December 2020

Security Audit

APEMaster

TSMA1AZJMcxR7DzYTP8A4Wqzf3V7CxEd29

www.grox.solutions

CRITICAL ISSUES (critical, high severity): **0**

Bugs and vulnerabilities that enable theft of funds, lock access to funds without possibility to restore it, or lead to any other loss of funds to be transferred to any party; high priority unacceptable bugs for deployment at mainnet; critical warnings for owners, customers or investors.

ERRORS, BUGS AND WARNINGS (medium, low severity): **4**

Bugs that can trigger a contract failure, with further recovery only possible through manual modification of the contract state or contract replacement altogether; Lack of necessary security precautions; other warnings for owners and users.

OPTIMIZATION POSSIBILITIES (very low severity): **3**

Possibilities to decrease cost of transactions and data storage of Smart-Contracts.

NOTES AND RECOMMENDATIONS (very low severity): **3**

Tips and tricks, all other issues and recommendations, as well as errors that do not affect the functionality of the Smart-Contract.

Conclusion:

In the APEMaster Smart-Contract were found no vulnerabilities and no backdoors, but were discovered some errors. The code was manually reviewed for all commonly known and more specific vulnerabilities.

Note: there was no pre-minting of this token and any additional minting of tokens (other than from APEMaster contract) is impossible.

So APEMaster Smart-Contract is safe for use in the main network.

Note: this audit concerns only APEMaster smart-contract, not other parts of APE project.

ERRORS, BUGS AND WARNINGS

1. Event wrong data (medium severity):

Event '`referralReward`' contains 3 parameters: referrer address, user address and amount of reward. But inside of '`_claimStaked`' function instead of user address referrer address is given again.

Note: it doesn't concern internal logic of smart-contract, events are used to display data on the website.

2. Unused variables (medium severity):

'`safeTokenTransfer`' and '`safeSendValue`' functions meaning is to decrease sending amount to balance of contract if amount exceeds balance. But actually modified 'amount' variable is missed and initial '`_amount`' parameter is sent. So using both functions makes no sense.

3. Level limit inaccuracy (low severity):

There is 10 level limits `(LEVEL_LIMIT)` for minting tokens by different yield `(LEVEL_YIELD)`. The sum of each limit is the maximum supply of the token (27.5M). Actually if you update pending minted amount too late the current minted amount `(_mintedCurrentLevel)` of each level can be more than limit.

It won't lead to breaking the limit of total supply, but it makes logic of yields not exact.

4. False requirement (low severity):

There is requirement for the owner that he cannot stake, but he can get around this requirement by temporarily transferring ownership to other account.

Note: owner cannot take advantage from this action.

OPTIMIZATION POSSIBILITIES

1. Excessive variable (very low severity):

There is '`addr`' parameter to store user's address inside of '`User`' struct. It can be totally removed since every use of this parameter can be replaced with simple '`msg.sender`'.

2. Excessive if/else statement (very low severity):

Zero-check of '`_referrer`' address parameter is excessive since zero address will not pass next exist-check. It can be removed.

3. Recording statistical parameters in the blockchain (very low severity):

There is 1 statistical variable '`totalReferrals`' that increases the cost of transactions and increase the amount of data stored in the blockchain:

Recommendation: use events and log this information instead of writing it to the blockchain.

Note: this comments do not affect the main functionality of the smart-contract and concern only cost of transactions.

NOTES AND RECOMMENDATIONS

1. Inside of '`_mintTokens`' function '`toMint`' variable can be modified in case to prevent exceeding of maximum supply of token, but '`_mintedCurrentLevel`' is recorded before that.

Recommendation: move 204 line of code to 210-211.

2. Usually memory variable '`user`' is used to fetch user's data. But at the 262 line of code fetches '`_users[_address].lastClaim`'.

Recommendation: use '`user.lastClaim`' instead.

3. '`unstake`' function sends token profit and staked amount to user, but leaves referral bonuses.

Recommendation: add referral bonus withdraw to '`unstake`' function.

Independent description of the smart-contract functionality:

APEMaster smart-contract provides opportunity to stake TRX to get APE tokens.

APE Token address:
https://tronscan.org/#/token20/TJNqwS7BJm1SFwnAn7gvJkZcB3QDhPHChe

The maximum total supply of the APE token is 27.5 millions APE.

There was no pre-minting of this token and any additional minting of tokens (other than from only staking contract) is impossible.

You can stake TRX using '`stake`' function and attaching any amount of TRX (more than 100). If new subsequent investment will be made - available reward is automatically withdrawn to wallet.

There is one-level referral program so at the time of first investment you can specify your referrer using '`stake(address _referrer)`' function.

If no referrer was given or referrer is not registered in contract before - the owner is set as referrer.

To withdraw your current available referral bonuses use '`claimReferralReward()`' function.

To get your investment back (excluding 10% fee) use '`unstake`' function. Also you can withdraw part of your investment using '`unstake(uint256 _amount)`' function. Current profit will be withdrawn at this actions.

The owners commission is 10% of total staked TRX value of each investment. To withdraw available commission owner uses '`withdrawFees`' function that is allowed only to the owner. Owner cannot withdraw more than available fee.

There is also '`emergencyWithdraw`' function to let every user claim his deposit without receiving token reward. It is made to prevent losses if there is something wrong with tokens but actually there is no need to use this function. If user calls this function he will receive deposit (excluding 10% fee), but will lose all token reward forever.

Token reward is calculated as follows*:

Every range (limit) of tokens (for example first - 5 millions) tokens are minted according the profitability ('yield') (amount of tokens minted per 1 staked TRX per 1 day).

Limit	5M	4.5M	4M	3.5M	3M	2.5M	2M	1.5M	1M	0.5M
Yield	0.019	0.018	0.017	0.016	0.015	0.014	0.013	0.012	0.011	0.010

* - there is some inaccuracy (see Bugs and errors section of audit).

Since any modifying state of blockchain is impossible without transaction initiated by somebody, needed tokens are minted at every action inside of smart-contract.

To get your available token reward use '`claimStaked()`' function.

The contract contains statistical functions that do not require sending transaction:

1. `pendingReward()` or `pendingReward(address _address)` - pending token reward of user.

2. `stats()` - current level of minting, yield of level, limit of level, minted amount of level, total APE supply, total TRX staked now, current owners fee available to withdraw.

3. `user()` or `user(address _address)` - investment, last claim of reward UNIX time, referrer address, available referral reward, total amount of referrals, pending reward, APE token balance of user and TRX balance of user.

December 2020

Disclaimer:

This audit is not a call to participate in the project and applies only to the Smart-Contract code at the specified address.

Do not forget that you are doing all financial actions at your own risk, especially if you deal with high-risk projects.

Warning:

Beware of fake audits.
All official info available on 3 resources only:

Website: **www.grox.solutions**
Telegram: **www.t.me/groxsolutions (@groxsolutions)**
YouTube: **www.youtube.com/c/groxsolutions**

If you have any questions or are interested in developing/auditing of Smart-Contracts, please contact us and we will consult you.

Telegram: **www.t.me/gafagilm (@gafagilm)**
E-mail: **info@grox.solutions**