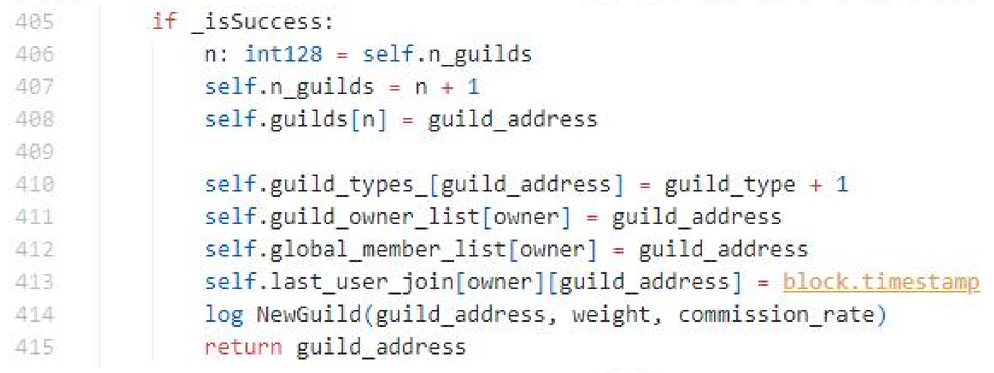
Severity Level	Low
Type	Business Security
Lines	GuildController.vy#L366-375
Description	In the GuildController contract, the initial owner of the guild can immediately quit the guild after transferring the owner permission to others, and will not quit the guild after joining the guild like other users after WEIGHT_VOTE_DELAY.



```
if _isSuccess:
    n: int128 = self.n_guilds
    self.n_guilds = n + 1
    self.guilds[n] = guild_address

    self.guild_types_[guild_address] = guild_type + 1
    self.guild_owner_list[owner] = guild_address
    self.global_member_list[owner] = guild_address
    log NewGuild(guild_address, weight, rate)
    return guild_address
```

Figure 5 Source code of *create_guild* function (Unfixed)

Recommendations	It is recommended to set the current time as the initial owner joining time of the guild when creating a guild.
Status	Fixed.



```
if _isSuccess:
    n: int128 = self.n_guilds
    self.n_guilds = n + 1
    self.guilds[n] = guild_address

    self.guild_types_[guild_address] = guild_type + 1
    self.guild_owner_list[owner] = guild_address
    self.global_member_list[owner] = guild_address
    self.last_user_join[owner][guild_address] = block.timestamp
    log NewGuild(guild_address, weight, commission_rate)
    return guild_address
```

Figure 6 Source code of *create_guild* function (Fixed)

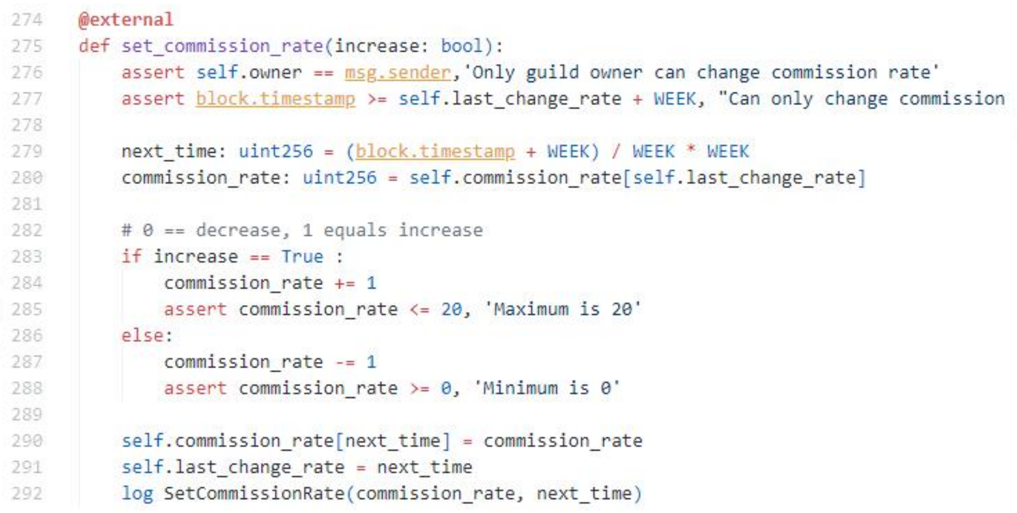
[VH-5] Guild rate modification limit error

Severity Level Low

Type Business Security

Lines Guild.vy#L274-292

Description According to the white paper, the rate of the guild in the Guild contract can be modified once a week, but the current code seems to be modified once every 2 weeks.

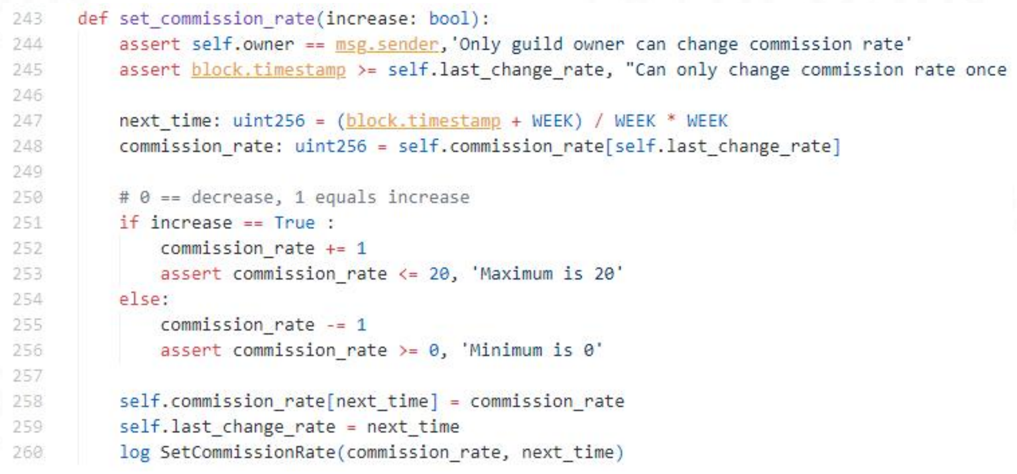


```
274 @external
275 def set_commission_rate(increase: bool):
276     assert self.owner == msg.sender,'Only guild owner can change commission rate'
277     assert block.timestamp >= self.last_change_rate + WEEK, "Can only change commission
278 
279     next_time: uint256 = (block.timestamp + WEEK) / WEEK * WEEK
280     commission_rate: uint256 = self.commission_rate[self.last_change_rate]
281 
282     # 0 == decrease, 1 equals increase
283     if increase == True :
284         commission_rate += 1
285         assert commission_rate <= 20, 'Maximum is 20'
286     else:
287         commission_rate -= 1
288         assert commission_rate >= 0, 'Minimum is 0'
289 
290     self.commission_rate[next_time] = commission_rate
291     self.last_change_rate = next_time
292     log SetCommissionRate(commission_rate, next_time)
```

Figure 7 Source code of *set_commission_rate* function (Unfixed)

Recommendations It is recommended not to add WEEK in the judgment.

Status Fixed.

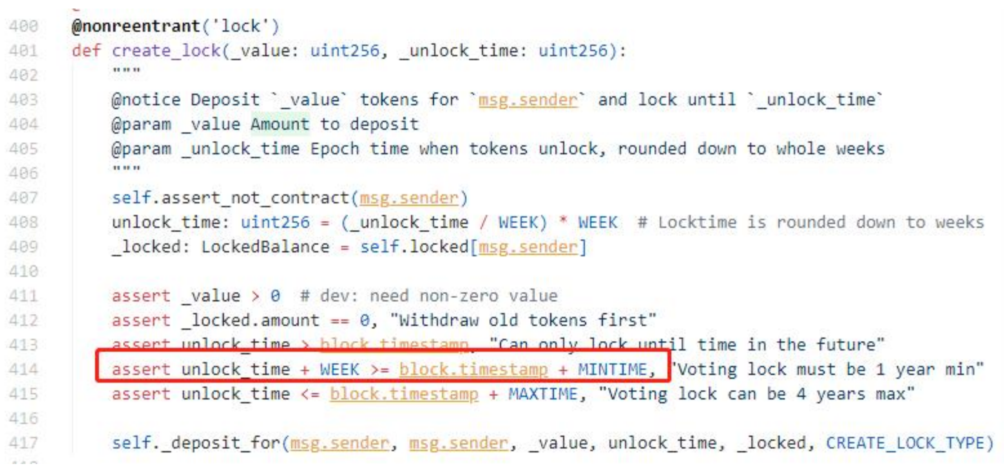


```
def set_commission_rate(increase: bool):
    assert self.owner == msg.sender,'Only guild owner can change commission rate'
    assert block.timestamp >= self.last_change_rate, "Can only change commission rate once

    next_time: uint256 = (block.timestamp + WEEK) / WEEK * WEEK
    commission_rate: uint256 = self.commission_rate[self.last_change_rate]

    # 0 == decrease, 1 equals increase
    if increase == True :
        commission_rate += 1
        assert commission_rate <= 20, 'Maximum is 20'
    else:
        commission_rate -= 1
        assert commission_rate >= 0, 'Minimum is 0'

    self.commission_rate[next_time] = commission_rate
    self.last_change_rate = next_time
    log SetCommissionRate(commission_rate, next_time)
```

Figure 8 Source code of *set_commission_rate* function (Fixed)

[VH-6] Incorrect minimum lock time judgment

Severity Level	Low
Type	Business Security
Lines	VotingEscrow.vy#L401-417
Description	In the VotingEscrow contract, a WEEK is added when judging whether the minimum lock time is reached, so that the minimum VRH lock time can be less than 365 days.



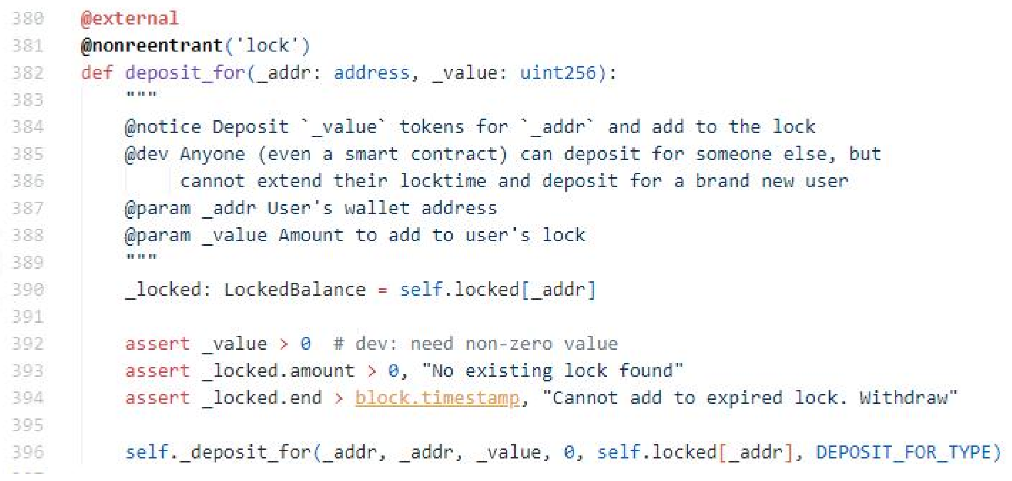
```
400  @nonreentrant('lock')
401  def create_lock(_value: uint256, _unlock_time: uint256):
402      """
403      @notice Deposit `_value` tokens for `msg.sender` and lock until `_unlock_time`
404      @param _value Amount to deposit
405      @param _unlock_time Epoch time when tokens unlock, rounded down to whole weeks
406      """
407      self.assert_not_contract(msg.sender)
408      unlock_time: uint256 = (_unlock_time / WEEK) * WEEK  # Locktime is rounded down to weeks
409      _locked: LockedBalance = self.locked[msg.sender]
410
411      assert _value > 0  # dev: need non-zero value
412      assert _locked.amount == 0, "Withdraw old tokens first"
413      assert unlock_time > block.timestamp, "Can only lock until time in the future"
414      assert unlock_time + WEEK >= block.timestamp + MINTIME, "Voting lock must be 1 year min"
415      assert unlock_time <= block.timestamp + MAXTIME, "Voting lock can be 4 years max"
416
417      self._deposit_for(msg.sender, msg.sender, _value, unlock_time, _locked, CREATE_LOCK_TYPE)
```

Figure 9 Source code of *create_lock* function (Unfixed)

Recommendations	If the return value is not needed, it is recommended to eliminate the return of the variable.
Status	Acknowledged. The project team confirms that it meets the design requirements.

[VH-7] Risk of accidental token lockup

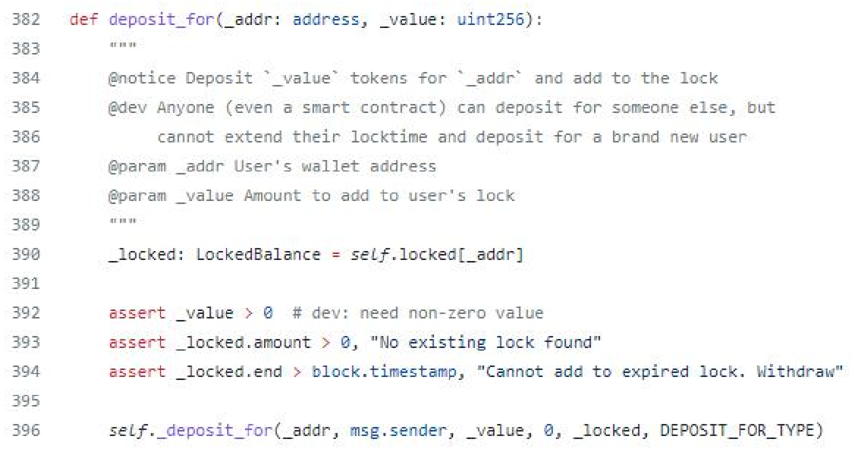
Severity Level	Low
Type	Business Security
Lines	VotingEscrow.vy#L380-396
Description	Any address can call the *deposit_for* function in the VotingEscrow contract to transfer the tokens of users who have authorized values to the contract to the contract and lock them.



```
@external
@nonreentrant('lock')
def deposit_for(_addr: address, _value: uint256):
    """
    @notice Deposit `_value` tokens for `_addr` and add to the lock
    @dev Anyone (even a smart contract) can deposit for someone else, but
         cannot extend their locktime and deposit for a brand new user
    @param _addr User's wallet address
    @param _value Amount to add to user's lock
    """
    _locked: LockedBalance = self.locked[_addr]

    assert _value > 0  # dev: need non-zero value
    assert _locked.amount > 0, "No existing lock found"
    assert _locked.end > block.timestamp, "Cannot add to expired lock. Withdraw"

    self._deposit_for(_addr, _addr, _value, 0, self.locked[_addr], DEPOSIT_FOR_TYPE)
```

Figure 10 Source code of *deposit_for* function (Unfixed)

Recommendations	It is recommended to delete the *deposit_for* function or set the token source address to msg.sender.
Status	Fixed.

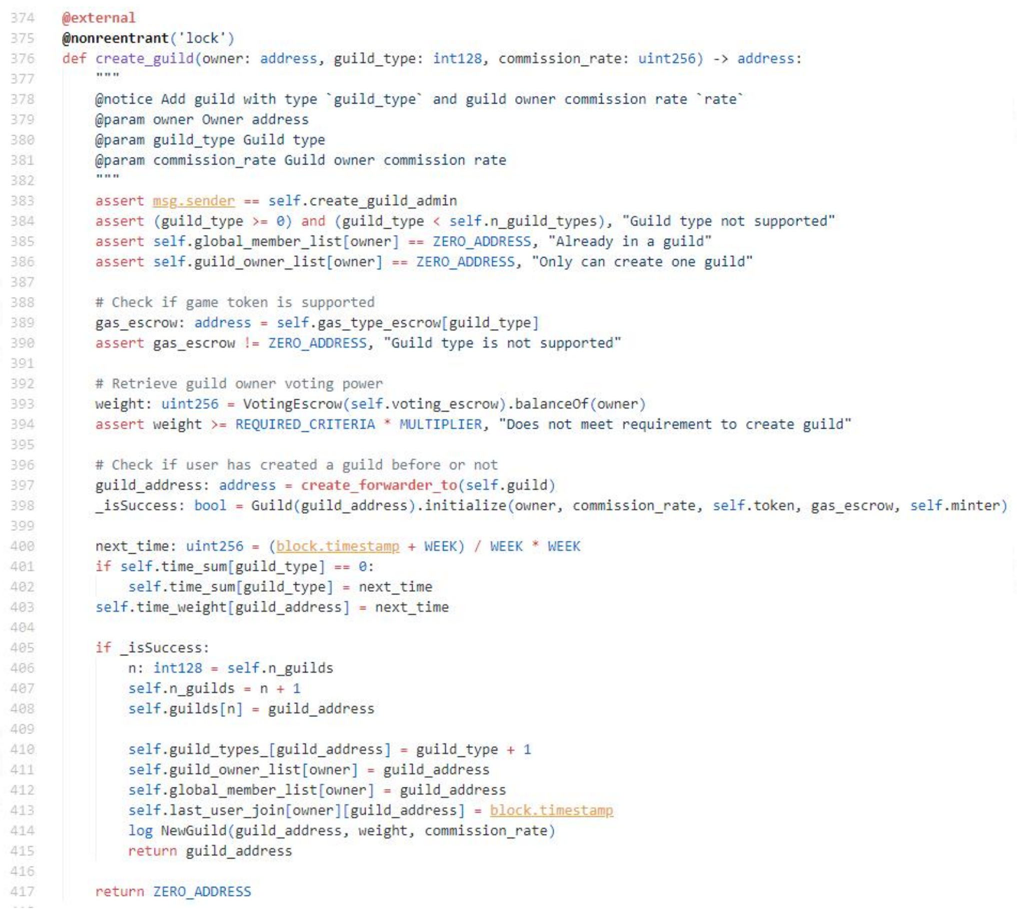


```
def deposit_for(_addr: address, _value: uint256):
    """
    @notice Deposit `_value` tokens for `_addr` and add to the lock
    @dev Anyone (even a smart contract) can deposit for someone else, but
         cannot extend their locktime and deposit for a brand new user
    @param _addr User's wallet address
    @param _value Amount to add to user's lock
    """
    _locked: LockedBalance = self.locked[_addr]

    assert _value > 0  # dev: need non-zero value
    assert _locked.amount > 0, "No existing lock found"
    assert _locked.end > block.timestamp, "Cannot add to expired lock. Withdraw"

    self._deposit_for(_addr, msg.sender, _value, 0, _locked, DEPOSIT_FOR_TYPE)
```

Figure 11 Source code of *deposit_for* function (Fixed)

[VH-8] The owner's data is not updated when creating a guild

Severity Level	Low
Type	Business Security
Lines	VotingEscrow.vy#L380-396
Description	When the administrator address in the GuildController contract calls *create_guild* to create a guild, the relevant data of the owner is not updated.



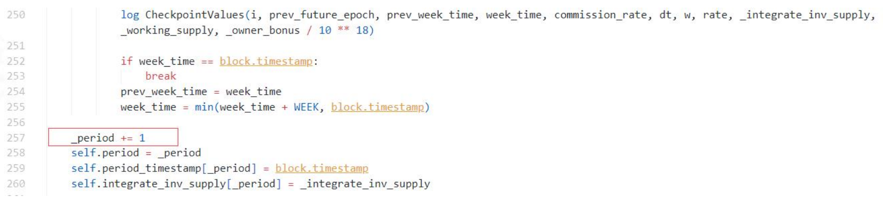
```
374  @external
375  @nonreentrant('lock')
376  def create_guild(owner: address, guild_type: int128, commission_rate: uint256) -> address:
377      """
378      @notice Add guild with type `guild_type` and guild owner commission rate `rate`
379      @param owner Owner address
380      @param guild_type Guild type
381      @param commission_rate Guild owner commission rate
382      """
383      assert msg.sender == self.create_guild_admin
384      assert (guild_type >= 0) and (guild_type < self.n_guild_types), "Guild type not supported"
385      assert self.global_member_list[owner] == ZERO_ADDRESS, "Already in a guild"
386      assert self.guild_owner_list[owner] == ZERO_ADDRESS, "Only can create one guild"
387
388      # Check if game token is supported
389      gas_escrow: address = self.gas_type_escrow[guild_type]
390      assert gas_escrow != ZERO_ADDRESS, "Guild type is not supported"
391
392      # Retrieve guild owner voting power
393      weight: uint256 = VotingEscrow(self.voting_escrow).balanceOf(owner)
394      assert weight >= REQUIRED_CRITERIA * MULTIPLIER, "Does not meet requirement to create guild"
395
396      # Check if user has created a guild before or not
397      guild_address: address = create_forwarder_to(self.guild)
398      _isSuccess: bool = Guild(guild_address).initialize(owner, commission_rate, self.token, gas_escrow, self.minter)
399
400      next_time: uint256 = (block.timestamp + WEEK) / WEEK * WEEK
401      if self.time_sum[guild_type] == 0:
402          self.time_sum[guild_type] = next_time
403      self.time_weight[guild_address] = next_time
404
405      if _isSuccess:
406          n: int128 = self.n_guilds
407          self.n_guilds = n + 1
408          self.guilds[n] = guild_address
409
410          self.guild_types_[guild_address] = guild_type + 1
411          self.guild_owner_list[owner] = guild_address
412          self.global_member_list[owner] = guild_address
413          self.last_user_join[owner][guild_address] = block.timestamp
414          log NewGuild(guild_address, weight, commission_rate)
415          return guild_address
416
417      return ZERO_ADDRESS
```

Figure 12 Source code of *create_guild* function

Recommendations	It is recommended to update owner-related data when creating a guild.
Status	Acknowledged. The project team confirms that it meets the design requirements.

[VH-9] Abnormal increase in period

Severity Level	Info
Type	Business Security
Lines	Guild.vy#L257-260
Description	The _period in *_checkpoint* function is increasing each time it is called, which may result in multiple periods corresponding to the same timestamp in the period_timestamp.



```
250             log CheckpointValues(i, prev_future_epoch, prev_week_time, week_time, commission_rate, dt, w, rate, _integrate_inv_supply,
                _working_supply, _owner_bonus / 10 ** 18)
251
252             if week_time == block.timestamp:
253                 break
254             prev_week_time = week_time
255             week_time = min(week_time + WEEK, block.timestamp)
256
257         _period += 1
258         self.period = _period
259         self.period_timestamp[_period] = block.timestamp
260         self.integrate_inv_supply[_period] = _integrate_inv_supply
```

Figure 13 Source code of *_checkpoint* function

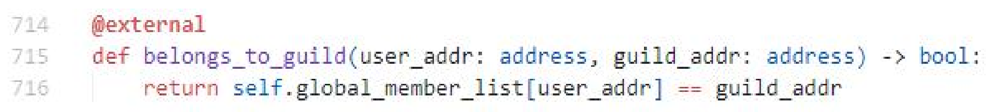
Recommendations	It is recommended to update period when the data has changed.
Status	Acknowledged. The project team confirms that it meets the design requirements.

[VH-10] *belongs_to_guild* function lacks view modifier

Severity Level	Info
Type	Coding Conventions
Lines	GuildController.vy#L714-716

Description

The *belongs_to_guild* function in the GuildController contract can add view modifiers to save gas consumption.



```
714  @external
715  def belongs_to_guild(user_addr: address, guild_addr: address) -> bool:
716      return self.global_member_list[user_addr] == guild_addr
```

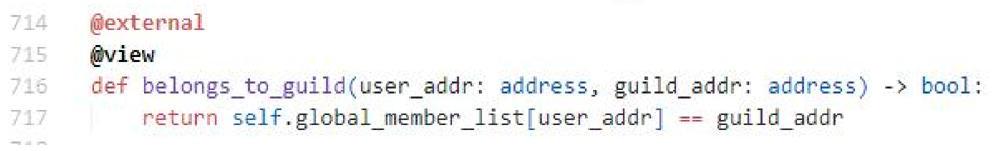
Figure 14 Source code of *belongs_to_guild* function (Unfixed)

Recommendations

It is recommended to add the view modifier to the *belongs_to_guild* function.

Status

Fixed.



```
714  @external
715  @view
716  def belongs_to_guild(user_addr: address, guild_addr: address) -> bool:
717      return self.global_member_list[user_addr] == guild_addr
```

Figure 15 Source code of *belongs_to_guild* function (Fixed)

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1 (Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Impact / Likelihood	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	High	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.4 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.5 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Compiler Version Security
		Deprecated Items
		Redundant Code
		require/assert Usage
		Gas Consumption
2	General Vulnerability	Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		call/delegatecall Security
		Returned Value Security
		tx.origin Usage
		Replay Attack
		Overriding Variables
		Third-party protocol interface consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable token price
		Centralized asset control
		Asset tradability
		Arbitrage attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in Blockchain.

3.4 About BEOSIN

Affiliated to BEOSIN Technology Pte. Ltd., BEOSIN is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions.BEOSIN has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, BEOSIN has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Blockchain Security

Official Website

https://www.beosin.com

Telegram

https://t.me/+dD8Bnqd133RmNWNl

Twitter

https://twitter.com/Beosin_com

Email

Contact@beosin.com